

IBM i DB2 and SQL School

(Course code OD47)

Instructor Guide

ERC 7.0



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AS/400®	DB™	DB2 Universal Database™
DB2®	Domino®	DRDA®
Integrated Language Environment®	iSeries®	i5/OS™
Language Environment®	Notes®	OmniFind®
OS/400®	Power Systems™	Power Systems Software™
Power®	Rational®	Redbooks®
RPG/400®	Solid®	System i®
System p®	System x®	VisualAge®
400®		

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

May 2013 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 2010, 2013.
This document may not be reproduced in whole or in part without the prior written permission of IBM.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Trademarks xv

Instructor course overview xvii

Course description xix

Agenda xxiii

Unit 1. Introduction to relational database management and DB2 for i 1-1
- Unit objectives 1-2
- DB2 for i 1-4
- Hierarchical database 1-6
- The ease of working with a relational database management system 1-8
- The relational database model 1-10
- IBM's DB2 family 1-12
- DB2 for i: ibm.com/systems/i/software/db2/ 1-14
- DB2 for i 1-16
- IBM i architecture 1-19
- IBM i: Integrated features 1-21
- Power + IBM i architecture 1-23
- Database architecture 1-25
- DB2 for i: XML support 1-27
- DB2 for i: Additional enhancements 1-29
- IBM i storage management enhancements for SSD 1-31
- IBM i objects 1-34
- Traditional program data definition 1-36
- Externally described IBM i file 1-38
- Externally described file example 1-40
- Externally described file components 1-42
- Legacy coding to define database file CLASSPF 1-44
- IBM i library 1-46
- Libraries 1-48
- Library/object structure 1-50
- IBM i objects 1-52
- Working with relational DBMS 1-54
- Sequence 1-56
- Select 1-58
- Project 1-60
- Union 1-62
- Inner join 1-64
- Left outer join 1-66
- SQL versus native IBM i object terminology 1-68
- More terminology 1-70
- Checkpoint 1-72

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary 1-74

Unit 2. Introducing Structured Query Language. 2-1
Unit objectives 2-2
2.1. Introducing SQL 2-5
Topic 1: Introducing structured query language 2-6
History of SQL 2-9
Some forms of SQL 2-12
SQL packaging 2-15
SQL components 2-19
SQL terms 2-23
Checkpoint 2-27
Exercise 2-29
Unit summary 2-31

Unit 3. Extracting data using SELECT 3-1
Unit objectives 3-3
3.1. Creating the database. 3-7
Topic 1: Creating the database 3-8
SQL components 3-10
Create 3-12
Create schema 3-15
Table: Cars 3-17
Creating a table: Things to consider 3-20
Create table 3-24
Create table members 3-27
Labels and column headings 3-29
Create view 3-31
Create index 3-33
Composite key 3-35
INSERT statement: Add rows to table 3-37
3.2. SQL data types 3-41
Topic 2: SQL data types 3-42
Value 3-44
Nulls 3-46
Data types 3-48
Numeric data types 3-50
Alphanumeric data types 3-53
Graphics data type 3-55
Date data type 3-57
Time data type 3-59
Timestamp data type 3-61
Other data types 3-63
3.3. The SELECT statement 3-65
Topic 3: The SELECT statement 3-66
SELECT statement 3-68
Table: CARS 3-71
Table: Members 3-74

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT all data . . . 3-76
SELECT some columns . . . 3-79
Deriving a new column . . . 3-81
ORDER BY (1 of 2) . . . 3-83
ORDER BY (2 of 2) . . . 3-85
Ascending or descending sequence . . . 3-88
Exercise . . . 3-90
DISTINCT . . . 3-92
WHERE column-name clause . . . 3-94
Conditions in WHERE clause . . . 3-96
Selecting on conditions in the WHERE clause . . . 3-98
WHERE: Other operators . . . 3-101
Less than, greater than, equal . . . 3-103
AND, OR . . . 3-105
Parentheses with multiple conditions . . . 3-107
Other WHERE clause considerations . . . 3-110
WHERE NOT (1 of 2) . . . 3-112
WHERE NOT (2 of 2) . . . 3-114
BETWEEN x AND y . . . 3-116
IN (x, y, z) . . . 3-118
LIKE (1 of 3) . . . 3-120
LIKE (2 of 3) . . . 3-122
LIKE (3 of 3) . . . 3-124
GROUP BY . . . 3-126
HAVING . . . 3-128
FETCH FIRST n ROWS ONLY . . . 3-131
3.4. Using System i Navigator . . . 3-133
Topic 4: Using System i Navigator . . . 3-134
System i Navigator overview . . . 3-136
Open target system . . . 3-138
Properties notebook for System i connection . . . 3-140
Connection supports database . . . 3-143
Open schema . . . 3-145
Display schema objects . . . 3-147
View table contents . . . 3-149
Run SQL Script (1 of 2) . . . 3-151
Run SQL Script (2 of 2) . . . 3-153
SQL statement examples . . . 3-155
Open JDBC properties notebook . . . 3-157
Set naming convention . . . 3-159
Enter SQL statement and run . . . 3-161
Results viewer . . . 3-164
Using SQL Assist . . . 3-166
FROM clause . . . 3-168
SELECT clause . . . 3-170
WHERE clause . . . 3-173
GROUP BY clause . . . 3-175
HAVING clause . . . 3-177

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

ORDER BY clause 3-179
The SELECT statement that was generated 3-181
3.5. Using the 5250 STRSQL interface 3-183
Topic 5: Using the 5250 STRSQL interface 3-184
STRSQL: Prompting with F4 3-186
STRSQL entry 3-189
Session defaults are important 3-191
Change defaults 3-194
Change session attributes 3-196
Rerun query 3-198
Exit and save session in source file 3-200
Saving in source file 3-202
Contents of source file 3-204
Cleanup and create SQL script 3-207
RUNSQLSTM 3-209
RUNSQLSTM steps 3-211
RUNSQL 3-213
Checkpoint (1 of 2) 3-215
Checkpoint (2 of 2) 3-217
Exercise 3-219
Unit summary 3-221

Unit 4. Functions 4-1
Unit objectives 4-2
4.1. Column functions 4-5
Topic 1: Column functions 4-6
Column functions 4-8
Column functions and calculated values 4-10
COUNT 4-12
AVG 4-14
MIN/MAX 4-16
SUM 4-18
4.2. Scalar functions 4-21
Topic 2: Scalar functions 4-22
Scalar functions 4-24
Substring character columns 4-27
CONCAT 4-29
STRIP 4-31
LTRIM and RTRIM 4-34
UPPER and TRANSLATE 4-36
Conversion functions 4-38
DECIMAL 4-40
CHAR 4-43
MOD 4-45
MIN and MAX 4-48
IFNULL, VALUE, and COALESCE 4-50
Date and time 4-52
Date/time scalar functions (1 of 3) 4-54

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date/time scalar functions (2 of 3) 4-56
Date/time scalar functions (3 of 3) 4-58
Date function samples 4-60
EXTRACT and MONTHNAME 4-62
YEAR 4-64
Special registers 4-67
4.3. CASE and CAST 4-69
Topic 3: CASE and CAST 4-70
CASE expressions (1 of 3) 4-72
CASE expressions (2 of 3) 4-75
CASE expressions (3 of 3) 4-77
CAST expressions 4-80
CAST examples 4-83
Checkpoint (1 of 2) 4-85
Checkpoint (2 of 2) 4-87
Exercise 4-89
Unit summary 4-91

Unit 5. JOIN and UNION 5-1
Unit objectives 5-2
5.1. Using JOIN 5-5
Topic 1: Using JOIN 5-6
Combining data using JOIN 5-8
Types of JOINs 5-10
INNER JOIN 5-13
INNER JOIN with USING clause 5-17
JOIN exercise 5-20
Correlation names 5-23
Add tables to the car club 5-26
Multi-table JOINs 5-29
Join of three tables using ON 5-31
Join of three tables using WHERE 5-34
LEFT OUTER JOIN 5-36
EXCEPTION JOIN 5-38
5.2. UNION 5-41
Topic 2: UNION 5-42
UNION of result tables 5-44
UNION and UNION ALL 5-47
Using dummy columns 5-49
UNION plus text column 5-51
UNION with ORDER BY 5-53
EXCEPT 5-55
INTERSECT 5-58
Checkpoint 5-61
Exercise 5-63
Unit summary 5-65

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 6. SubSELECTs 6-1
Unit objectives 6-2
6.1. Using subSELECTs and subqueries. 6-5
Topic 1: Using subSELECTs and subqueries 6-6
What is a subSELECT? 6-8
Why subSELECT? 6-10
Using subSELECT 6-12
Predicates and subqueries 6-15
Subquery with basic predicate 6-18
Subquery with the IN predicate (1 of 3) 6-21
Subquery with the IN predicate (2 of 3) 6-23
Subquery with the IN predicate (3 of 3) 6-25
IN predicate used with nullable column (1 of 2) 6-28
IN predicate used with nullable column (2 of 2) 6-30
Subquery with ALL predicate (1 of 2) 6-32
Subquery with ALL predicate (2 of 2) 6-34
Subquery using ANY or SOME (1 of 2) 6-37
Subquery using ANY or SOME (2 of 2) 6-39
Correlated subquery (1 of 3) 6-42
Correlated subquery (2 of 3) 6-44
Correlated subquery (3 of 3) 6-46
Subquery with EXISTS predicate (1 of 2) 6-49
Subquery with EXISTS predicate (2 of 2) 6-51
Scalar subselect 6-54
Checkpoint 6-56
Exercise 6-58
Unit summary 6-60

Unit 7. Query tools 7-1
Unit objectives 7-2
7.1. Using Query Manager. 7-5
Topic 1: Using Query Manager 7-6
Features of Query Manager 7-8
Start Query Manager: STRQM 7-11
Query Manger Profile 7-14
Work with tables 7-17
Work with Query Manager Queries 7-20
Edit Query 7-23
Run Query 7-26
Display Report 7-28
Exit by pressing either F3 or Enter 7-30
F13 Edit form > Edit column formatting 7-33
Edit Column Formatting 7-35
F4 > Select data type 7-38
Select Column Editing 7-40
Edit Column Formatting 7-42
Select Report Format > Edit page headings 7-44
Specify Page Heading 7-46

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

F3 > Save completed form 7-49
Option 9 = Run Query 7-51
Query final result 7-53
7.2. Using Query Manager procedures 7-55
Topic 2: Using Query Manager procedures 7-56
Query Manager procedures 7-58
Run Query Manager procedure 7-61
Display your output 7-63
Other QM features 7-65
Checkpoint 7-68
Exercise 7-70
Unit summary 7-72

Unit 8. Database maintenance. 8-1
Unit objectives 8-2
8.1. SQL data definition and manipulation 8-5
Topic 1: SQL data definition and manipulation 8-6
SQL components 8-8
Create review 8-10
Create table members 8-11
Column names 8-13
Labels and column headings 8-15
Alter table 8-17
Inserting rows using a SELECT statement 8-19
UPDATE 8-22
Advanced UPDATE techniques 8-25
DELETE 8-28
DROP 8-30
Exercise 8-32
8.2. Data integrity 8-35
Topic 2: Data integrity 8-36
Data integrity 8-38
GRANT and REVOKE 8-41
Check constraints 8-43
Defining check constraints 8-46
Referential integrity 8-48
8.3. SQL SYS objects. 8-51
Topic 3: SQL SYS objects 8-52
SQL catalog 8-54
SQL collection catalog views 8-56
Using SQL collection catalog views 8-59
Impact of changing a column 8-61
8.4. SQL objects exercise environment 8-63
Topic 4: SQL objects exercise environment 8-64
TESTEMP table 8-66
Create table TESTEMP 8-68
Create view PERSINFO 8-70
Inserting rows into the table 8-73

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Insert multiple rows 8-75
Update columns (1 of 2) 8-77
Update columns (2 of 2) 8-79
Delete rows 8-81
Drop table TESTEMP 8-83
Reminders 8-85
Checkpoint 8-87
Exercise 8-89
Unit summary 8-91

Unit 9. Using embedded SQL 9-1
Unit objectives 9-3
9.1. Introducing embedded SQL 9-7
Topic 1: Introducing embedded SQL 9-8
Embedded SQL: Some questions 9-10
Coding choices 9-12
Why use embedded SQL? 9-15
What is embedded SQL? 9-18
Inquiry program: Native I/O 9-21
Inquiry program: SQL I/O 9-25
Program compilation 9-27
RPG IV ILE preprocessor 9-30
Embedded SQL: Under the covers (1 of 2) 9-33
Embedded SQL: Under the covers (2 of 2) 9-35
SQL Communications Area 9-37
SQL Descriptor Area 9-39
Exercise 9-42
9.2. Writing programs that use embedded SQL 9-45
Topic 2: Writing programs that use embedded SQL 9-46
Languages supported 9-48
Delimiting SQL statements 9-50
V5R4 free-form SQL statements 9-53
Host variables 9-55
SQL mapped to host variable data type 9-58
SELECT one row of a table 9-62
SELECT INTO host variable 9-64
Using structure as host variable 9-67
SQL precompiler 9-69
SQL precompiler: Parameters (1 of 4) 9-72
SQL precompiler: Parameters (2 of 4) 9-75
SQL precompiler: Parameters (3 of 4) 9-77
SQL precompiler: Parameters (4 of 4) 9-79
SQL Communication Area 9-81
Output of precompiler (1 of 2) 9-83
Output of precompiler (2 of 2) 9-85
9.3. Error handling 9-87
Topic 3: Error handling 9-88
What is available? 9-90

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Communication Area 9-92
SQLCODE and SQLSTATE 9-95
SQLCODE and SQLSTATE 9-98
Examples of SQLCODE and SQLSTATE 9-100
SQL state error classes 9-102
Monitoring for errors using WHENEVER 9-105
Scope of WHENEVER 9-107
Using SQLCODE (1 of 4) 9-109
Using SQLCODE (2 of 4) 9-111
Using SQLCODE (3 of 4) 9-114
Using SQLCODE (4 of 4) 9-116
Error handling: Considerations 9-118
Handling SQL errors using SQLCODE 9-120
Handling SQL errors using SQLSTATE 9-122
Subroutine or CallP error procedure 9-124
Using indicator variables 9-126
Exercise 9-129
9.4. Using a cursor for multiple row result set 9-131
Topic 4: Using a cursor for multiple row result set 9-132
Types of embedded SQL 9-134
SELECT multiple rows of a table 9-136
Using a serial cursor (1 of 5) 9-139
Using a serial cursor (2 of 5) 9-141
Using a serial cursor (3 of 5) 9-143
Using a serial cursor (4 of 5) 9-145
Using a serial cursor (5 of 5) 9-147
UPDATE / DELETE current row 9-149
FOR UPDATE OF clause 9-151
Using SCROLLable cursors 9-153
Blocked fetch 9-156
CLOSE the cursor 9-158
Exercise 9-161
9.5. Dynamic embedded SQL 9-163
Topic 5: Dynamic embedded SQL 9-164
Types of embedded SQL 9-166
Types of dynamic SQL 9-168
Static SQL versus dynamic SQL 9-170
Dynamic SQL statements 9-172
Dynamic non-SELECT: No parameter markers 9-175
Dynamic non-SELECT: Parameter markers (1 of 2) 9-178
Dynamic non-SELECT: Parameter markers (2 of 2) 9-180
Dynamic fixed-list SELECT with parameter markers (1 of 2) 9-182
Dynamic fixed-list SELECT with parameter markers (2 of 2) 9-185
Dynamic varying list SELECT 9-187
Dynamic varying list SELECT: The process 9-189
Using the SQLDA (1 of 2) 9-191
Using the SQLDA (2 of 2) 9-193
SQLDA data structure 9-196

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint 9-199
Exercise 9-201
Unit summary 9-203

Unit 10. An overview of stored procedures 10-1
Unit objectives 10-2
10.1. Introducing stored procedures 10-5
Topic 1: Introducing stored procedures 10-6
Why CALL? 10-8
What is a stored procedure? 10-10
Call from client: Without stored procedure 10-12
Call from client: With stored procedure 10-15
Call external stored procedure 10-18
Create external stored procedure 10-20
Parameter styles 10-23
SQL stored procedures 10-26
10.2. External stored procedures 10-29
Topic 2: External stored procedures 10-30
Developing an external stored procedure 10-32
Stored procedure to call report program (1 of 3) 10-34
Stored procedure to call report program (2 of 3) 10-36
Stored procedure to call report program (3 of 3) 10-38
Stored procedure with parameter (1 of 3) 10-40
Stored procedure with parameter (2 of 3) 10-43
Stored procedure with parameter (3 of 3) 10-46
Exercise 10-49
10.3. SQL stored procedures 10-51
Topic 3: SQL stored procedures 10-52
SQL stored procedures 10-54
Code and create SQL stored procedure 10-56
Call SQL stored procedure 10-59
Result set 10-61
Logic flow control in stored procedures 10-63
Logic flow example in stored procedure (1 of 2) 10-66
Logic flow example in stored procedure (2 of 2) 10-68
What's next? 10-70
Checkpoint 10-73
Unit summary 10-75

Unit 11. Database tools with System i Navigator 11-1
Unit objectives 11-2
How do I access the database tools? 11-4
Which tool should I use? 11-7
About System i Navigator 11-9
How do I get database fixes? 11-11
How do I start the database? 11-13
Explorer-like interface 11-15
Database object creation 11-18

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Run SQL Scripts 11-20
Save results (1 of 3) 11-22
Save results (2 of 3) 11-24
Save results (3 of 3) 11-26
Show SQL statement from create dialogs 11-28
Keeping schema source 11-30
Generate the SQL 11-32
Data dictionaries 11-34
Journaling 11-36
Table description 11-38
Table description activity 11-40
How do I synch up my db and user IDs? 11-42
New in 7.1: Database Maintenance folder 11-44
Constraint repair 11-46
Index builds in progress 11-48
Right-click build in progress 11-50
Alter table status 11-52
Reorganize table 11-54
Reorganize table progress monitoring 11-56
From the maintenance folder 11-58
Reorganization status display 11-60
Reorganize index build phase 11-62
Manage index rebuilds 11-64
Health Center 11-66
Environmental limits 11-68
Health Center 11-70
Additional activity counters in i 7.1 11-73
Work management for database jobs 11-75
SQL details for jobs 11-77
SQL Details for Jobs 11-79
Who has those table rows locked? 11-81
Live analysis: Index Advisor 11-83
Index Advisor: Scroll to the right 11-85
Live analysis: SQL Plan Cache Statements 11-87
SQL Performance Monitor toolset 11-89
SQL Performance Monitor filtering 11-91
SQL Performance Monitor analysis 11-93
SQL Performance Monitor comparison 11-96
Analyze SQL Performance Monitors with Visual Explain 11-98
Visual Explain 11-100
Visual Explain while running 11-102
Index Advisor 11-104
Statistics 11-106
Statistics requests running in the background 11-108
Stats requests for QDBFSTCCOL job 11-110
Show indexes 11-112
Evaluating your indexes and MQTs 11-114
Enhanced index management 11-117

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Analyze performance of SQL programs and packages 11-119
Checkpoint 11-121
Unit summary 11-123

Appendix A. Checkpoint solutions A-1

Appendix B. SQL terms. B-1

Bibliography X-1

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AS/400®	DB™	DB2 Universal Database™
DB2®	Domino®	DRDA®
Integrated Language Environment®	iSeries®	i5/OS™
Language Environment®	Notes®	OmniFind®
OS/400®	Power Systems™	Power Systems Software™
Power®	Rational®	Redbooks®
RPG/400®	Solid®	System i®
System p®	System x®	VisualAge®
400®		

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor course overview

This course is designed mainly for IBM i programmers but some *programming oriented* users might be in your audience as well.

This class will teach the students how to use the SQL language interactively to access data on the IBM i.

This course is the course that IBM i customers can attend to learn about the different elements of DB2 for i, SQL, and the ways that both can be used. Comprehensive workstation exercises complement the lecture material. A schema and a number of tables will be built by the students. Then the students will write the SQL statements to retrieve and manipulate data in their individual schemas.

Since this course is offered to users as well as programmers, some students who have limited programming experience might require assistance if they are challenged by the programming nature of SQL.

A number of tables are used for examples in lecture as well as in the lab exercises. These tables are included in the master student schemas OL37XXCOL and OL38XXCOL. The instructor should be very familiar with the columns as well as the data values when you teach the class. I have tried to place copies of the record formats as well as the tables' values in the notebook whenever they are used in order to allow easy reference to the course database.

The agenda will fill four and a two thirds days completely. You should plan for all but the last day to end between 4:00 and 4:30 p.m. Some students will require more time than is scheduled to perform the machine exercises. Your only choices are to offer additional lab time after class on the last day or before class on the days succeeding the first day. Students will usually ask when class is scheduled to end; the safest thing to do is to tell them to plan to depart mid-afternoon on the last day.

Note: We have combined DB2 for i and SQL material, some of which might not be of major interest to non-programmers nor non-database administrators. A few of the topics rely upon programming concepts and knowledge that some of your students might not possess. The estimated topic timings and the level of topic detail has compensated for these possibilities. However, you must still carefully manage the agenda. For the material on functions and sub-selects, embedded SQL statements, stored procedures and System i Navigator database tools, use your judgment and measure of your audience as guides. The unit on database maintenance, while interesting only to some students, serves as an excellent recap of important topics covered in

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

the course and serves as an excellent self-evaluation for your students. While you cover this unit and its exercises, remind your students of this point and suggest that they complete the exercises as best they can.

This course also focuses on two facilities of SQL that are useful in application development:

1. Embedded SQL
2. SQL stored procedures

The **embedded SQL** unit begins by showing the students how they can simply replace existing native I/O statements in an RPG IV program with embedded SQL statements that perform the same function or do more.

As you present SQL statements such as DELETE or UPDATE, stress that they look like the RPG IV opcodes but have a much broader scope and must therefore be carefully managed.

Error handling from a basic point of view is discussed, but those who want more detail should plan to attend the **OL39/OL390** class.

Stored Procedures includes a simple lab that the students should complete prior to leaving the class. At a minimum, the instructor should perform the exercise as a demonstration to the students. Stored Procedures enable better handling of transactions between a client application and the IBM i server.

Course strategy

This version of the course supports V7R1 of the IBM i operating system. There are several new features in this release that are included in the materials including new support of expressions in the LIKE clause, EXCEPT, INTERSECT.

With this release, you should have your students at least try using System i Navigator as a development tool, especially Run SQL Scripts and its SQL assist tool. Legacy 5250 interfaces will work but they will not be enhanced and will not support new functions. This is not to say, for example, that LIKE support of expressions will not work in STRSQL, but, the prompter will not support some features of the language.

For ILO delivered classes, if you have access to an enabled web camera, please use it to display yourself during your introduction in the welcome/administration unit of the class. It's also advised that you use your web camera at the close of the class when seeking students' evaluations of their experiences.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Course description

IBM i DB2 and SQL School

Duration: 5 days

Purpose

SQL is a very powerful programming and database language that is widely used in business today. For example, SQL can be used to perform interactive database queries as well as to extract data from a server and pass that data to a Java applet running on a client PC. SQL is frequently used in i system high-level languages as an alternate means to access the database.

This course is designed to teach i programmers and highly skilled i system users how to access the database using SQL. Students will learn how to create, maintain, and query the database using the interactive SQL interface and System i Navigator.

Audience

This course is designed primarily for experienced IBM i programmers. However, very skilled i system users who want to learn how to use the SQL programming language as a means to access the DB2 for i database might also consider attending this class.

This course is not recommended for users who need to perform simple queries. Query for IBM i is an end user oriented tool designed for this purpose.

This course is not recommended for users who need to perform WebQuery queries. WebQuery for IBM i is an end user, programmer and administrator oriented tool designed for this purpose. You can learn about WebQuery for IBM i by attending such courses as OV550 - *DB2 WebQuery for Advanced Users, Developers and Administrators* - ILO or OD050, the classroom equivalent.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Prerequisites

Before attending this class, the student should be able to:

- Use a Windows-based PC
- Navigate and use a Windows-based desktop
- Use basic IBM i navigation tools, including:
  - CL commands
  - Menus
  - Online Help
  - `WRKSPLF` and related commands to manage output
  - `WRKJOB`, `DSPMSG`, `DSPJOB` commands and so forth to perform basic problem determination
- Use the Program Development Manager/Source Entry Utility or the RSE LPEX Editor to create and maintain source files
- List and explain the use of data types supported by the DB2 for i database

These skills can be gained by attending OL49/OL490 *IBM i Application Programming Facilities Workshop*.

Objectives

After attending this class, the student should be able to:

- List IBM i database objects
- Use both SQL and native i terminology to describe objects
- State two ways that SQL can be used in a program
- Use interactive SQL and System i Navigator to create DB2 objects
- Explain how SQL uses journaling and commitment control
- Use UPDATE, INSERT, ALTER, and DELETE to maintain an SQL database
- Use Data Definition Language (DDL) to describe tables and views
- Use UPDATE, INSERT, ALTER, and DELETE to modify SQL tables and maintain a database
- Use the SELECT statement to extract data from tables and views
- Code SQL queries that include column and scalar functions
- Code SQL queries that include inner and outer joins
- Use SQL Query Manager for i tool
- Describe how SQL stored procedures are created
- Describe and use the System i Navigator database tools

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Contents

Course units:

1. Introduction to relational database management and DB2 for i
2. Introducing Structured Query Language
3. Extracting data using SELECT
4. Functions
5. JOIN and UNION
6. SubSELECTs
7. Query tools
8. Database maintenance
9. Using embedding SQL
10. An overview of stored procedures
11. Database tools with System i Navigator

Curriculum relationship

Other courses that might enhance your knowledge are:

OL49/OL490 *System i Application Programming Facilities Workshop*

Assuming an IBM i programming background, a programmer would attend the database classes in this order:

OL62/OL620 *DB2 UDB for i Database Coding and Implementation Using DDS and CL Commands*

OD47/OD470 *DB2 for IBM i and SQL School or OL38 and OL39*

OL38/OL380 *Developing IBM i Applications Using SQL*

OL39/OL390 *DB2 UDB for i Database Advanced Programming*

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Agenda

Day 1

(00:30) Welcome and administration
(01:30) Unit 1. Introduction to relational database management and DB2 for i
- (00:15) Checkpoint

(01:15) Unit 2. Introducing Structured Query Language
- (00:30) Introducing SQL
- (00:15) Checkpoint
- (00:30) Workstation exercise: Getting to know SQL

(02:00) Unit 3. Extracting data using SELECT
- (00:15) Creating the database
- (00:15) SQL data types
- (00:15) The SELECT statement
- (00:30) Workstation exercise: How SELECT works
- (00:45) The SELECT statement (continued)

Day 2

(02:00) Unit 3. Extracting data using SELECT (continued)
- (00:30) Using System i Navigator (also known as IBM i Navigator)
- (00:15) Using 5250 STRSQL
- (00:15) Checkpoint
- (01:00) Workstation exercise: Using the SELECT statement

(01:50) Unit 4. Functions
- (00:15) Column functions
- (00:30) Scalar functions
- (00:15) CASE and CAST
- (00:05) Checkpoint
- (00:45) Workstation exercise: Using SQL functions

(01:35) Unit 5. JOIN and UNION
- (00:30) Using JOIN
- (00:30) UNION
- (00:05) Checkpoint
- (00:30) Workstation exercise: JOIN and UNION

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Day 3

(01:05) Unit 6. SubSELECTs
- (00:30) Using subSELECTs and subqueries
- (00:05) Checkpoint
- (00:30) Workstation exercise: Coding subSELECTs

(01:30) Unit 7. Query tools
- (00:45) Using Query Manager
- (00:45) Workstation exercise: Using SQL Query Manager

(03:30) Unit 8. Database maintenance
- (00:15) SQL data definition and manipulation
- (00:30) Workstation exercise: Enhancing an SQL table
- (00:30) Data integrity
- (00:15) SQL SYS objects
- (00:20) SQL objects exercise environment
- (00:10) Checkpoint
- (01:30) Workstation exercise: Maintaining database objects

Day 4

(05:10) Unit 9. Using embedded SQL
- (00:30) Introducing embedded SQL
- (00:45) Workstation exercise: Change CHAIN to SELECT
- (00:30) Writing programs that use embedded SQL
- (00:30) Error handling
- (00:45) Workstation exercise: Changing native to SQL I/O
- (01:00) Using a cursor for multiple row result set
- (00:10) Checkpoint
- (01:00) Workstation exercise: Using a cursor

Day 5

(03:00) Unit 9. Using embedded SQL (continued)
- (00:45) Dynamic embedded SQL
- (00:15) Checkpoint
- (02:00) Workstation exercise: Dynamic embedded SQL

(01:45) Unit 10. An overview of stored procedures
- (00:10) Introducing stored procedures
- (00:30) External stored procedures
- (00:30) Workstation exercise: Create a stored procedure
- (00:30) SQL stored procedures
- (00:05) Checkpoint

(01:45) Unit 11. Database tools with Systems i Navigator
- (01:40) Database tools with Systems i Navigator
- (00:05) Checkpoint

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 1. Introduction to relational database management and DB2 for i

Estimated time

01:30

What this unit is about

The DB2 for i is a relational database. This unit describes how it is implemented on the IBM i system.

The concept of a relational database is introduced, followed by a description of the IBM i database features.

Externally-defined files are described and compared to the more traditional structures of earlier systems. Physical files (tables), logical files (views and indexes), members, libraries, and schema are also discussed.

What you should be able to do

After completing this unit, you should be able to:

- Describe a relational database
- Describe features of DB2 for i
- Define basic IBM i database objects
- Compare native IBM i with SQL terminology

How you will check your progress

- Checkpoint questions

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe a relational database
- Describe features of DB2 for i
- Define basic IBM i database objects
- Compare native IBM i with SQL terminology

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

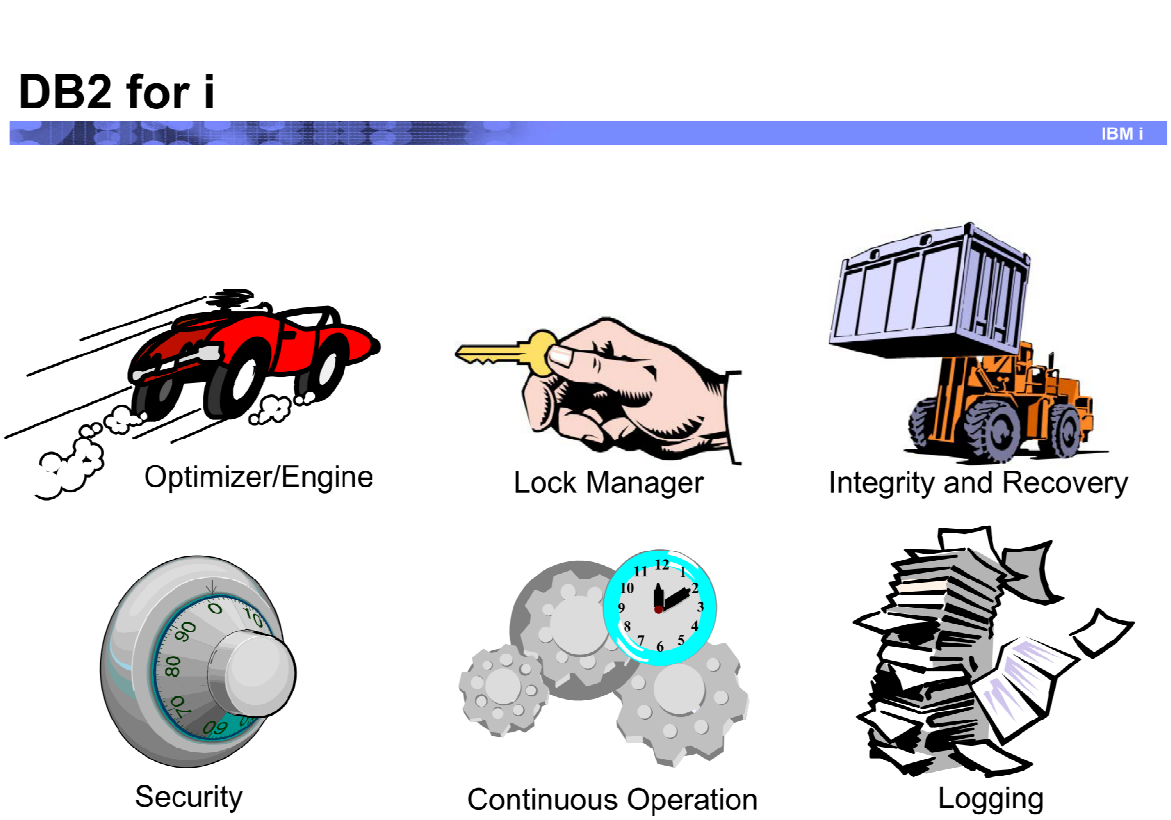
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-2. DB2 for i OD477.0

Notes:

A database management system provides a set of integrated data management services. The services might include but are not limited to:

- Concurrency
- Integrity
- Recovery
- Security
- Optimization
- Record locking
- Journaling and auditing

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Working with a Database Management System (DBMS) is like working with a group of highly trained professionals that provide a set of integrated and sophisticated data management services. Most database management systems have built into them the ability to allow multiple applications access to the same data at the same time. The ability to handle concurrency eliminates the requirement to replicate data or serialize work.

Database management systems have built-in functions to handle data integrity and recovery. For example, if an application's job is to transfer funds from a savings account to a checking account (a logical unit of work), the DBMS will ensure that the entire transaction is successful or, if something goes wrong (program "abends" (ends due to an error) after it has subtracted the money from the savings account but before it has added it to the checking account), the DBMS will put everything back the way it was before the transaction started. Furthermore, database management systems have a set of utilities that handle such things as backup and recovery.

Additional information — Database management systems come with some standard features that are not necessarily part of a file system. These include such things as:

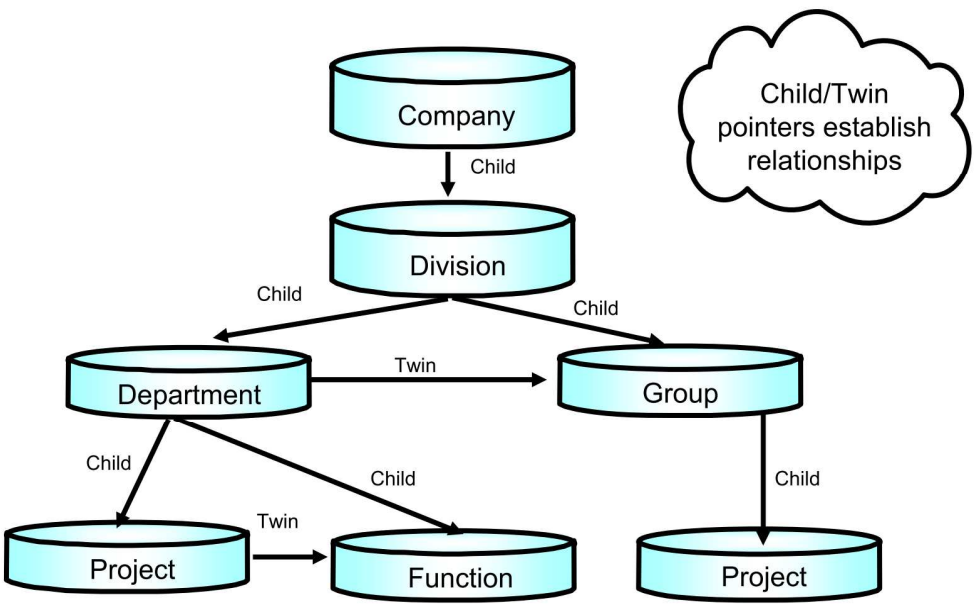
1. Security: A means of securing access to the data and other resources on a more granular basis than just the entire file.
2. Tracing facility: Produces a record of significant event within the DBMS. You might use this information for such things as billing, to identify poorly performing transactions, or to analyze system resource usage.
3. Features to enable continuous availability (24 X 7) so that the DBMS does not have to be stopped in order to add new users or resources or to change parameters that the database management system uses.

Transition statement — There are many types of database management systems. One of these types is called a hierarchical database management system.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Hierarchical database

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-3. Hierarchical database OD477.0

Notes:

Designed with a “parent/child” perspective, records from one file are related to records in another file through pointers. To find related records, we start by reading a record in one file and following its pointers to related records.

Hierarchical database management systems are complex to program for (so application development is slow), but some hierarchical database management systems perform very fast.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — A hierarchical database is used to manage files that contain related data. In this example, we see that we have a file that contains records organized by company. A record in the `Company` file may have one or more related records in the `Division` file. The data relationships are established with child pointers, and these pointers are embedded into every record in the database.

Additional information — To find a specific child record in the `Division` file for a specific parent record in the `Company` file requires finding the parent record and following its pointer to the division record. Application data may be accessed and viewed in a hierarchical manner, but most novices to data processing do not think in hierarchical terms. Learning to program and administer a hierarchical database management system is complex and requires months of training, and consequently, application development remains very slow and cumbersome.

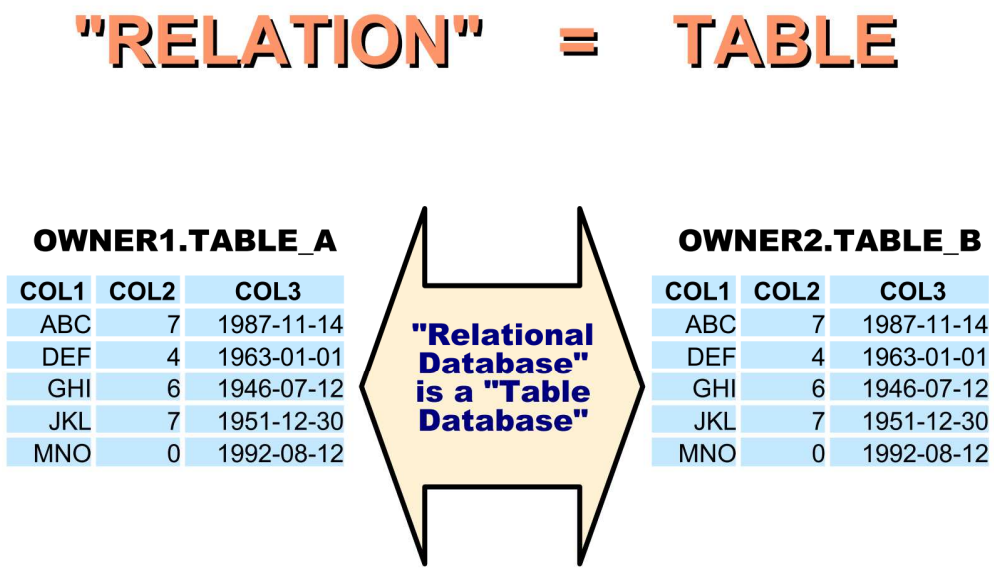
However, some hierarchical database management systems are very fast, and as a result of their tremendous processing speed, they continue to have a major presence in the data processing world.

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The ease of working with a relational database management system

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-4. The ease of working with a relational database management system OD477.0

Notes:

A relational database matches data by using common characteristics found within the data set. There are no embedded pointers in the data that establishes those relationships.

A relational database can be perceived as a set of tables and can be manipulated in accordance with the relational model of data. This relational model has a set of objects used to store, access, and manage data. The set of objects includes tables, views, and indexes.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The relational database model

IBM i

- All data items are in base tables.
- Views of the data in the base tables are created using the relational operators of sequence, select, project, union, and join.
- Any change to the data is immediately reflected in all the views.
- There are no embedded pointers or linkages in the data records/rows.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-5. The relational database model OD477.0

Notes:

These are the rules that relational databases follow.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IBM's DB2 family

IBM i

- Three code bases...
  - Based on the system history, architecture, and operating system
    - DB2 for i (System i, iSeries, and AS/400)
    - DB2 for z (System z, zSeries, and S/390)
    - DB2 for luw (Linux, UNIX, Windows , System p, and System x)

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-6. IBM's DB2 family OD477.0

Notes:

The IBM family of databases includes DB2 for i, DB2 for z and DB2 for luw (Linux, UNIX, Windows) for System p and System x.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

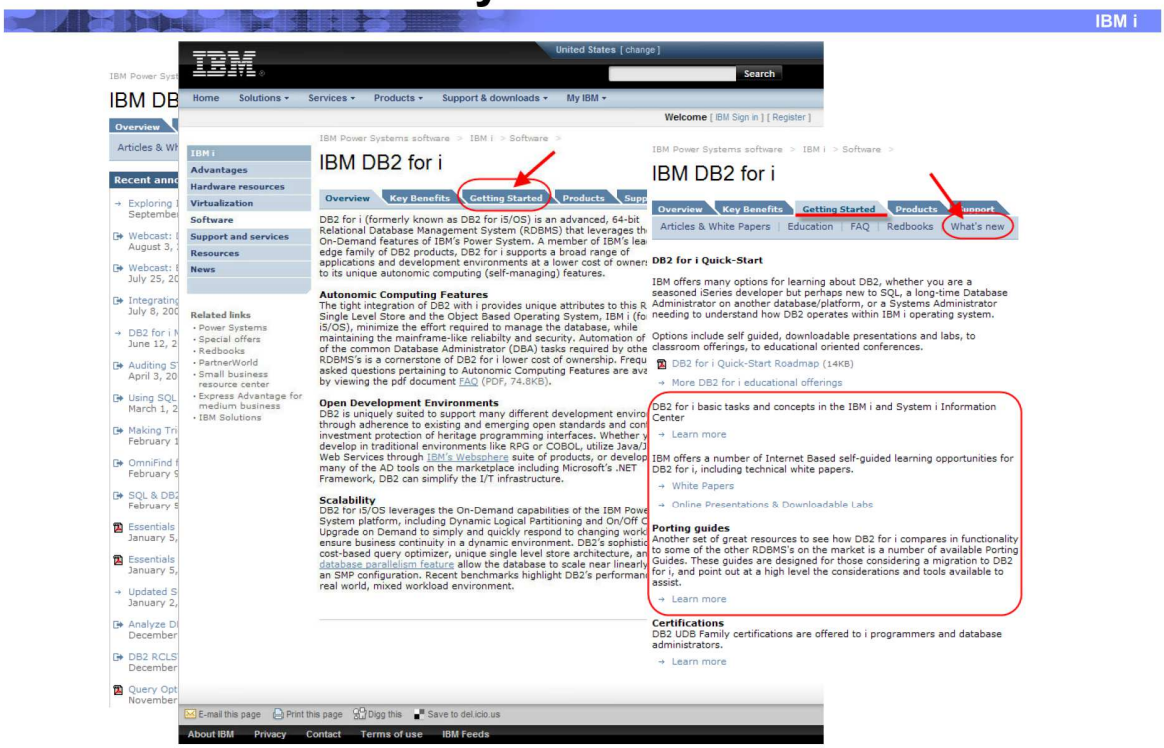
Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DB2 for i: ibm.com/systems/i/software/db2/



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-7. DB2 for i: ibm.com/systems/i/software/db2/ OD477.0

Notes:

Stay tuned to the DB2 for i web page for further enhancements and the latest features.
http://ibm.com/systems/i/software/db2/

This site provides key benefits, product information, getting started (including an education roadmap), and links to many technical white papers and articles, coding examples, and more.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DB2 for i

IBM i

- Fully relational DB integrated into IBM i since 1988
- Includes SQL interfaces and IBM i native record level access interfaces
- Integrated transaction management and security components
- DB2 objects are IBM i objects: Managed the same as other objects
- DB2 for i is a native file interface (store rows on and columns on disk)
- Ease of use advantages over traditional DBMS implementations
- Far less administration due to the integrated nature

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-8. DB2 for i OD477.0

Notes:

The integrated relational database on IBM i and its predecessors (AS/400, iSeries, and System i) has always been known for its ease of use and simplicity. The tight integration with the IBM i operating system allows DB2 for i to automatically take care of many lower-level tasks (such as storage allocation) that would normally require a database administrator (DBA) on other platforms. Security, reliability, scalability, data access speed, and audit-ability are other long-time strengths of the database that IBM continues to extend and leverage.

DB2 for i Version 7 Release 1 complies with the following IBM and Industry SQL standards:

- ISO (International Standards Organization) 9075: 2003, Database Language SQL – Core
- ISO (International Standards Organization) 9075: 2003, Database Language SQL – Entry Level
- ISO (International Standards Organization) 9075-3: 2003, Database Language SQL – Part 3: Call-level interface

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- ISO (International Standards Organization) 9057-4: 2003, Database Language SQL – Part 4: Persistent Stored Modules SQL/PSM)
- ANSI (American National Standards Institute) X3.135-2003, Database Language SQL – Core
- ANSI (American National Standards Institute) X3.135-3: 2003, Database Language SQL – Part 3: Call-level interface (SQL/CLI)
- ANSI (American National Standards Institute) X3.135-4: 2003, Database Language SQL – Part 4: Persistent Stored Modules SQL/PSM)

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — The following standards compliance has been included in the student notes for those who are interested:

DB2 for i Version 7 Release 1 complies with the following IBM and Industry SQL standards:

- ISO (International Standards Organization) 9075: 2003, Database Language SQL – Core
- ISO (International Standards Organization) 9075: 2003, Database Language SQL – Entry Level
- ISO (International Standards Organization) 9075-3: 2003, Database Language SQL – Part 3: Call-level interface
- ISO (International Standards Organization) 9057-4: 2003, Database Language SQL – Part 4: Persistent Stored Modules (SQL/PSM)
- ANSI (American National Standards Institute) X3.135-2003, Database Language SQL – Core
- ANSI (American National Standards Institute) X3.135-3: 2003, Database Language SQL – Part 3: Call-level interface (SQL/CLI)
- ANSI (American National Standards Institute) X3.135-4: 2003, Database Language SQL – Part 4: Persistent Stored Modules (SQL/PSM)

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM i architecture

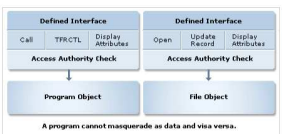
IBM i

Single level store



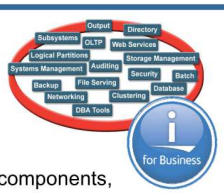
Automates and optimizes storage management

Object based architecture



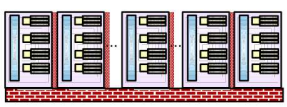
Enables integrity, security, and virus-resistance

Integration



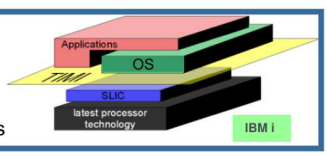
Integrates business components, for example DB2 database

Work management



Provides built-in application virtualization

Technology independent machine interface



Ensures application compatibility across multiple technology generations

An ***architecture*** devoted to business stability, simplicity, security, and scalability.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-9. IBM i architecture OD477.0

Notes:

The IBM i machine architecture is based on a high-level machine that implements many of the basic supervisory and resource management functions of operating systems and subsystems.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

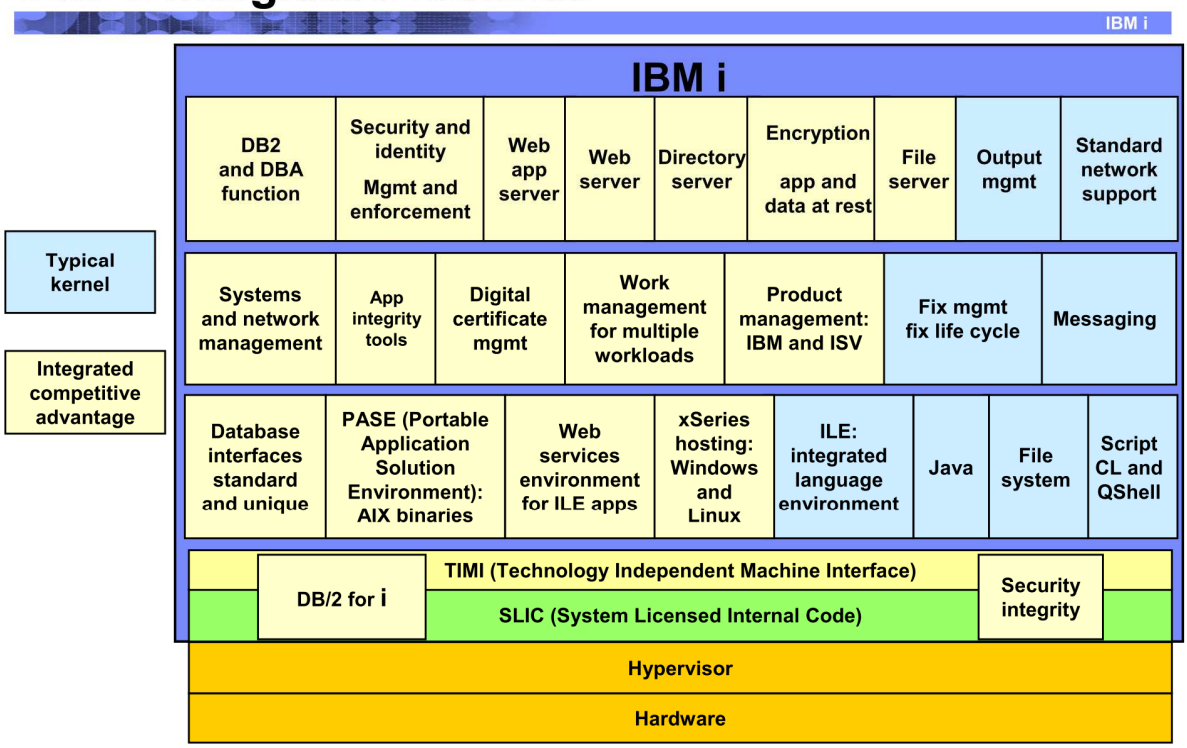
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IBM i: Integrated features



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-10. IBM i: Integrated features OD477.0

Notes:

IBM i has key features which are not available in a typical competitive or comparable IBM operating system. These functions are available as priced add-ons, or purchasable applications aside from IBM i.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

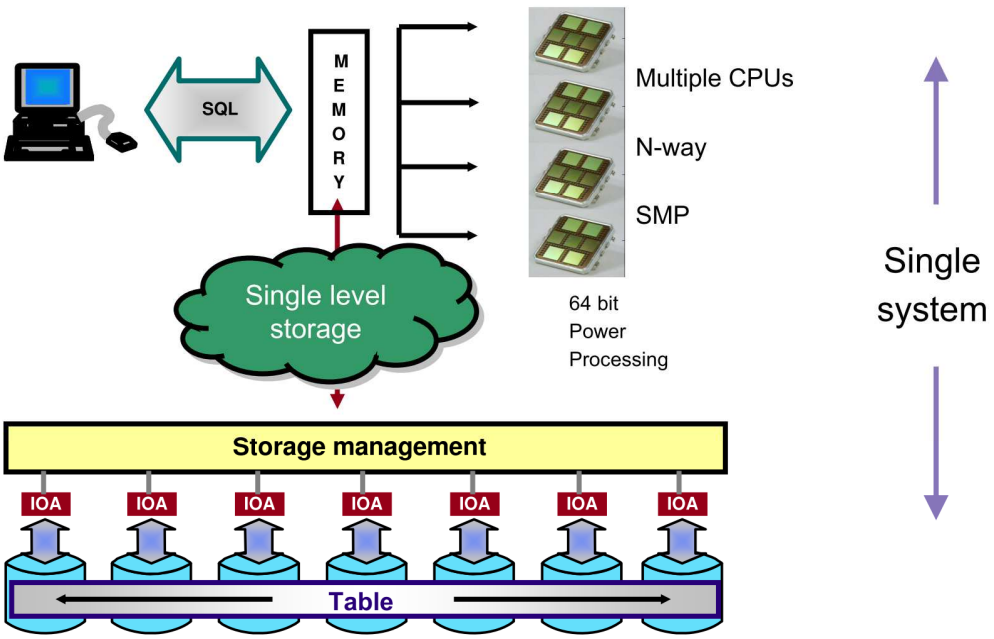
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Power + IBM i architecture

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-11. Power + IBM i architecture OD477.0

Notes:

This architecture has a positive affect upon SQL requests due to the optimization and implementation of the hardware.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — The unique IBM i architecture allows many implementation methods (table scan, table pre-load, SMP, and so on) that lead to optimum SQL performance. Cover the features presented in the chart from the bottom - up.

- Independent I/O subsystem / storage management
- Tables spread across all disk units (scatter loaded)
- Single Level Storage and large memory system
- 64 bit addressing, 40 G max main storage
- Multiple processors for N-way and SMP support
- Fourth generation 64 bit RISC

Additional information —

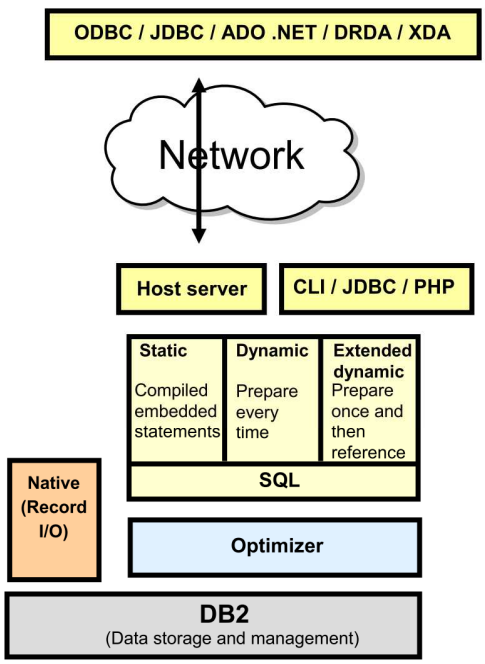
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Database architecture

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-12. Database architecture OD477.0

Notes:

Network requests for data take full advantage of database management's optimizer and query engine.

ADO = Active data objects (for example, OLE DB)

OLE = Object linking and embedding

ODBC = Open database connectivity

JDBC = Java database connectivity

CLI = Call level interface

Implemented using ODBC or directly to Host Server

ODBC/JDBC/ADO = client query program interfaces

CLI/JDBC = server query program interfaces

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Object linking and embedding (OLE) allows for accessing data (including, but not limited to SQL) from a variety of sources in a uniform manner. This illustrates the optimal manner in which the database can handle requests for data.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DB2 for i: XML support

IBM i

- XML is an industry standard for exchanging information between customers, suppliers, and partners.
- Rich XML support now available with DB2 for i
  - XML data type stores XML documents supporting database operations
  - Decompose (shred) XML documents into relational columns
  - Generate XML documents from existing relational data
- OmniFind Text Search Server provides support for searching XML documents
  - Search elements of an XML document (for example, customer name = Smith)
  - SQL statements use OmniFind to search the XML documents
  - Available with IBM i for no additional charge
- Strategic replacement for XML Extenders Program Product



PO #	Customer #	Date	Purchase Order
123	2468	5/27/09	~ XML ~
456	1357	6/10/09	~ XML ~

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-13. DB2 for i: XML support OD477.0

Notes:

In many applications today, business processes and logic are being controlled by data that is described in XML which is a standard way to describe information that is exchanged between customers and their suppliers and partners. That data was not immediately available to DB2 for i. The new enhancements in 7.1 make XML easier to process in DB2 for i by providing an XML data type that allows applications to store XML documents in DB2 and perform standard DB2 operations on those objects. You can also take existing data stored in DB2 and create an XML document from that data to make it easier to exchange information with partners using standard interfaces.

This is a strategic replacement for the XML Extenders product of previous releases.

The OmniFind Text Search Server makes it easy to search XML documents for specific information within a document such as a customer name. This is a separate product that is available with IBM i for no additional charge.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

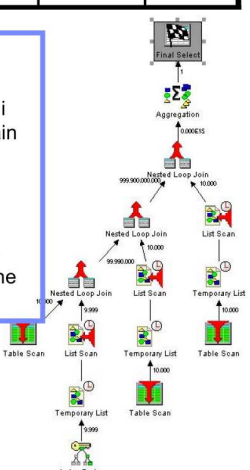
DB2 for i: Additional enhancements

IBM i

- Column Level Encryption
  - Allows for transparent (no application changes) encryption of a specific column in a database table accessed through SQL or native
  - Solutions from tool providers including Patrick Townsend, Linoma Software, and Protegrity supply encryption algorithms
- Application developers
  - MERGE, array support, global variables, and consuming result sets in RPG allows for more powerful and efficient programming
- Performance
  - Adaptive query processing can modify query plan while the query is running to significantly improve performance
  - Advanced SQE query optimizer now supports native logical files
- Management
  - New tooling to monitor long running operations
  - SQL_CANCEL procedure to cancel long running queries
  - Random or Sequential I/O statistics to identify tables that can benefit from SSDs

First Name	Last Name	City	State	Credit Car d#
Megan	Jones	Minneapolis	Minnesota	*&^%$*
Casey	Smith	Ames	Iowa	$%@^

On-Demand Performance Center
- Rich tools for DB2 for i
- Including Visual Explain
- Adaptive Query Processing can automatically build an index to replace table scan …taking queries from 30 minutes to one minute.



DB2 is an integrated database supporting transaction processing and business intelligence applications.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-14. DB2 for i: Additional enhancements OD477.0

Notes:

In addition to XML enhancements, there are many other DB2 enhancements for i 7.1. Column Level Encryption is one of those often requested features that is now available. DB2 provides the ability to encrypt individual columns for security purposes such as credit card numbers, and social security numbers. IBM has also partnered with the tool providers shown here, to supply encryption algorithms which can be used by application providers in order to take advantage of this support in their applications.

Additional enhancements for application developers, query performance improvements, and new support to monitor and manage queries is also provided in this release.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — The next visual highlights a benefit from i architecture coupled with the integrated database and storage management.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

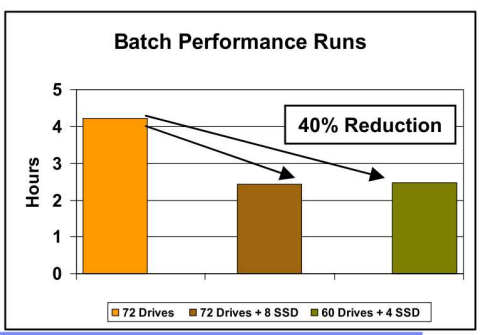
IBM i storage management enhancements for SSD

IBM i

- IBM i supports hierarchical storage management
  - Now IBM i automatically collects I/O performance data and moves most active data to Solid State Drives (SSD)
- DB2 for i supports SSD as preferred media
  - New DB2 Random Read Statistics
- Additional enhancements for SSDs
  - New *SSD-Aware* utilities
  - Improved performance instrumentation
  - Usability enhancements
- SSD Analyzer Tool
  - Designed to help determine if SSDs can help improve application performance
  - Runs on IBM i 5.4 or 6.1 system#



Reduces batch run time by 40% with SSDs*



SSDs can improve performance of long running batch jobs or queries. IBM i can easily get the right data on the SSDs.

*http://www.ibmsystemsmagpowersystemsibmidigital.com/nxtbooks/ibmsystemsmag/ibmsystems_power_200909/index.php#/16
Download http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3780

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-15. IBM i storage management enhancements for SSD OD477.0

Notes:

The storage management architecture of IBM i has made it easy to support Solid State Drives (SSDs). Since 2009, SSDs have offered tremendous performance improvements over spinning drives, but at a higher cost. It is important to know when and where to utilize SSDs in order to get the best return on investment. IBM i now makes this easier by providing a new hierarchical storage management feature which can automatically collect I/O performance data and move the most used data to the solid state drives where better performance is desired. Support for SSDs has been integrated with DB2 for i such that a DB2 file/table can be identified as a candidate to be placed on an SSD. More statistics are collected which can be used by a new SSD Analyzer Tool to determine if a system could significantly improve performance by using SSDs. This tool is also supported on i 5.4 and 6.1 systems. Experience with this technology has shown that customers can often benefit by replacing a small portion of their traditional disk drives with SSDs for data that is accessed frequently.

Here is a benchmarked example that illustrates how going from 72 spinning drives to 60 spinning drives and four SSDs resulted in a 40% reduction in the time it took to complete

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

an important batch run. For more information on the SDD Analyzer Tool for IBM I, visit the following technical document link:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3780

For more information on the SSD performance benchmark, refer to the following link:

http://www.ibmsystemsmagpowersystemsibmidigital.com/nxtbooks/ibmsystemsmag/ibmsystems_power_200909/index.php#/16

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — This is just another great example of how POWER architecture plus DB2 integration can increase system performance.

Details —

Additional information — The SSD Analyzer Tool is designed to help you determine if SSDs could help improve performance on your systems. The tool works with the performance data that is collected on your system and works with releases V5R4, V6R1, and V7R1. You will download a save file and restore the library QIBMSSD from the save file to your system. If you are currently at V5R4, you need to run a SETUP step before you start collecting performance data to use with the tool. There is an IBM technical document (ID # PRS3780) that describes this process in more detail.

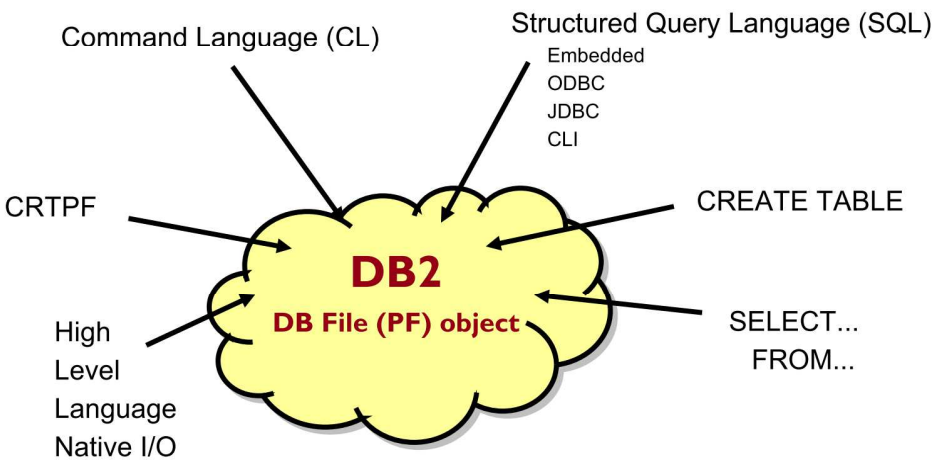
Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IBM i objects

IBM i

One database management system with multiple interfaces



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-16. IBM i objects OD477.0

Notes:

There is one database management system (DBMS) that handles native record level access or SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

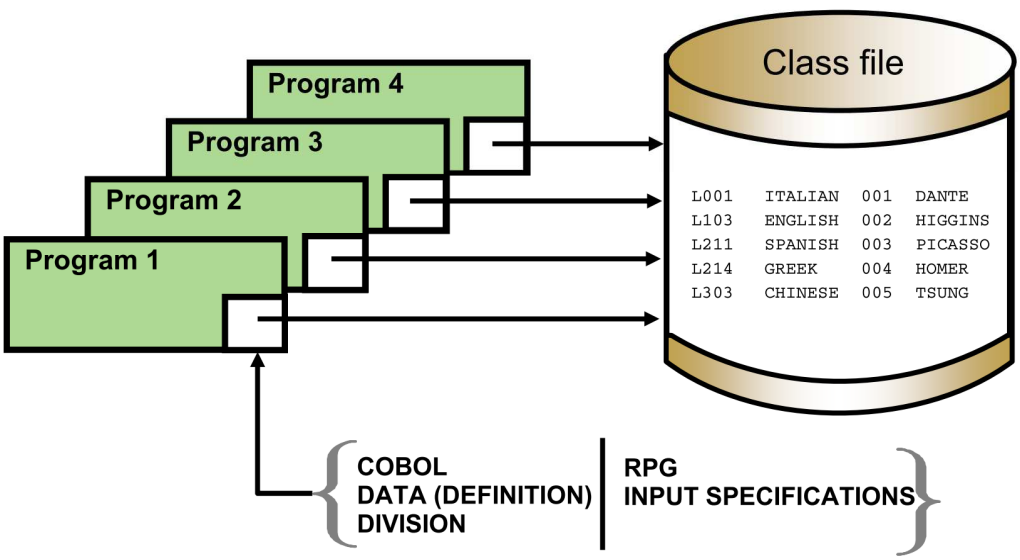
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Traditional program data definition

IBM i



Changing a file's format requires changing data definition in each program.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-17. Traditional program data definition OD477.0

Notes:

Many of you have seen programs that define data in each and every program. The same file may be defined over and over in different programs. These are called *program described files*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

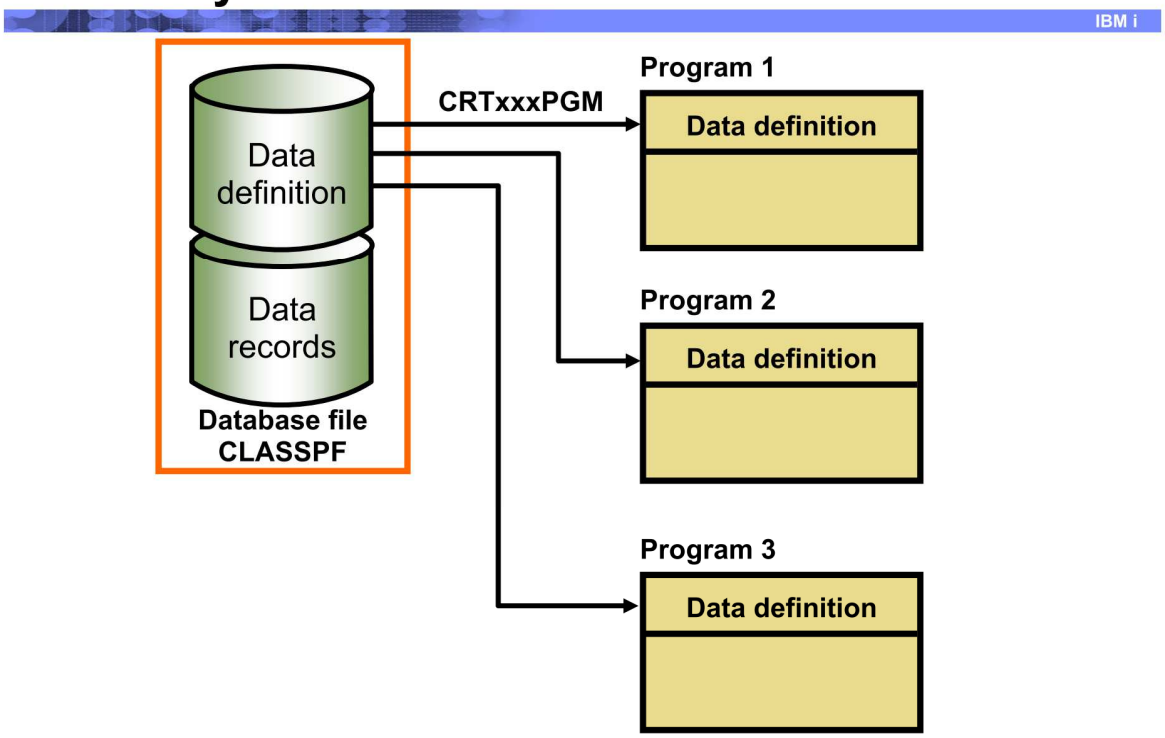
Additional information —

Transition statement — The IBM i overcomes this problem by providing the fully relational database and by enabling the definition of data externally. Files can then be defined as externally-described to the program.

The problem is that it is all too easy for a program to be written incorrectly and have definitions coded that do not match the data in the file. This is a classic problem with which many can identify.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Externally described IBM i file



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-18. Externally described IBM i file OD477.0

Notes:

When a file is externally described and the program is coded to refer to the external description, the file's external description is copied and becomes part of the program object when the program is created (compiled).

IBM i data files contain a record format at the front end of each file.

Programmers do not write definitions in their programs; they simply define the file as externally described. This causes the compiler to extract and copy the record definitions in from the file.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

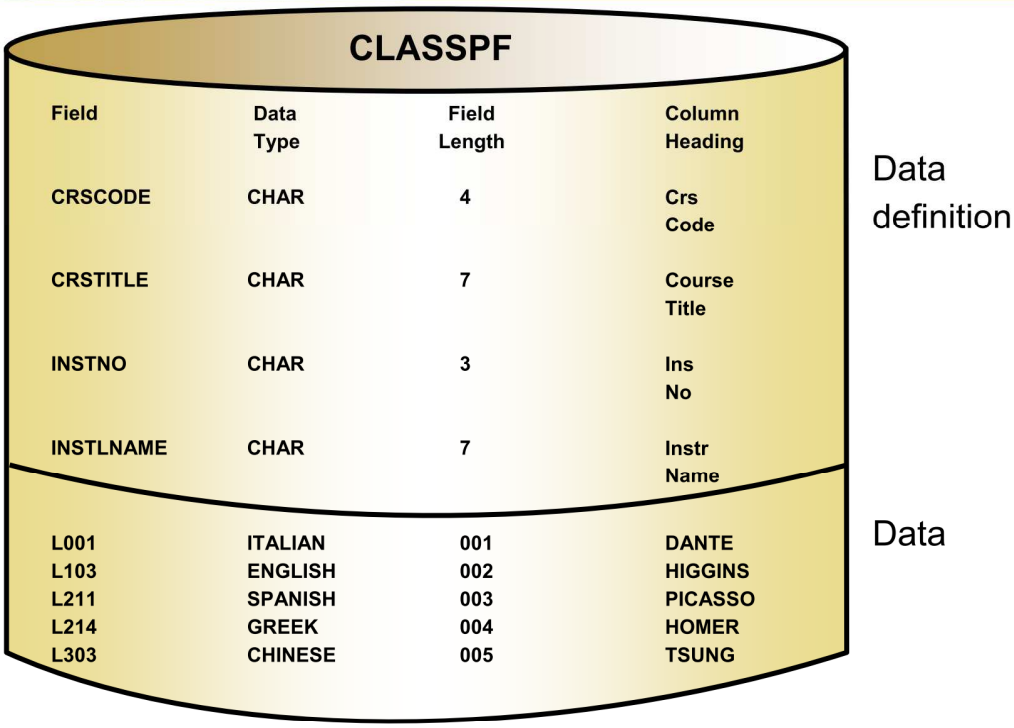
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Externally described file example

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-19. Externally described file example OD477.0

Notes:

At program creation (compilation) time, the data definition is copied into the program object, that is, the executable program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

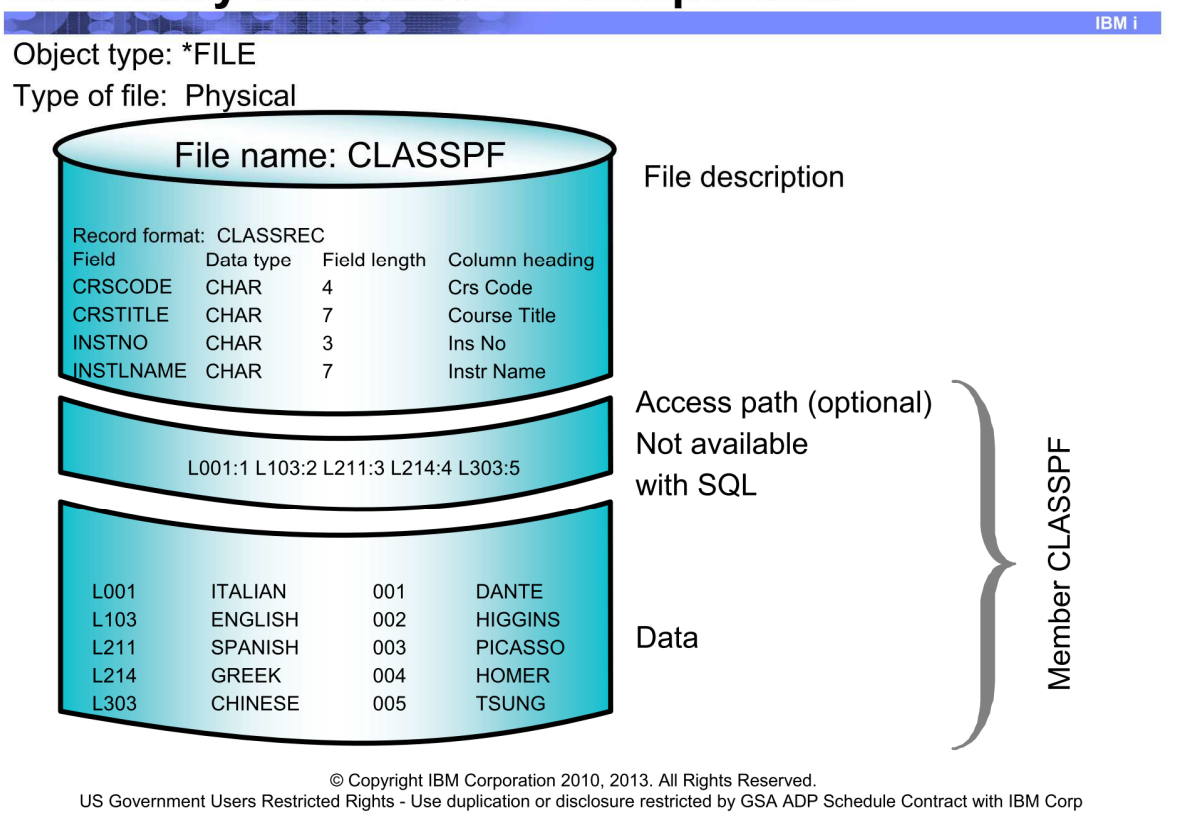
Details — We will be using the `CLASSPF` file as an example in this unit. Differentiate the definition part of the file versus the actual physical data.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Externally described file components



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-20. Externally described file components OD477.0

Notes:

In a relational database, an externally described file must have only one record format.

In the example, you see the note about no access path (index) to an externally described file. This is a rule of SQL. SQL creates tables in arrival sequence only. An index over the table must be created in order to access the data in a different sequence.

In native IBM i support, an access path can be included with a physical file. This is different than the SQL implementation using the CREATE TABLE statement.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

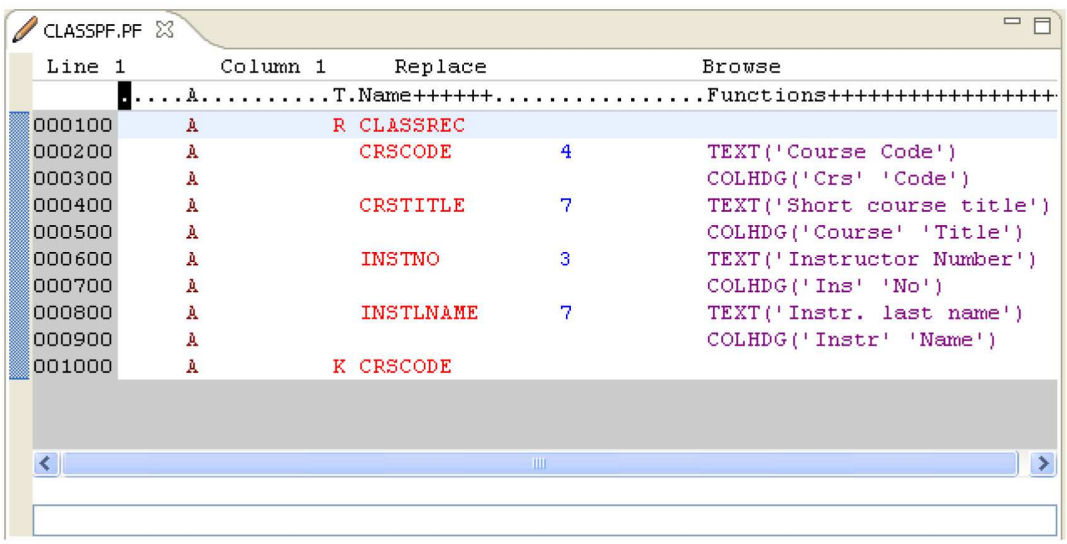
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Legacy coding to define database file CLASSPF

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-21. Legacy coding to define database file CLASSPF OD477.0

Notes:

Data description specifications (DDS) can be used to define database objects for native i access. Programming languages (such as RPG IV, COBOL and C) have their own proprietary access interfaces (known as record-level I/O) that access native i database objects as well.

SQL can also be used to define database objects for the i.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

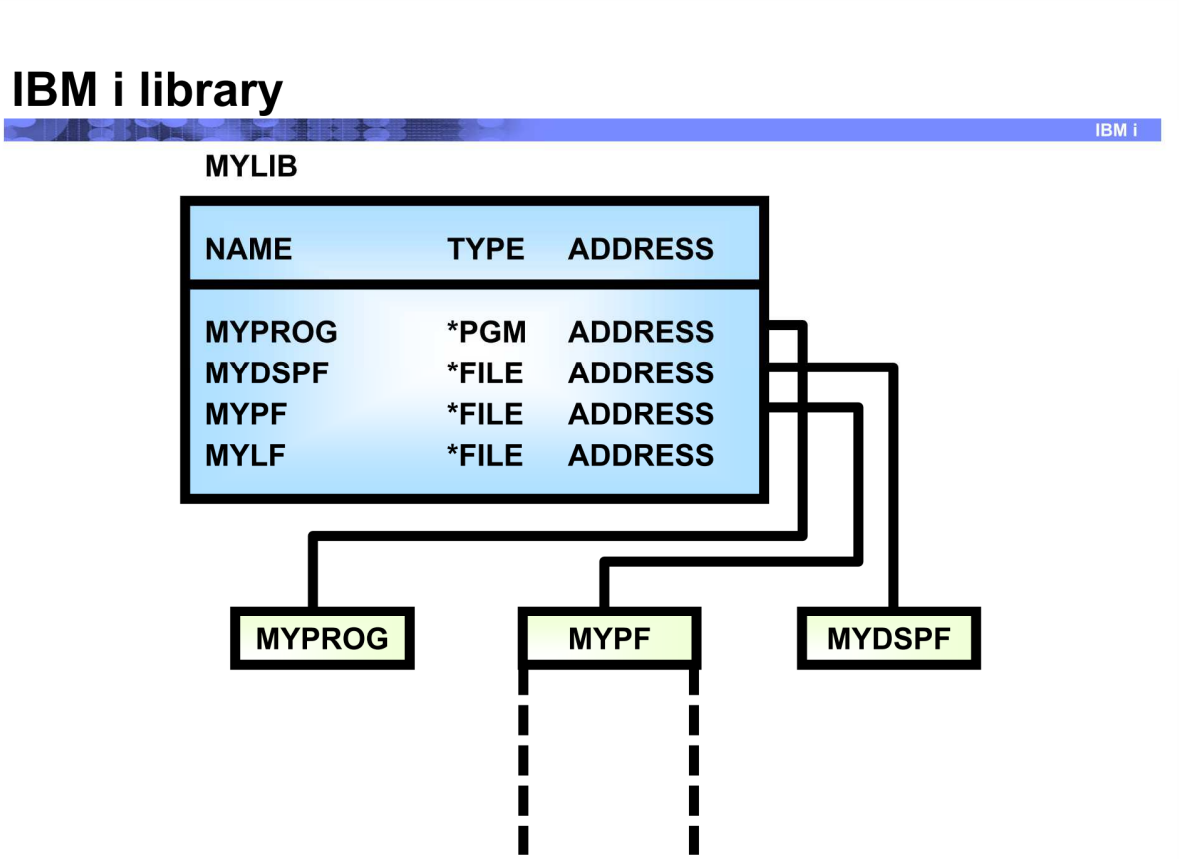
Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-22. IBM i library OD477.0

Notes:

For each object in a specific library, the library contains the object's name, type and a system managed transparent address.

An IBM I library is different from libraries of other systems. It has an open-ended directory that never fills up. It contains all object types (files, queues, job descriptions, and so forth), in addition to programs.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

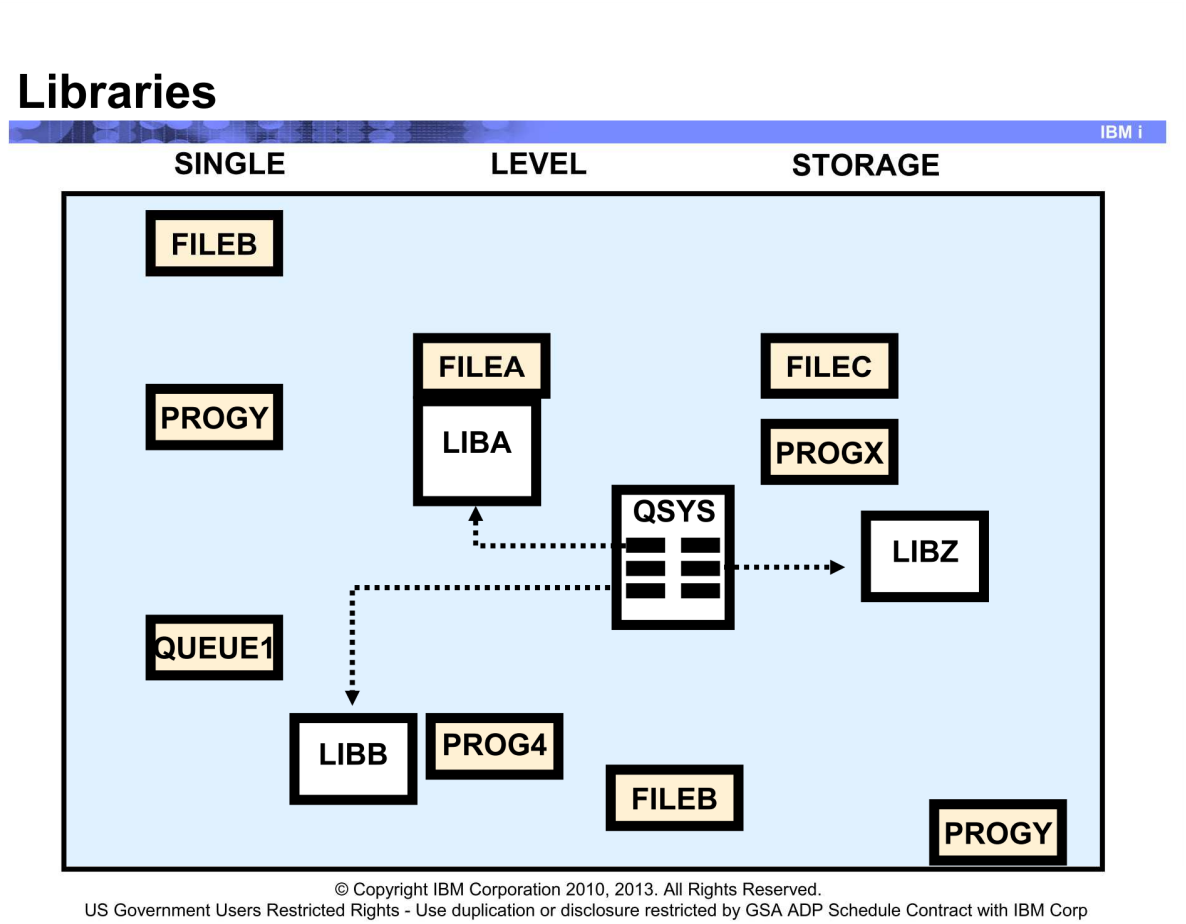
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-23. Libraries OD477.0

Notes:

Libraries themselves are objects in the IBM i. As such, they also reside in a system library named **QSYS**. Libraries can never reside anywhere other than in the QSYS file system.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

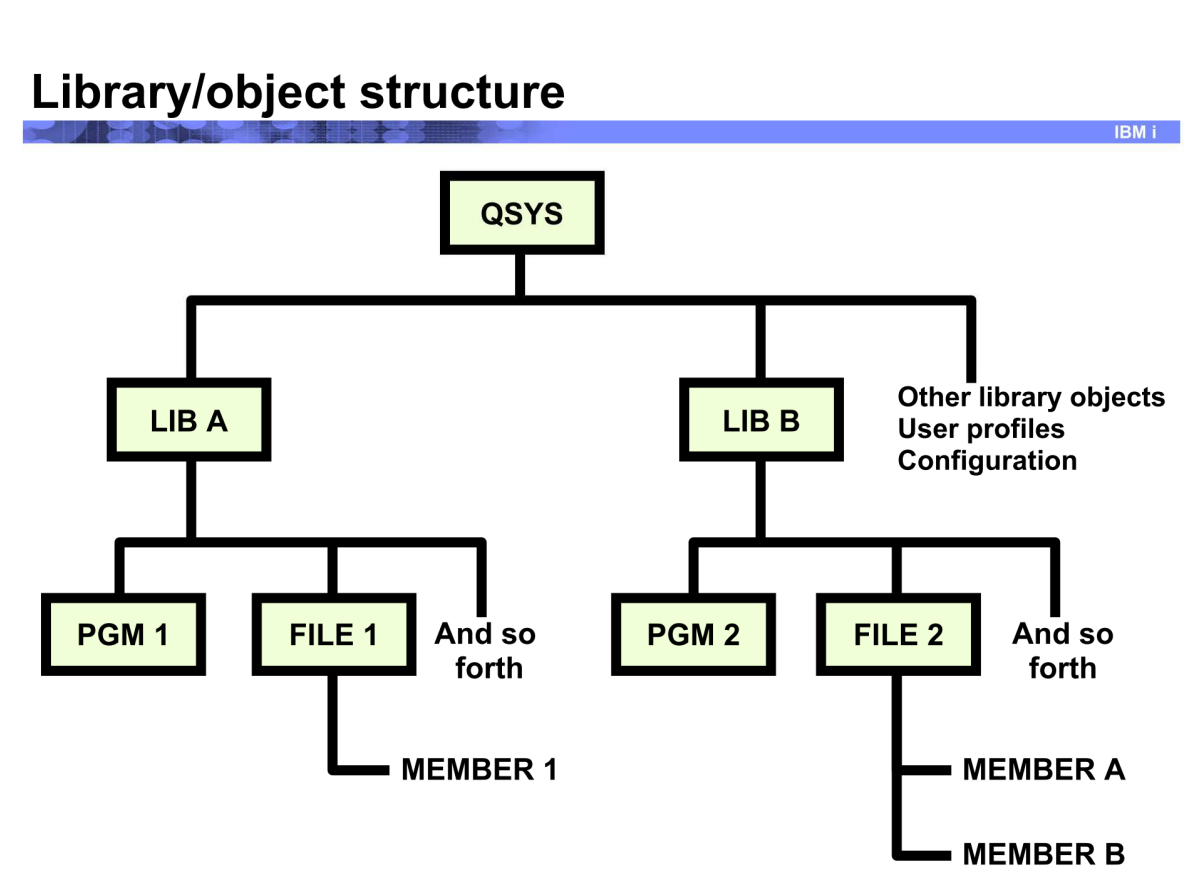
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-24. Library/object structure OD477.0

Notes:

Notice that File 2 has two distinct sets of data. These sets are known as members. Each member has a unique name and organizes your data such that Member A data can be processed independently from Member B data by using the file and member name.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

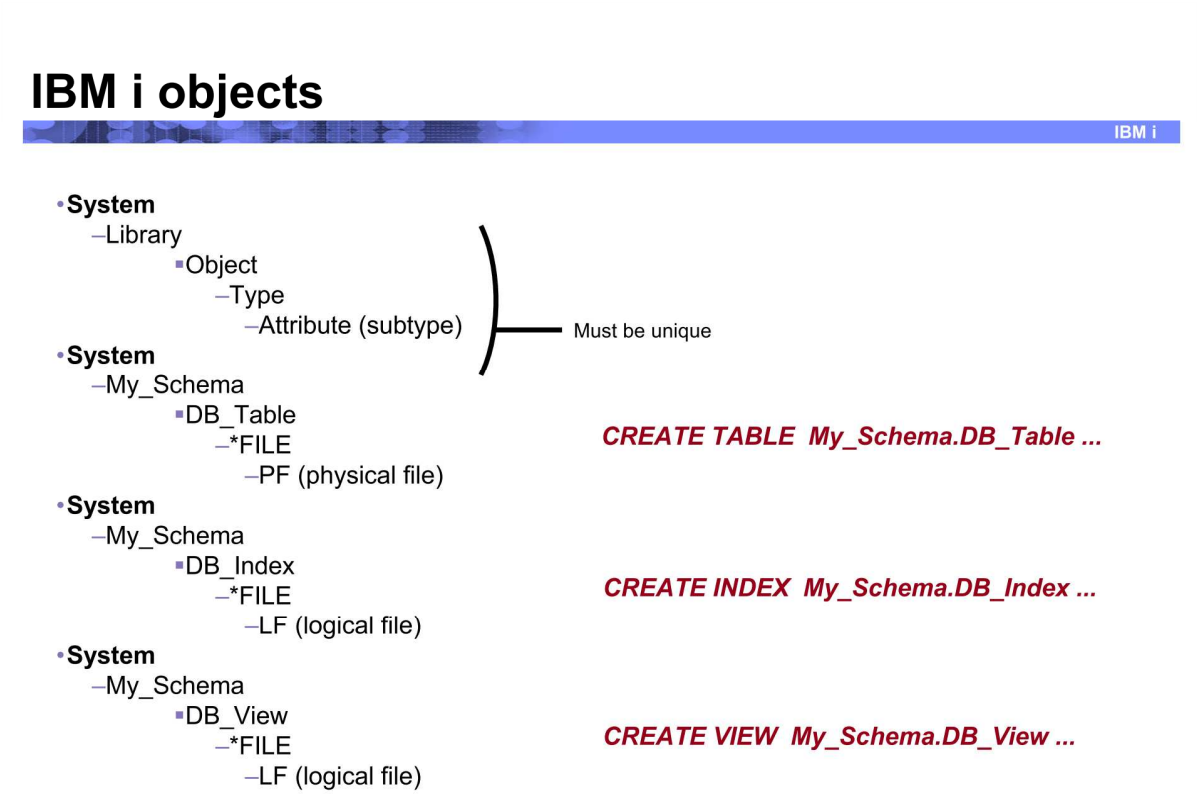
Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-25. IBM i objects OD477.0

Notes:

The implementation of SQL on the IBM i reveals the following:

- A table is a physical file.
- An index is a logical file.
- A view is an unkeyed, logical file.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Working with relational DBMS

IBM i

- Selection: Subsetting the records/rows of a file/table
- Sequence: Ordering of the records/rows
- Projection: Subsetting the fields/columns of a file/table
- Union: Two or more tables with identical arrangement of fields/columns merged vertically. Conceptually the resulting file/table is the same width, but has more records/rows.
- Join: Two or more files/tables with common fields/columns whose records/rows are combined horizontally, based on the common fields/columns, to make new rows, each of which are a combination of the fields/columns of the files. Conceptually (in the simplest case), the resulting file/table has the same number of records/rows, but is wider.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-26. Working with relational DBMS OD477.0

Notes:

When you perform selection, the resulting table consists of a subset of rows from the original table based upon certain criteria.

When you perform sequencing, the resulting rows are ordered.

When you perform a projection, conceptually the resulting rows have only the requested columns.

When you perform a union, two or more tables with the same number of columns and like column attributes are merged and treated as though they were one table.

When you perform a join, conceptually (in the simplest case), the resulting table has the same number of rows, but is wider and has more columns.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — These are general relational database operators, implemented in the IBM i system using data description specifications (DDS) coding or structured.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Sequence

IBM i

COURSE CODE	COURSE NAME	NUMBER ENROLLED	INSTRUCTOR	QTR	CLASS ROOM	CLASS MAX
L001	ITALIAN	12	DANTE	93/1	1	30
L103	ENGLISH	23	HIGGINS	93/1	4	40
L211	SPANISH	05	CERVANTES	93/1	4	40
L303	CHINESE	44	TSUNG	93/1	3	50
L001	ITALIAN	23	DANTE	93/1	1	30
L214	GREEK	11	HOMER	93/1	6	35
L211	SPANISH	01	HIGGINS	93/2	4	40
L103	ENGLISH	00	HOMER	93/2	4	40
L001	ITALIAN	2	CERVANTES	93/2	1	30
L306	JAPANESE	3	HOMER	93/2	6	35
L370	COBOL	99	HOMER	93/2	5	99

Sequence the file on INSTRUCTOR

COURSE CODE	COURSE NAME	NUMBER ENROLLED	INSTRUCTOR	QTR	CLASS ROOM	CLASS MAX
L211	SPANISH	05	CERVANTES	93/1	4	40
L001	ITALIAN	2	CERVANTES	93/2	1	30
L001	ITALIAN	12	DANTE	93/1	1	30
L001	ITALIAN	23	DANTE	93/1	1	30
L103	ENGLISH	23	HIGGINS	93/1	4	40
L211	SPANISH	01	HIGGINS	93/1	4	40
L214	GREEK	11	HOMER	93/1	6	35
L103	ENGLISH	00	HOMER	93/2	4	40
L306	JAPANESE	3	HOMER	93/2	6	36
L370	COBOL	99	HOMER	93/2	5	99
L303	CHINESE	44	TSUNG	93/1	3	80

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-27. Sequence OD477.0

Notes:

The sequence relational operator logically sorts the file.

On the IBM i, sequencing may be achieved through one or more key fields defined in the physical file or a separate view (logical file).

Most query tools allow the data to be sequenced dynamically, at the time the query is processed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

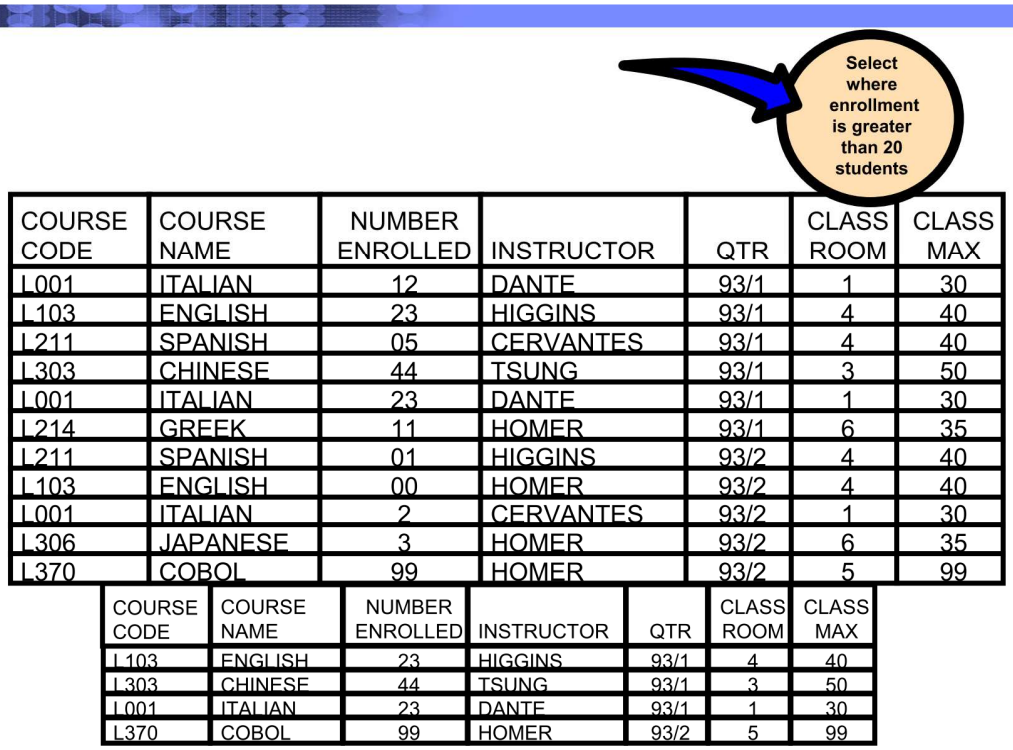
Additional information — A keyed logical file, being a permanent object, offers performance benefits for record sequencing. It may also be used by high level language (HLL) programming languages such as RPG IV and COBOL to simplify record processing.

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Select

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-28. Select OD477.0

Notes:

The select relational operator performs a logical record select.

Only the records (rows) selected are visible and can be updated when this view is used.

Record selection can also be achieved through a logical file, though this is not normally recommended. Most query tools provide more powerful and flexible record selection capabilities.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

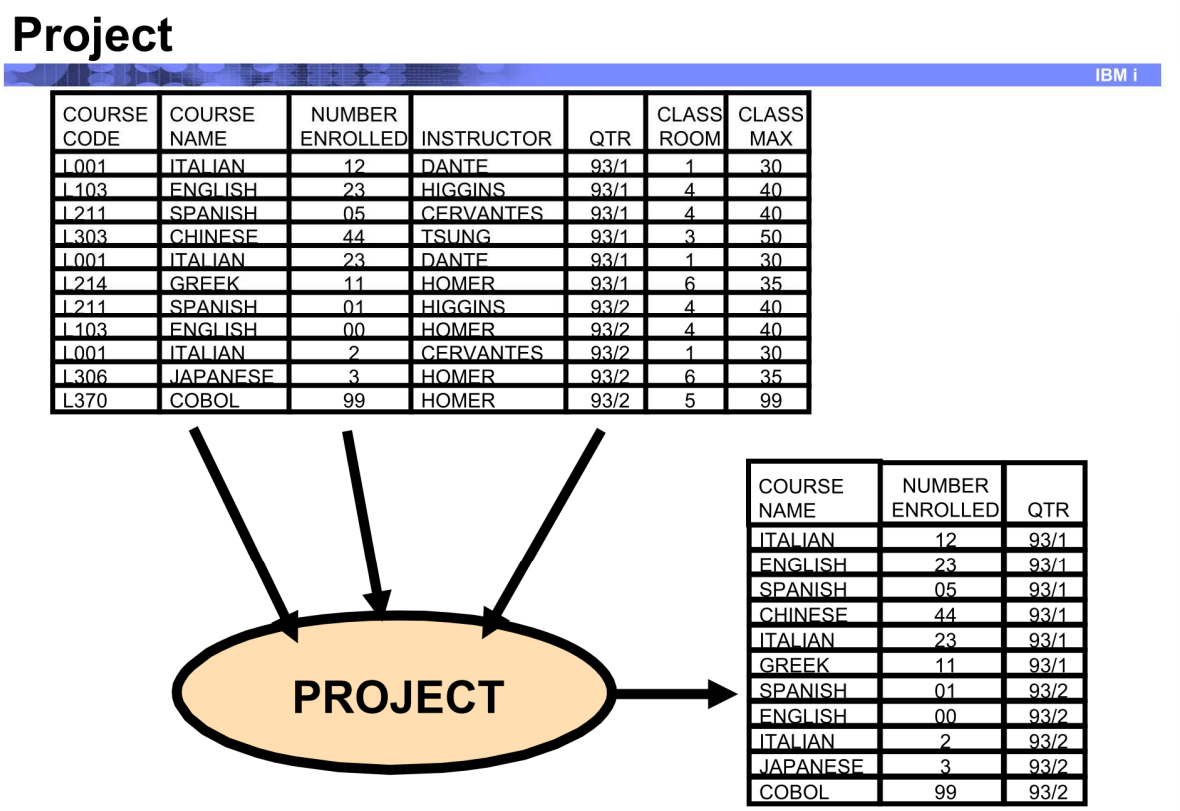
Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-29. Project OD477.0

Notes:

The project relational operator performs a field select.

Only the fields (columns) projected are visible and can be updated when this view is used.

This operator is often used for security and end-user queries.

Logical files can be used for column projection.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

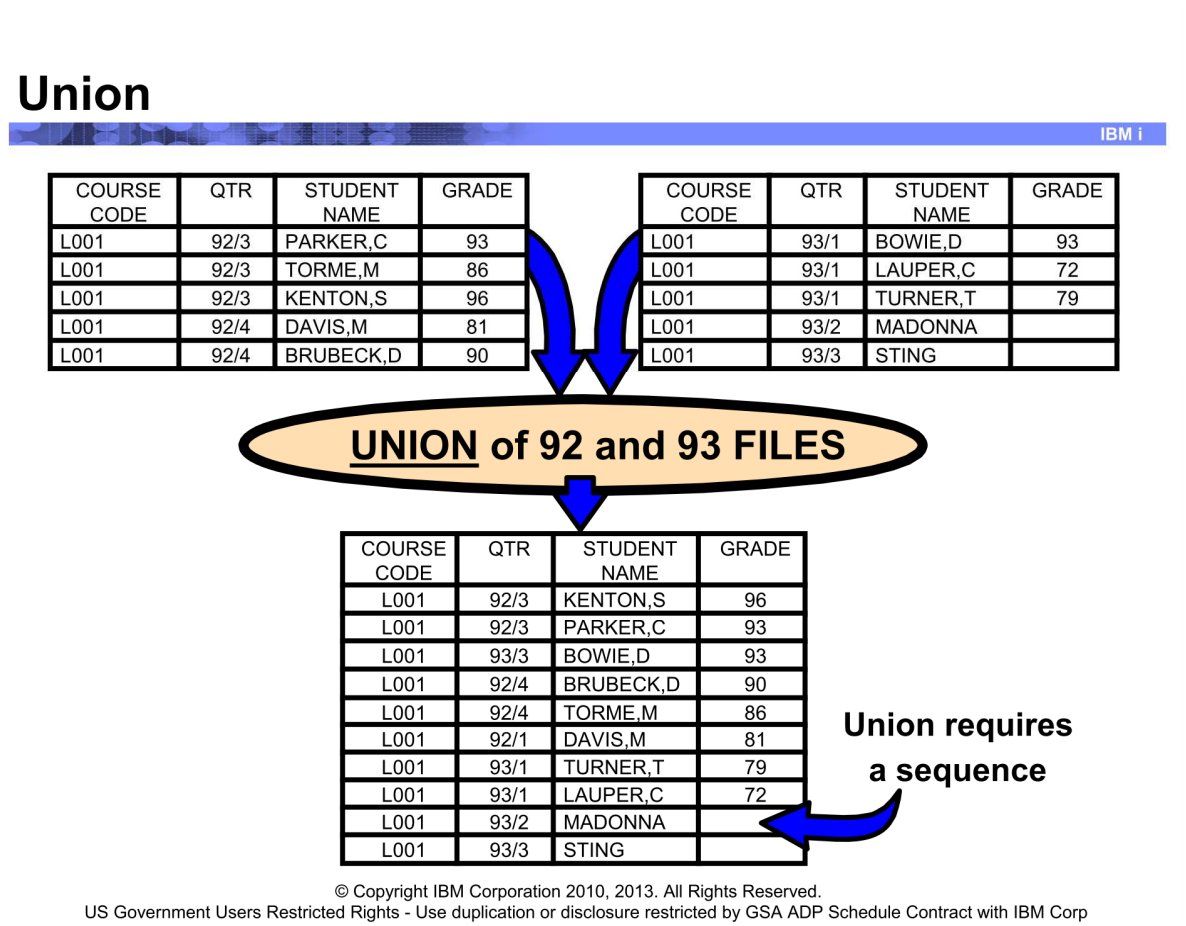


Figure 1-30. Union OD477.0

Notes:

Union stacks multiple (same record format) small tables together to form a single large table. This operator is used extensively in mainframe applications to keep individual table size down and performance up.

The union above was sequenced by the GRADE column. A union of two files on the IBM i requires that the file have a key.

Unions and joins are another function of logical files.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

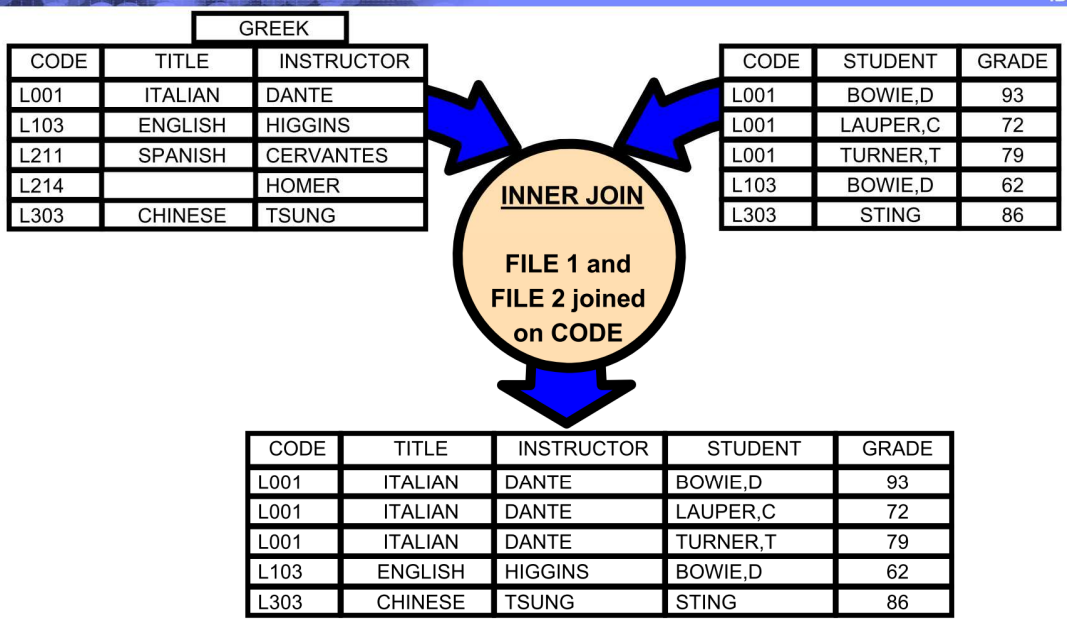
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Inner join

IBM i



Only rows that match are joined.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-31. Inner join OD477.0

Notes:

The inner join is also referred to as a natural join or a matching join.

Joins allow combination of data stored as separate files (due to data moralization) into a convenient and usable form.

Data presented through a joined logical file can only be used for query purposes. In other words, the underlying records cannot be changed or deleted, neither can any new records be added.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

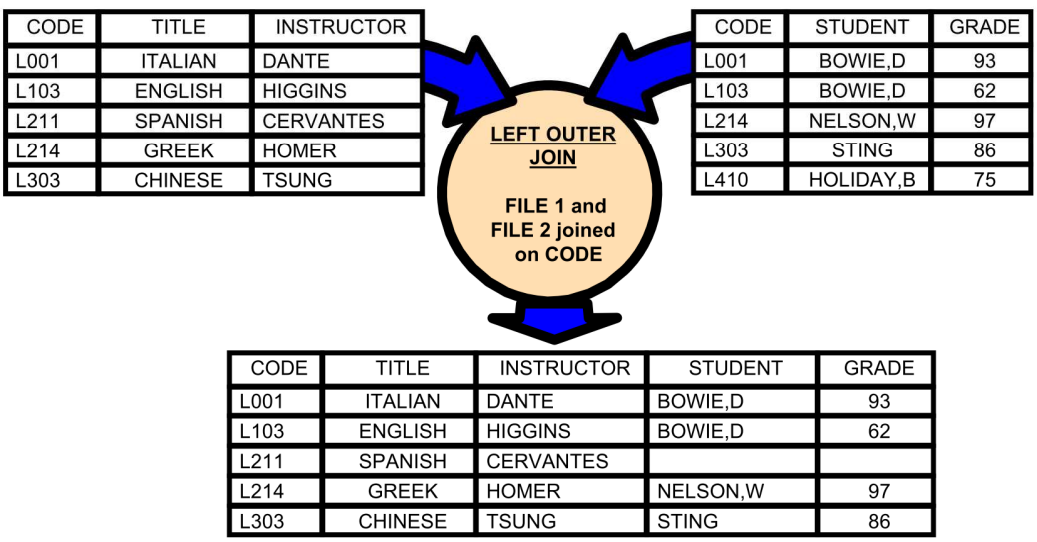
Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Left outer join

IBM i



Contains all rows from the left table

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-32. Left outer join OD477.0

Notes:

The left outer join is sometimes referred to as a value fill or a default value join.

In this type of join there is a left (primary) and a right (secondary) file. Some primary records may have no matching secondary records, in this case the secondary fields are assigned their default value.

With a join default value specified, the system returns a record, even though the record is missing in the secondary file. With a join default value specified, missing character fields normally use blanks; missing numeric fields use zeros.

In SQL the missing secondary fields would be null and not the default value (blanks or zero).

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — With a join default value specified, the system returns a record, even though the record is missing in the secondary file. With a join default value specified, missing character fields normally use blanks; missing numeric fields use zeros.

In SQL the missing secondary fields would be null and not the default value (blanks or zero).

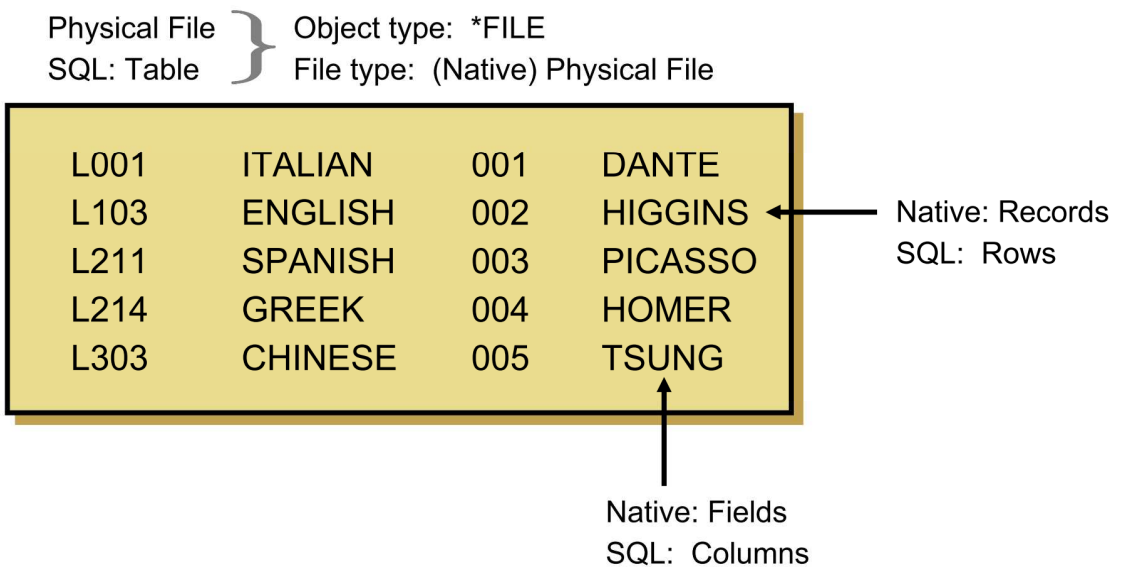
Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL versus native IBM i object terminology

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-33. SQL versus native IBM i object terminology OD477.0

Notes:

An SQL schema (also known as a collection) has a journal, a journal receiver, a catalog and optionally a data dictionary. They are not shown in the figure.

- Collections

  The objects in a relational database are organized into sets called collections or schemas. A collection or schema provides a logical classification of objects in the database. Collections are similar to an IBM I library.

- Table

  A table is a two-dimension object consisting of rows and columns. It is used to store user data. Tables are physical files that are maintained by the database manager.

- Rows and columns

  Tables are made up of columns and rows. There is no inherent order of the rows within a table. At the intersection of every column and row is a specific data item called a value. A column is a set of values with the same data type. A row is a sequence of associated values.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

More terminology

IBM i

SQL term	IBM i term
Table	Physical file
Row	Record
Column	Field
View	Logical file
Index	Keyed logical file
Collection/schema	Library
Log	Journal
Isolation level	Commitment control level

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-34. More terminology OD477.0

Notes:

Terms on the left equate to the terms on the right. Traditional i users are accustomed to using the terms on the right. While the terms used might be different, the function provided is the same.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Journaling and locking are optional on the i. If students ask, journaling adds about 5 - 8% overhead on the system.

Mention that there are no table spaces. DB2 for i automatically carves out the necessary storage space for objects. No database administrative action is required.

Additional information —

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OD477U01** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OD477U01** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OD477U01** and ask the students to call out the answers to the questions on the screen

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: All data items in a relational database are stored in base tables.

2. Which of the following are relational operators?
   a. Selection
   b. Projection
   c. Joins
   d. All of the above

3. True of False: The IBM i implementation of a relational database has a performance advantage over the traditional implementation in that it is an integrated part of the operating system.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-35. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: All data items in a relational database are stored in base tables.
   The answer is True.

2. Which of the following are relational operators?
   a. Selection
   b. Projection
   c. Joins
   d. All of the above
   The answer is All of the above.

3. True or False: The IBM i implementation of a relational database has a performance advantage over the traditional implementation in that it is an integrated part of the operating system.
   The answer is True.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe a relational database
- Describe features of DB2 for I
- Define basic IBM i database objects
- Compare native IBM i with SQL terminology

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 1-36. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 2. Introducing Structured Query Language

Estimated time

01:15

What this unit is about

This unit introduces Structured Query Language (SQL). We discuss the terms used by SQL to describe the DB2 for IBM i database. This unit gives the students the background necessary to understand how to use SQL to access the database and relates the SQL terms and methods to the well known terms and commands they have been using.

Only basic concepts are covered. Further detail and exercises follow in subsequent units.

The history of SQL and its importance in the industry are also discussed.

What you should be able to do

After completing this unit, you should be able to:

- Explain the meaning of the following SQL terms:
  - Schema and collection
  - Table
  - View
  - Row
  - Column
  - Key
  - Index
- Describe the origins of SQL
- Name two significant applications where SQL can be used

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercise

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Explain the meaning of the following SQL terms:
  - Schema and collection
  - Table
  - View
  - Row
  - Column
  - Key
  - Index
- Describe the origins of SQL
- Name two significant applications where SQL can be used

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

2.1. Introducing SQL

Instructor topic introduction

What students will do — We examine the history of SQL and its importance in the industry. Then we define commonly used SQL terms.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

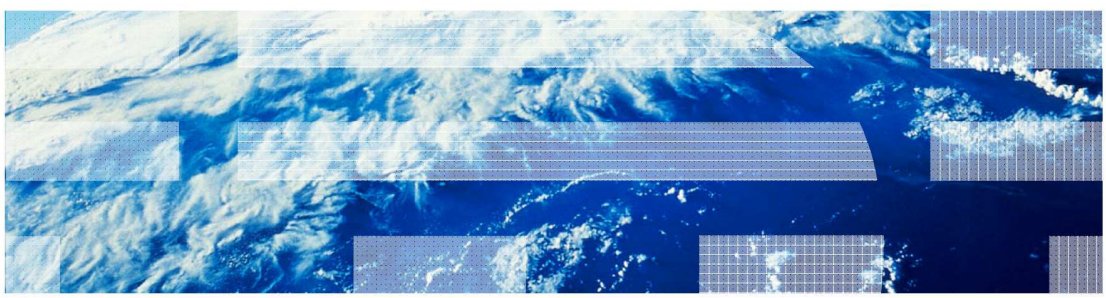


Figure 2-2. Topic 1: Introducing structured query language OD477.0

Notes:

SQL has been the industry standard query tool for a number of years. SQL is implemented on the IBM i and is officially named DB2 for IBM i SQL.

SQL is a standardized language for defining and manipulating data in a relational database. In accordance with the relational model of data, the database is perceived as a set of tables, relationships are represented by values in tables, and data is retrieved by specifying a result table that can be derived from one or more base tables.

The ability to create SQL objects is built into the DB2 for IBM i Database Manager, which is part of the base operating system. However, the ability to query these objects requires that 5770-ST1 DB2 Query Manager and SQL Development Kit be installed on your system or you must use some other SQL-enabled product. For example, you could use System i Navigator which uses Java Database Connectivity (JDBC) to access SQL objects. **Note**: If using V4 iSeries Navigator, Object Database Connectivity (ODBC) is used to access SQL objects.

SQL statements are executed by the database manager. One of the tasks performed by the database manager is to transform the specification of a result table into a sequence of

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

internal operations that optimize data retrieval. This transformation occurs when the SQL statement is prepared. This transformation is also known as *binding*.

All executable SQL statements must be *prepared* before they can be executed. The result of preparation is the executable or operational form of the statement.

SQL can be used to create and manage databases and to extract information from and about databases. SQL can be run both interactively or from within an application program. Queries might also be designed and managed by the Query Manager for IBM i.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

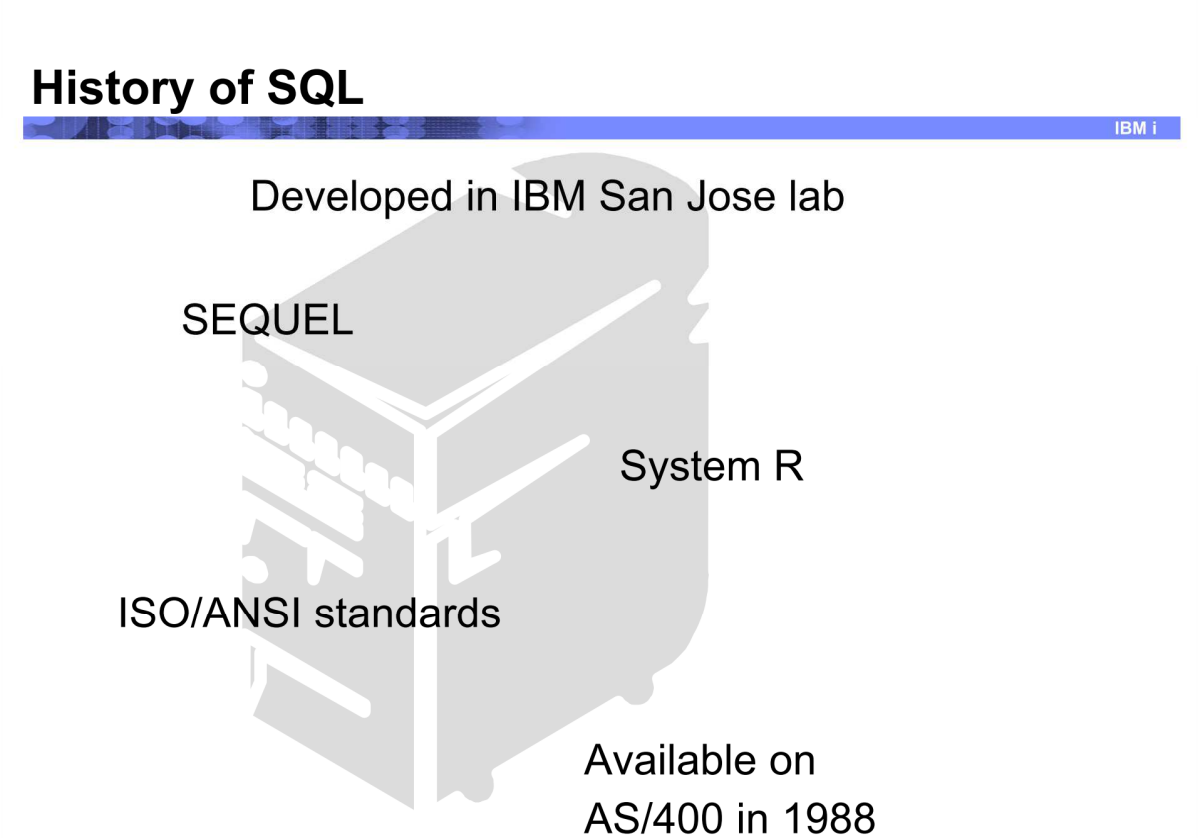
Details —

Additional information — The method of preparing an SQL statement and the persistence of its operational form distinguish static SQL from dynamic SQL.

Transition statement — Before we discuss SQL and the many features of the language, let's discuss the history of SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-3. History of SQL OD477.0

Notes:

SQL was originally named Structured English Query Language (SEQUEL), but when it was discovered that the name SEQUEL conflicted with an existing trademark, the name was changed to Structured Query Language (SQL) in the late 1970s.

SQL was developed in conjunction with the System R project in the IBM San Jose laboratory. The System R development team developed the first relational database prototype. One of the lead IBM developers was Mr. Don Chamberlin who also led in the development of the SQL language.

Although SQL was developed and prototyped by IBM, its first commercial implementation was released by Relational Software in 1979. Their product, Oracle, was later adopted as the name of the company.

Over the years, SQL has become the standard query tool of the IT industry and has been implemented on almost every platform available today.

To enable portability, standards have been defined for SQL implementations by both the ANSI organization (number X3.135-1986) and the ISO organization (number 9075-1987).

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL was first available on the AS/400 with V1R1.1 of OS/400 in late 1988.

References:

- *A Complete Guide to DB2 Universal Database* (Don Chamberlin), A Brief History of SQL

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — We want to focus the students' attention on the importance of SQL as a database access tool. SQL was invented by IBM and has been adopted across the industry as the standard tool used to access databases.

Details —

Additional information —

Transition statement — Next, we will stress some of the many implementations of SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Some forms of SQL

IBM i

- Interactive SQL: Prototyping tool
- Embedded SQL: SQL statements in IBM i application programs
- ODBC: SQL call interface from the PC to the IBM i
- JDBC: Java call interface from the PC to the IBM i
- Stored procedures: Special IBM i programs that serve PCs
- Run SQL scripts tool of System I Navigator

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-4. Some forms of SQL OD477.0

Notes:

On the IBM i, SQL is used in many different ways. A few are:

1. **Interactive SQL**

   The Interactive SQL interface is used to prototype queries. SQL statements are processed dynamically and they are broken down into component parts known as tokens. If the tokens are syntactically correct, the statement is executed.

   You can use Interactive SQL (`STRSQL` command) and Run SQL Scripts in System i Navigator to develop queries in our machine exercises.

2. **Run SQL Scripts**

   The Run SQL Scripts interface is a PC-based GUI interface, included with System i Navigator, part of IBM i Access for Windows, that can also be used to prototype queries on the IBM i server. SQL statements are processed dynamically and they are broken down into component parts known as tokens. If the tokens are syntactically correct, the statement is executed.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

A component of this facility, SQL Assist, enables you to *build* SQL and insert it into a script.

3. **Embedded SQL**

   Many application programs include SQL statements that are embedded in Report Program Generator (RPG) IV, COBOL, and other high level language (HLL) programs to perform I/O that accesses and manipulates the database.

4. **ODBC**

   ODBC is defined by Microsoft. We implement it as the call level interface and provide a set of functions that can be embedded in application programs to access a database. Command line interface (CLI) is often implemented in client-server applications.

5. **JDBC**

   Java-based client-server applications use the JDBC interface to access the database. JDBC is based on SQL functions and calls.

6. **Stored procedure**

   A stored procedure is an application program that runs on a server that is bound to a database. A separate client application can call a stored procedure to access the database on the server. This is an efficient way of using the power of SQL to access the database on the IBM i server.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

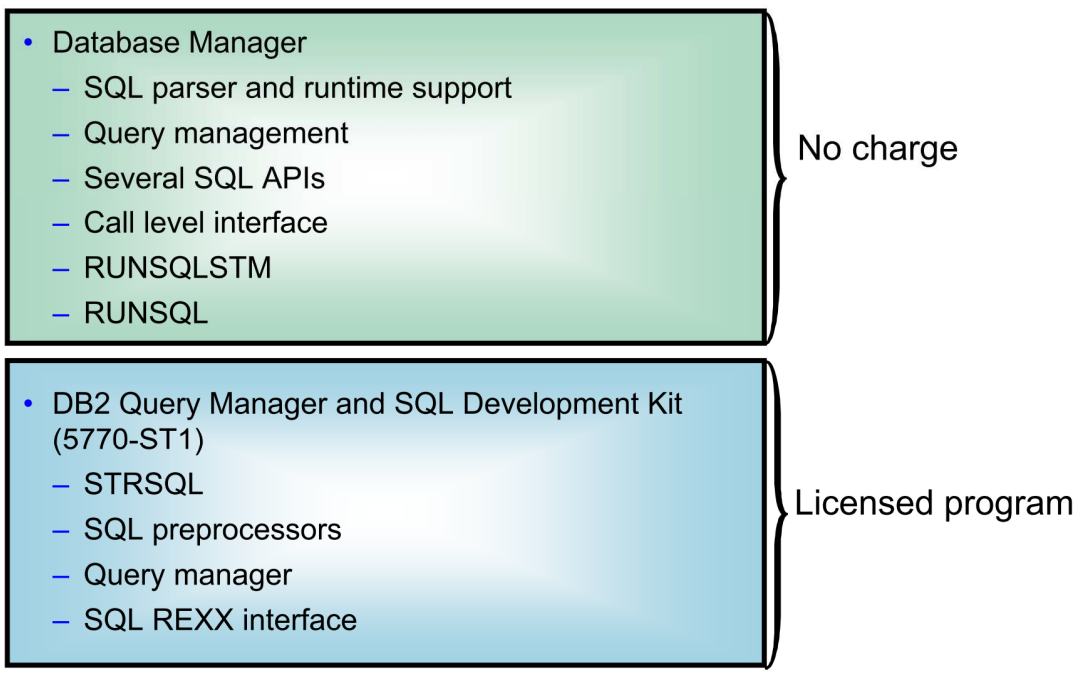
Details — Point out that in this class, students will be learning the SQL language and will focus on using Interactive SQL. We will use both STRSQL and System i Navigator as tools to access the course database. The other areas will not be covered in this class, but for those who want to learn more about them, they are taught in the follow on course OL38/OL380, *Developing IBM i Applications Using SQL*.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL packaging

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-5. SQL packaging OD477.0

Notes:

The visual shows you some of the DB2 for IBM i features that are included at no additional charge in the Database Manager contrasted with what is included in the chargeable product 5770-ST1.

Note: Certain system packages might include a server license for 5770-ST1 bundled into the price. Check with your IT manager to see whether a copy of the product is available at your site.

Included at no charge with the Database Manager are:

- **Runtime support**

  If you have written and compiled an IBM i program that includes embedded SQL statements, you can run the executable program on any IBM i.

  SQL runtime is available on every IBM i and the `WRKQMQRY` command allows you to execute SQL on any IBM i regardless of whether you have the licensed program installed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- **Query management**

  Query management is an IBM i programmer tool that is used by application programs that access Query/400 objects.

- **QSQPRCED and QSQCHKS APIs**

  The QSQPRCED application programming interface (API) provides extended dynamic SQL capability. SQL statements can be prepared into an SQL package and then executed using this API. Statements prepared into a package by this API persist until the package or statement is explicitly dropped. QSQPRCED is part of the IBM i/OS licensed program.

  The QSQCHKS API syntax checks SQL statements.

- **Call level interface**

  The SQL call level interface allows users of any of the Integrated Language Environment (ILE) languages to access SQL functions directly through procedure calls to a service program provided by the system. Using the SQL call level interface, one can perform all the SQL functions without the need for a precompile. This is a standard set of procedure calls to prepare SQL statements, execute SQL statements, fetch rows of data, and even do advanced functions such as accessing the catalogs and binding program variables to output columns.

- **`RUNSQLSTM` CL command**

  This tool allows a group of one or more SQL statements to be run as a set. We use this facility during the class. Typically, you prototype individual statements using `STRSQL`, save the statements as a group in a source member, and then process the source member using `RUNSQLSTM`. Statements such as CREATE, DELETE, DROP, and INSERT can be run using this facility, but SELECT, which produces a result set, cannot be run in this way.

  Since V5R1, this tool is a part of OS/400 or IBM i/OS.

- **`RUNSQL` CL command**

  Allows an SQL statement to be run from the IBM i command line or within a CL program without needing a source file. Statements such as CREATE, DELETE, DROP, INSERT, and UPDATE can be run using this facility.

Included in the IBM DB2 query manager and SQL Development Kit (5770-ST1) licensed program product are:

- **Interactive workbench (`STRSQL`)**

  This tool is used to prototype SQL statements. We use it in this class, as well as the GUI PC tool, Run SQL Scripts in System i Navigator.

- **Programming language precompilers**

  Application programmers use these to develop programs that include embedded SQL statements. Often, using SQL is more effective or provides better function than coding

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

normal I/O statements. The SQL Development Kit provides precompilers for processing SQL statements embedded in the following programming languages:

- ILE C
- ILE C++
- VisualAge C++
- ILE COBOL
- COBOL
- PL/I
- RPG III (also known as RPG/400)
- ILE RPG for i (RPG IV)

- **DB2 Query Manager for IBM i/OS**

  DB2 Query Manager for IBM i/OS is an end user tool that is SQL based. It provides comprehensive facilities that enable you to use prompting to create SQL queries and to create data, add data, maintain data, and run reports on the database tables and views. Data edit capabilities are included in Query Manager.

  We introduce you to this tool in a later unit of this class.

- **SQL REXX interface**

  The SQL REXX interface allows you to run SQL statements in a REXX procedure.

5770-ST1

This product is required only for Query Manager and SQL application development. Once created, these applications can be run on other IBM i systems without this product installed, using the DB2 Database Manager support provided with IBM i/OS.

Note: You can use the System i Navigator Run SQL Scripts facility, including the SQL Assist feature, without having 5770-ST1 installed on the IBM i server.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Emphasize that the SQL runtime environment is included with every IBM i. Emphasize that we also cover the SQL language in this class and that we use the interactive SQL interface as well as System i Navigator extensively. We also use the `RUNSQLSTM` CL command.

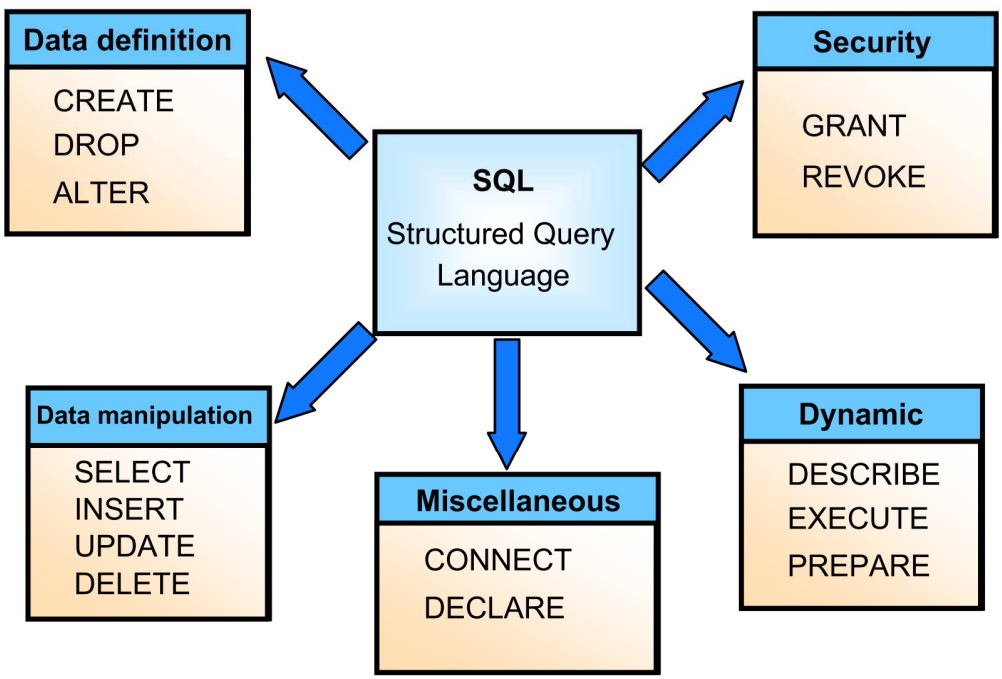
For users familiar with the Eclipse development environment, you may want to mention the QuantumDB Eclipse Plugin. Quantum DB is a simple but powerful database access plug-in for the Eclipse Development Platform. It is similar to System I Navigator's database interface in that allows you to connect to databases and review schemas, tables, views and indexes. You can issue updates, deletes and inserts using simple, easy-to-use wizards. You can also issue ad hoc queries and other SQL statements against the database. To learn more for yourself and to refer interested students, go to the following site: http://quantum.sourceforge.net/

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL components

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-6. SQL components OD477.0

Notes:

These are the major components of the Structured Query Language. The visual shows just a few of the SQL statements available in the language:

- **Data Definition Language (DDL)** is used to create, drop, and alter SQL objects and their description and definition.

  For example, you can define a column or field and then add a constraint to restrict the range of data allowed. The i supports the following DDL statements:

  - ALTER TABLE
  - COMMENT ON
  - CREATE SCHEMA or CREATE COLLECTION
  - CREATE INDEX
  - CREATE PROCEDURE
  - CREATE SCHEMA

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

  - CREATE TABLE
  - CREATE VIEW
  - DROP SCHEMA or DROP COLLECTION
  - DROP INDEX
  - DROP PACKAGE
  - DROP PROCEDURE
  - DROP SCHEMA
  - DROP TABLE
  - DROP VIEW
  - LABEL ON
  - RENAME

  The GRANT and REVOKE statements are also considered DDL statements, but are grouped under the **Security** category in the visual.

- **Data Manipulation Language (DML)** is used to select and present data, and change the value of columns or fields.

  The IBM i supports the following DML statements:

  - CLOSE
  - COMMIT
  - DECLARE CURSOR
  - DELETE
  - FETCH
  - INSERT
  - LOCK TABLE
  - OPEN
  - ROLLBACK
  - SELECT INTO
  - UPDATE

- **Security (or CONTROL)** statements are actually DDL statements, but are being addressed separately because they are used to grant or revoke authority to SQL objects.

  The IBM i supports the following statements:

  - GRANT PACKAGE
  - GRANT PROCEDURE

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

  - GRANT TABLE
  - REVOKE PACKAGE
  - REVOKE PROCEDURE
  - REVOKE TABLE
- **DYNAMIC - SQL** can be dynamically prepared in order to improve execution.

  The IBM i supports the following DYNAMIC statements:
  - DESCRIBE
  - EXECUTE
  - EXECUTE IMMEDIATE
  - PREPARE
- **MISCELLANEOUS** statements allow access to databases.

  The IBM i supports the following as MISCELLANEOUS statements:
  - BEGIN DECLARE SECTION
  - CALL
  - CONNECT
  - DECLARE PROCEDURE
  - DECLARE STATEMENT
  - DECLARE VARIABLE
  - DESCRIBE TABLE
  - DISCONNECT
  - END DECLARE SECTION
  - INCLUDE
  - RELEASE
  - SET CONNECTION
  - SET OPTION
  - SET RESULT SETS
  - SET TRANSACTION
  - WHENEVER

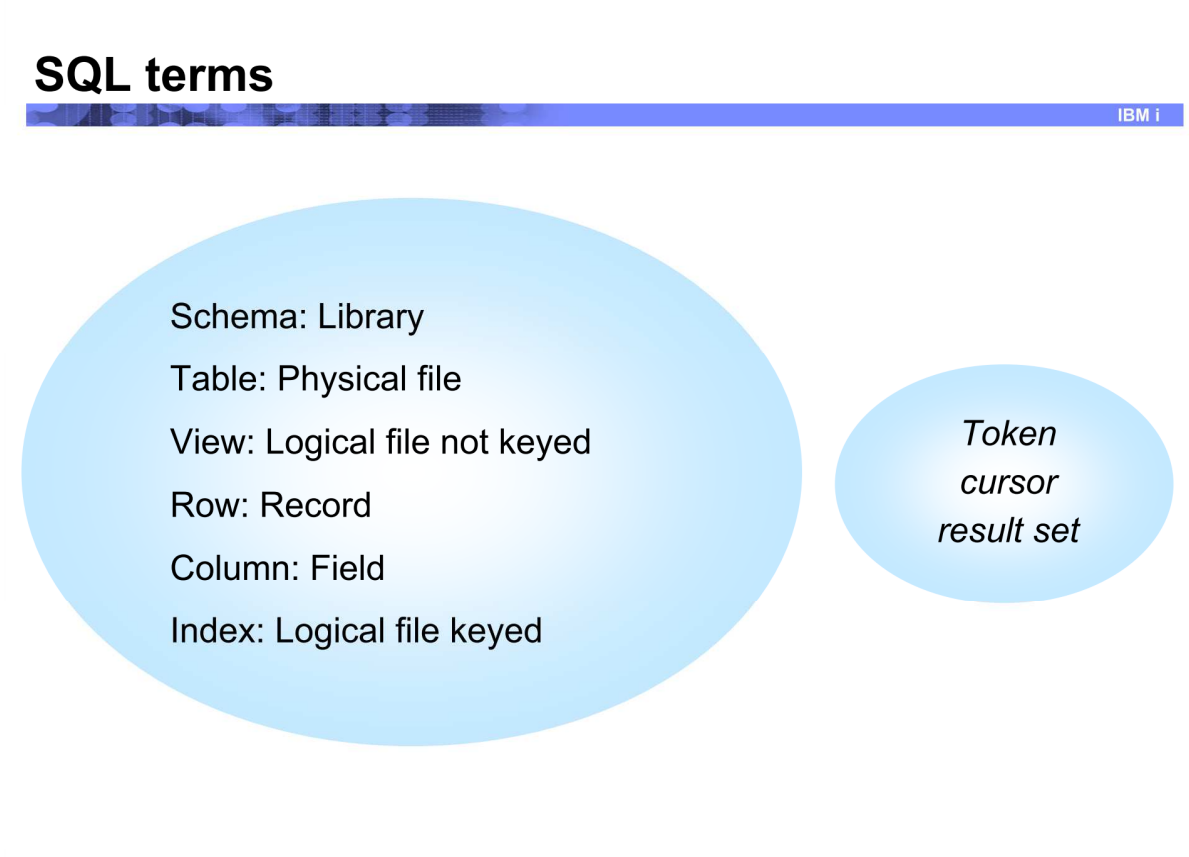
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — It is not necessary to cover all of these SQL statements. Provide an overview only. We cover most of the statements in this class. We do not cover the statements in the Miscellaneous or Dynamic categories.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-7. SQL terms OD477.0

Notes:

- **Schema (collection)**

  The objects in a relational database are organized into sets called *schemas* (or collections). A schema (or collection) provides a logical classification of objects in the database.

  A schema is often known as a *collection*. Schemas and collections are synonymous. Since V5R2 of the *SQL Reference*, the term *schema* is used more frequently than collection as schema is the industry standard term.

  Schemas (or collections) are similar to i libraries.

- **Table**

  A table is a two-dimensional object consisting of rows and columns. It is used to store user data. Tables are physical files that are maintained by the database manager.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- **Rows and columns**

  Tables are made up of columns and rows. There is no inherent order of the rows within a table. At the intersection of every column and row is a specific data item called a *value*. A column is a set of values of the same type. A row is a sequence of values, such that the nth value is a value of the nth column of the table.

  You have already been using rows and columns for many years. The fields you define in Data Description Specifications (DDS) and RPG are columns and the records in your data files are rows.

- **View**

  A view provides an alternate way of presenting the data from one or more tables. It is a non-keyed logical file.

- **Index**

  An SQL index is a subset of the data in the columns of a table that are logically arranged in either ascending or descending order. Indexes are used by the system for faster data retrieval. An SQL index is a keyed logical file.

You will also encounter the following terms as you use SQL:

- **Token**

  The basic syntactical units of the SQL language are called tokens. A token consists of one or more characters, excluding blanks, control characters, and characters within a string constant. Tokens are derived as a result of parsing, or breaking up, an SQL statement in preparation to execute it.

  In this sample SQL statement:

```
SELECT EMPNO, EMPNAME FROM EMPLOYEE
```

  SELECT, EMPNO, EMPNAME, FROM, and EMPLOYEE are tokens. When the statement is prepared, it is parsed into these tokens.

  Tokens are classified as either *ordinary* or *delimiter*:

  - An *ordinary* token is a numeric constant, an ordinary identifier, a host identifier, or a keyword.
  - A *delimiter* token is a string constant, a delimited identifier, an operator symbol, or any of the special characters shown in the syntax diagrams.

- **Cursor**

  An SQL cursor is the same as a file cursor. The cursor is used to point to the record in the file that is to be processed next.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

- **Result set**

  An SQL result set is a group of rows that are derived as a result of an SQL query. Result sets are derived as a result of processing a SELECT statement and satisfying certain criteria that are defined within the SELECT statement.

While we have discussed a number of SQL terms, there are many more.

Go to the *IBM i Information Center* website to view IBM documentation containing more SQL terms.

Included in the appendix at the back of this course notebook, there is a glossary of SQL terms for your reference.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — We emphasize using the SQL names for objects rather than the traditional term. For example, we refer to a table rather than a physical file.

Review each of the terms described in the notes to ensure that the students understand what they are.

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U01** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U01** and click play.

c. Ask the students to indicate using green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U01** and ask the students to call out the answers to the questions on the screen.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: An IBM i logical file is known as a view in SQL terminology.

2. True or False: SQL has always been known as Structured Query Language.

3. True or False: The interactive SQL facility (invoked by the command STRSQL) is included as part of the base operating system.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-8. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: An IBM i logical file is known as a view in SQL terminology.

   The answer is True.

2. True or False: SQL has always been known as Structured Query Language.

   The answer is False. It was initially known as SEQUEL, Structured English Query Language. A copyright conflict was discovered in the late 1970s and the name was changed to SQL at that time.

3. True or False: The interactive SQL facility (invoked by the command STRSQL) is included as part of the base operating system.

   The answer is False. No, you must purchase the DB2 Query Manager for i and SQL Development Kit. It is bundled in certain enterprise packages.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Getting to know SQL*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-9. Exercise OD477.0

Notes:

Perform the workstation exercise *Getting to know SQL*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Get the students on the workstation early so that they can learn that SQL allows them to use tools and IBM i/OS objects with which they are familiar.

Details — Introduce the exercise environment. Student IDs and passwords have been set up automatically for them. Orient them to this when you introduce the first exercise. Remember, the password is set to expire at first sign-on.

For this lab, we use the 5250 interface, which the students have used before. In subsequent exercises, they may use the 5250 interface and IBM i/OS tools, but we strongly recommend that they at least try to use System i Navigator at some point during lab exercises.

System i Navigator is the GUI interface that enables the SQL user to create and Run SQL Scripts (like STRSQL), to view SQL tables, to run CL commands, and to use the Database Navigator. System i Navigator is IBM's strategic tool for SQL interface to the system. Tools such as STRSQL are stable and will not be enhanced.

The students create personal schemas based on userids assigned to them. They are of the form:

`User id`	`OL37`*`nn`*
`Password`	**`OL37pwd`**

If a student is named team number 01, his or her user id would be OL37*01*. The student should follow the exercise instructions and create a library named OL37*01*LIB.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Explain the meaning of the following SQL terms:
  - Schema and collection
  - Table
  - View
  - Row
  - Column
  - Key
  - Index
- Describe the origins of SQL
- Name two significant applications where SQL can be used

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 2-10. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 3. Extracting data using SELECT

Estimated time

04:00

What this unit is about

This unit describes how to use SQL Data Manipulation Language (DML) to define, access and manipulate the IBM i database. We explain how to use the SELECT statement and how to use selection criteria and predicates.

This unit uses the prototyping capability of the interactive STRSQL interface, as well as its ability to save your session for subsequent execution in a group of SQL statements or for inclusion in program code. While STRSQL was never designed to be an intuitive user facility, you will learn that its prompting and Help are good productivity aids.

System i Navigator's ability to create and Run SQL Scripts is shown as an alternate query interface. System i Navigator is much easier to use than the 5250's STRSQL.

We also learn to use the `RUNSQLSTM` command to execute SQL statements that are stored in source members.

What you should be able to do

After completing this unit, you should be able to:

- Use the CREATE statement to build a table, view, or an index
- List the data types supported by SQL
- Write SQL queries using SELECT
- Use predicates to define the scope of your query
- Use the 5250 STRSQL tool to prototype SQL statements
- Use the RUNSQLSTM facility to run a group of SQL statements
- Use System i Navigator to create and Run SQL Scripts
- Use System i Navigator to display tables

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercises

The students will perform several exercises to extract data from a class schema. Several queries will be written that will reinforce the student's understanding of how to write interactive SQL queries.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Use the CREATE statement to build a table, view, or an index
- List the data types supported by SQL
- Write SQL queries using SELECT
- Use predicates to define the scope of your query
- Use the 5250 STRSQL tool to prototype SQL statements
- Use the RUNSQLSTM facility to run a group of SQL statements
- Use System i Navigator to create and Run SQL Scripts
- Use System i Navigator to display tables

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

3.1. Creating the database

Instructor topic introduction

What students will learn — The students will learn how data is described using Data Definition Language before they learn to access databases on the IBM i.

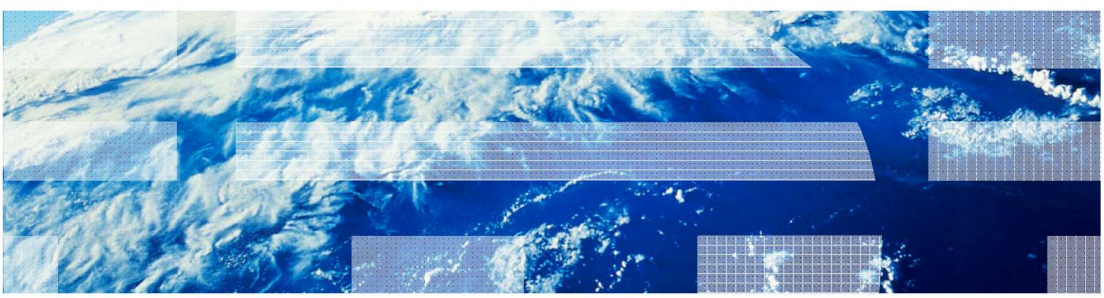
Instructor notes — We will focus our attention on those SQL Data Definition Language and Data Manipulation Language statements that define and provide data for us to query:CREATE and INSERT. We will use examples that will illustrate how the course database was defined and created.

While you are covering the various statements in this topic, please point out often that SQL statements can be executed interactively or can be used in a program to create and manipulate data.

Remember that you can still access IBM i physical and logical files without having to perform any conversion from native to SQL defined objects.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-2. Topic 1: Creating the database OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

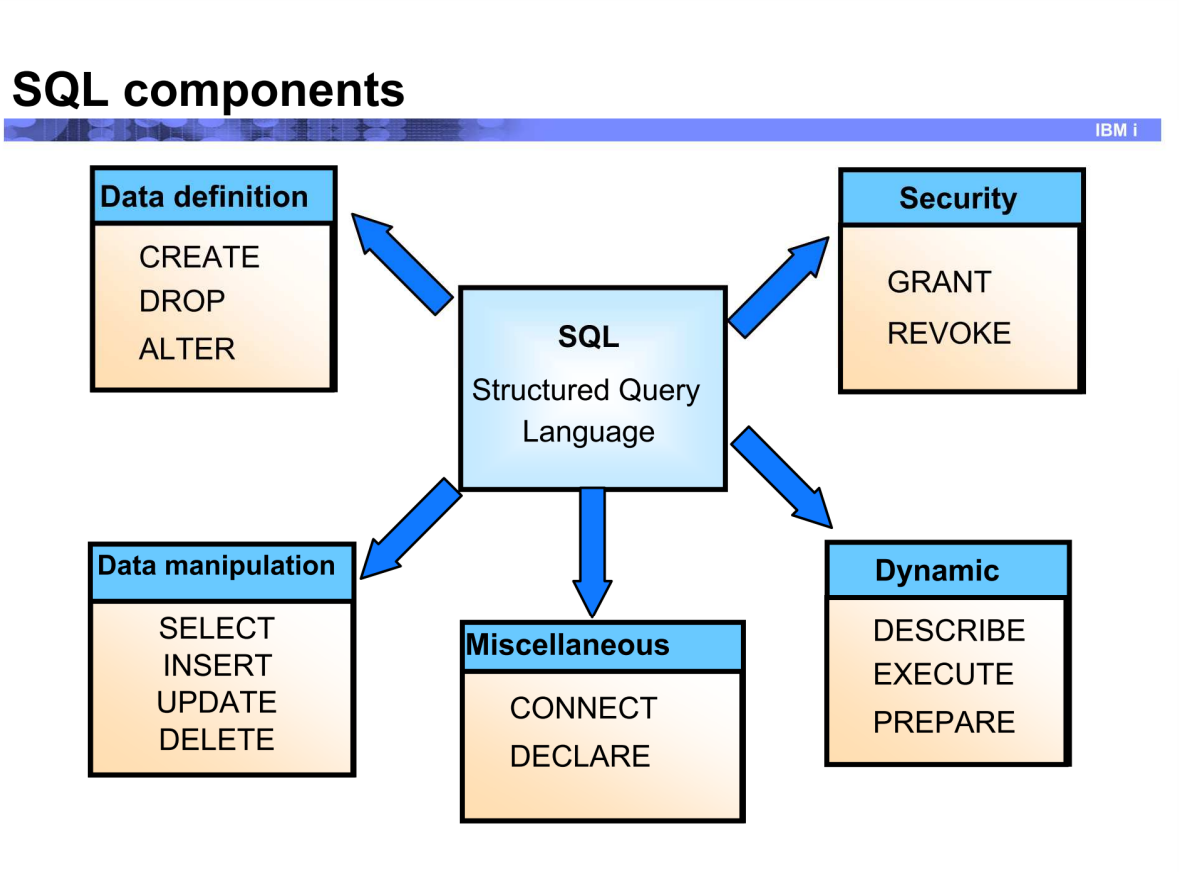
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-3. SQL components OD477.0

Notes:

In this topic, we discuss Data Definition Language (DDL) and Data Manipulation Language (DML) statements that you can use to create or modify SQL objects:

- Data Definition Language and Data Manipulation Language are integral parts of the SQL language.
- The SQL programmer describes data using DDL.
- The SQL programmer can also CREATE, ALTER, or DROP SQL schemas, tables, and views.
- The SQL programmer can INSERT new rows, UPDATE existing rows, or DELETE existing rows from a table.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

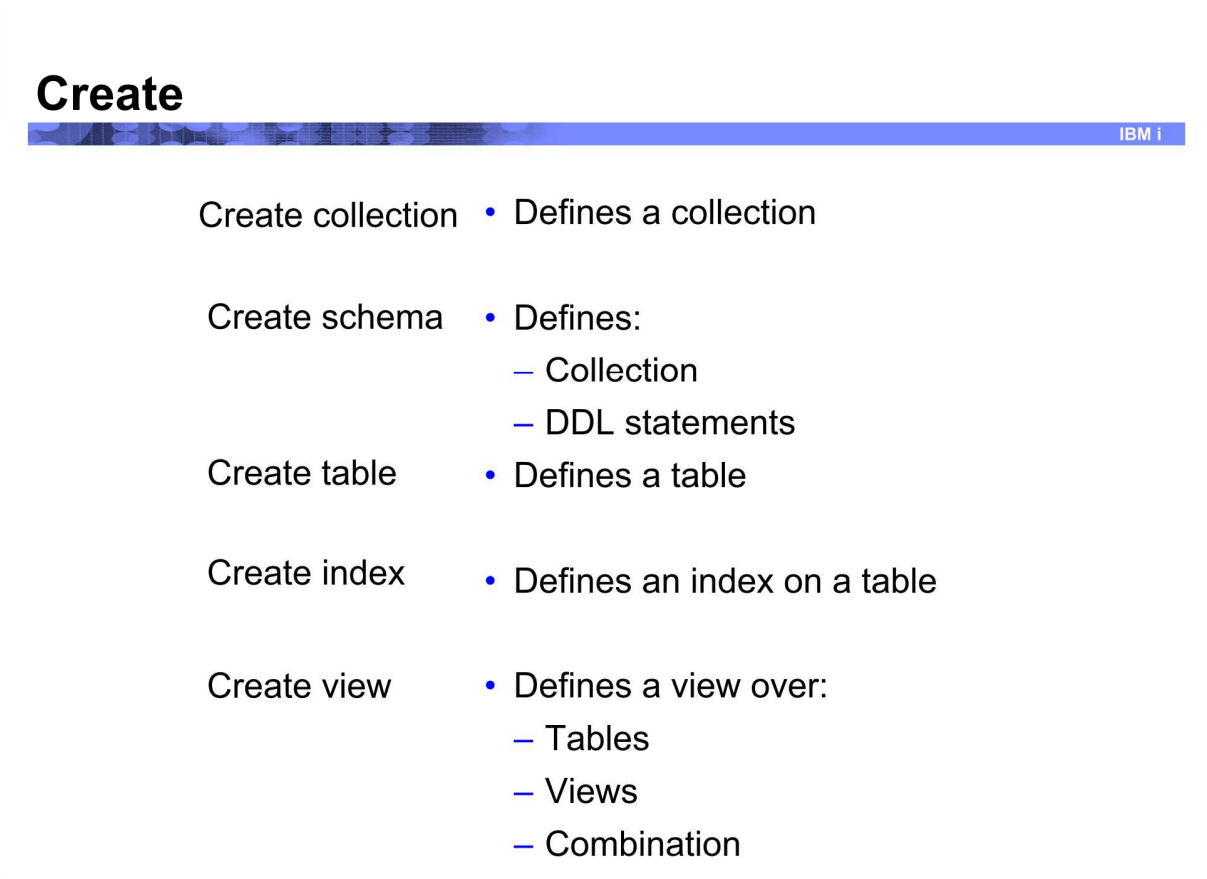
Purpose — Focus on the DDL and DML statements that are highlighted. We discuss these statements in this topic. GRANT and REVOKE follow in a later unit.

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-4. Create OD477.0

Notes:

CREATE has several purposes. This visual summarizes them. CREATE SCHEMA and CREATE COLLECTION are similar to CRTLIB.

The CREATE COLLECTION statement creates a schema in which tables, views, indexes, and packages can be created.

The CREATE SCHEMA statement defines a schema and optionally creates tables, views, aliases, indexes, and distinct types. Comments and labels can be added in the catalog description of tables, views, aliases, indexes, columns, and distinct types. Table, view, and distinct type privileges can be granted to users.

Note: The *SQL Reference Manual* states that beginning with version 5, CREATE SCHEMA supersedes CREATE COLLECTION: "*The terms collection and schema are synonymous. In prior versions of this book, the term collection was generally used. This and future versions of this book will generally use the term schema, because it is the standard term used by most database products.*"

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CREATE TABLE is similar to CRTPF.

CREATE INDEX and CREATE VIEW are similar to CRTLF.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — All of these commands, except CREATE SCHEMA, can be run using the interactive SQL interface. CREATE SCHEMA often requires using RUNSQLSTM.

CREATE SCHEMA can be used (as of V5R1) interactively through STRSQL provided that you are only creating a library and not tables. Otherwise an SQL script must be used with RUNSQLSTM if you are creating views indexes, inserting rows, and so on).

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create schema

IBM i

- Create schema CARCLUB
- Repository of:
  - Tables
  - Views
  - Indexes
- Created as a IBM i library
- Also created automatically in schema:
  - Catalog
  - Journal
  - Journal receiver

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-5. Create schema OD477.0

Notes:

The CREATE SCHEMA statement can be used to create a schema (library), and optionally creates tables, views, aliases, indexes, and distinct types. Comments and labels can be added in the catalog description of tables, views, aliases, indexes, columns, and distinct types. Table, view, and distinct type privileges can be granted to users.

In this visual, we look at CREATE SCHEMA.

Schemas contain objects like a library. In contrast to CRTLIB, a schema also automatically contains a catalog as well as a journal and a journal receiver.

When a table, view, or index is created, it is assigned to exactly one schema. That is, when you create a table, you assign it to a specific schema in the CREATE statement in a manner similar to the way in which you assign a file to a library.

Note: You can access existing IBM i physical files in your libraries, even though they were not created as SQL tables and they do not reside in SQL schemas. It is not necessary to move existing physical files into a schema. However, the physical files and its fields will not be documented in the SQL catalogs for tables and columns.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Schemas are created with the CREATE SCHEMA statement. A schema consists of a library, a journal, a journal receiver, a catalog, and optionally, a data dictionary. All files can be created or moved into an SQL schema if the SQL schema does not contain a data dictionary. If the SQL schema contains a data dictionary, IBM i source physical files or non-source physical files with one member can be created, moved, or restored into a SQL schema. However, logical files cannot be placed in an SQL schema because they cannot be described in the data dictionary.

Additional information — The data dictionary files are named `QIDCTP`*`t`*`nn` where the *t* is the type of file, either physical or logical.

Dictionary files are optional and only provided to support IDDU (S/36). That is why they are not included in the visual. In most cases, a dictionary is not necessary and should not be built.

Transition statement —

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Table: Cars

IBM i

Column

CARNUM	MEMBERNUM	CARMAKE	CARMODEL	CARYEAR
1	1	Oldsmobile	Cutlass 442	1967
2	1	Chevrolet	Corvette	1967
3	2	Ford	Mustang	1965
4	3	Chevrolet	Corvette	1985
5	5	Chevrolet	Corvette	1965
6	6	**Ferrari**	308	1984
7	7	Chevrolet	El Camino	1977
8	1	Ford	Taurus SHO	1995
10	1	Maserati	Diablo	1971
11	10	Chevrolet	Chevelle SS454	1970

Row

VALUE=intersection of row and column

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-6. Table: Cars OD477.0

Notes:

A table:

- Has an unique name, in this case CARS.
- Consists of rows and columns.
- The intersection of a row and a column is known as a *value*.
- All values in a column are the same data type.
- 0, ( ) (blank), and NULL are unique and different values.
- Every column is available to an SQL statement.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Is created as a physical file:
  - Maximum number of columns is 8000
  - Maximum length of a row is 32766 bytes
  - Update/Add/Delete capable
  - Size = *NOMAX
  - Contiguous = *NO
  - Allocate = *NO
  - Name Length = 30

In this table, CARS, the columns are defined as follows:

COLUMN NAME	DATA TYPE	LENGTH	SCALE
CARNUM	DECIMAL	3	0
MEMBERNUM	DECIMAL	3	0
CARMAKE	CHAR	15	
CARMODEL	CHAR	15	
CARYEAR	DECIMAL	4	0

In lecture, we have been using a Car Club organization as a source of examples to show you during class. Our club is made up of members who are interested in owning and enjoying old cars. For the purposes of examples, there are two tables in this application:

1. **MEMBERS:** Contains names and addresses of members of the car club. An individual can be a member of the club even if he or she does not own a car that qualifies as an old car. Interest is sufficient.
2. **CARS**: Contains information on the vehicles owned by club members. It cross references to the MEMBERS table using the member number. Each member can own zero or many old cars. A car qualifies as an old car if it is more than ten model years old.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This visual describes the terms you need to know when viewing a table. Focus the students' attention on the terms: row, column, and value. Mention that this table is one that we will be using as a sample table in the class.

Remind the students that the CARS table has a related table, named MEMBERS. Each row in the MEMBERS table identifies a unique individual of a car club (people who enjoy and show or parade their cars). Each row in the CARS table identifies a unique car that is owned by a MEMBER of the club. Each member can own no cars or many cars.

Details — Since 1978, we have been calling tables (the relational term), physical files. Each table has a unique name.

A table contains a description of the data, as well as the data itself. A table consists of rows and columns.

Each row of a table has the same columns as all other rows. Each column contains values of the same data type. Data values are found at the intersection of rows and columns. There is only one data value at each intersection.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Creating a table: Things to consider

IBM i

Field or Column Name	Type	Length	Decimals/ Scale	Null Capable
MEMBERNUM				
LASTNAME				
FIRSTNAME				
DUESAMOUNT				
STREET				
CITY				
STATE				
ZIP				

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-7. Creating a table: Things to consider OD477.0

Notes:

To define a table, in addition to providing a name for the table, you must include the names and attributes of its columns. The definition can include other attributes of the table, such as the primary key.

Try to fill in the blank spaces in the above figure. Use your knowledge of data types to decide what the data type should be for each column. We will review them as a class and discuss different choices.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Use the following to help you:

```
MEMBERNUM   - Membership Number
LASTNAME    - Last Name
FIRSTNAME   - First Name
DUESAMOUNT  - Amount of Dues Paid
STREET      - Street Address
CITY        - City or town
STATE       - State or Province
ZIP         - ZIP or Postal Code
```

Here is some sample data from the MEMBERS table:

```
MEMBERNUM  LASTNAME        FIRSTNAME    DUESAMOUNT  STREET                  CITY          STATE   ZIP

     1     Slaney          Bob          35.00     3325 Golf Course Rd.      Atlanta       GA    30338
     2     Humphries       Ernest       35.00     67 Elsemere Ave.          Norfolk       VA    20248
     3     Shrauger        Mary         35.00     490 Grand Ave.            Owego         NY    30101
     4     Moline          Jim          35.00     125 Welch Road            Ithaca        PA    30303
     5     Armstrong       Doug         35.00     25 Slate Bottom Creek Dr. West Seneca   PA    30556
     6     Jamison         Annette      35.00     9 White Pine Road         Bethesda      NY    20350
     7     Williams        Diane        35.00     923 Eastridge Road        Timonium      PA    10216
    10     Simpson         Gregory      40.00     33 Mystic Lane            Williamsville PA    13309
     :
     :
```

The option of declaring null capable columns might be new. In SQL, the Nulls option is implemented as an option on the CREATE or ALTER statements. Nulls can be one of three possible values for our purposes:

1. NULL capable column
2. Not NULL capable
3. Not NULL capable with default where an initial value is automatically assigned based upon the data type. For example, a character column would be assigned the initial value of ' ' (blank) and a numeric column would be assigned an initial value of '0' (zero).

Note: The default that is used by SQL is NOT NULL WITH DEFAULT (option **3** in the visual).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Tell the students that this visual is derived from the prompted STRSQL interface on the system.

Details — We want the students to try to determine data types for a sample table. This particular table is the MEMBERS table. It is related to the CARS table that we showed earlier. This table contains one row for each member of the car club. The students can use the data type topic to help them. This visual is extracted from the prompted SQL STRSQL interface. Note that the default set by the STRSQL interface for the *nulls* attribute for each row is NOT NULL WITH DEFAULT (3).

Ignore the FOR Field prompt for now. If anyone asks you, this provides an i5/OS name for the column. This name is optional. If the column name is more than 10 characters long and the FOR column name is not specified, an i5/OS name will be generated when the table is created.

Let the students take five minutes and decide individually the data type and possible length for each column.

Review the suggested answers below with the students and discuss alternatives that they have chosen.

Specify CREATE TABLE Statement

Type information, press Enter.

File ..... MEMBERS Name Library ..... OL37V1COL Name, F4 for list

Nulls: 1=NULL, 2=NOT NULL, 3=NOT NULL WITH DEFAULT

Field FOR Field Type Length Scale Nulls MEMBERNUM DEC 3 0 3
LASTNAME CHAR 25 3 FIRSTNAME CHAR 15 3 DUESAMOUNT
DEC 5 2 3 STREET CHAR 25 3 CITY CHAR 15
3 STATE CHAR 2 3 ZIP DEC 5 0 3

File CONSTRAINT ..... N Y=Yes, N=No Distributed File ..... N Y=Yes, N=No

Any of the CHAR fields could be VARCHAR.

Once you have finished this step, ask the students which columns should be null capable. Have them mark their notes and then discuss their suggestions with the class.

```
MEMBERNUM  - Membership Number   - NOT NULL
LASTNAME   - Last Name           - NOT NULL
FIRSTNAME  - First Name          - NOT NULL
DUESAMOUNT - Amount of Dues Paid - NOT NULL WITH DEFAULT
STREET     - Street Address      - NULL
CITY       - City or town        - NULL
STATE      - State or Province   - NULL

ZIP        - ZIP or Postal Code  - NULL
```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

There are different acceptable answers. A null value might be acceptable for the address columns since a member of our club might have recently moved and not have a permanent address.

You could argue that the dues column could be null if we had not decided the amount of the dues or if we had not mailed the dues invoices out for this membership period.

You could also argue that ZIP should be SMALLINT instead of DECIMAL.

Lastly, the data types for LASTNAME, FIRSTNAME, and other character columns could be either CHAR or VARCHAR. CHAR is the recommended data type for this database.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create table

IBM i

```
create table carclub/cars

                (CARNUM     DECIMAL(3,0),

                 MEMBERNUM  DECIMAL(3,0),

                 CARMAKE    CHAR(15),

                 CARMODEL   CHAR(15),

                 CARYEAR    DECIMAL(4,0))
```

OR

Create table carclub.cars

OR

Create table cars

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-8. Create table OD477.0

Notes:

A *base table* is created using the CREATE TABLE statement and is used to hold persistent user data. A *result table* is a set of rows that the database manager selects or generates from one or more base tables.

Base tables can be thought of as input physical files from which you extract data to create result tables.

The CREATE TABLE statement illustrates the syntax of SQL. Note that:

- Parameters for the statement are enclosed in brackets.
- Parameters are separated by commas.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

In this visual, in the table **CARS**, the columns are defined as follows:

COLUMN NAME	DATA TYPE	LENGTH	SCALE
CARNUM	DECIMAL	3	0
MEMBERNUM	DECIMAL	3	0
CARMAKE	CHAR	15	
CARMODEL	CHAR	15	
CARYEAR	DECIMAL	4	0

Normally, by default, columns are defined as null capable when you enter the CREATE TABLE SQL statement. However, when you use the 5250 STRSQL tool and use its prompting feature, the CREATE TABLE statement assumes that each column is to be created NOT NULL WITH DEFAULT.

Notice the various forms of referencing a table in the visual:

1. CARCLUB/CARS: Is the i standard library/file form of accessing a table. This form is known as *SYS naming format and is fully qualified.
2. CARCLUB.CARS: Is the SQL standard format. This form is also fully qualified.
3. CARS: Is unqualified. Depending on the naming convention that you are using in your SQL session, the system will search for the CARS table differently:
   - When your session is set to use *SYS naming convention, your *CURLIB (or QGPL) will be searched.
   - When the session is set to use *SQL naming convention, the system uses your *USRPRF name as the library name.
   - For the exercises in this class, we will typically use the *SYS naming convention rather than the *SQL naming convention.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Remember that SQL supports data types in a way that is similar to DDS and RPG.

Additional information — Since a table and a physical file store data in the same way, we can access tables or physical files using SQL, as well as the traditional methods of data access.

The naming of SQL objects can again be discussed at this point. SQL allows a table to have a name longer than 10 characters. To avoid any problems with names, it might be a good idea to use past conventions by limiting the names of tables and views to 10 characters. It is possible to assign an alias to an SQL name.

You might mention that SQL tables default to REUSEDLT(*YES), which is the opposite of the `CRTPF` command.

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create table members

IBM i

```
CREATE TABLE CARCLUB/MEMBERS
         (MEMBERNUM DEC (3, 0) NOT NULL WITH DEFAULT,
          LASTNAME CHAR (25) NOT NULL WITH DEFAULT,
          FIRSTNAME CHAR (15) NOT NULL WITH DEFAULT,
          DUESAMOUNT DEC (5, 2) NOT NULL WITH DEFAULT,
          STREET CHAR (25) NOT NULL WITH DEFAULT,
          CITY CHAR (15) NOT NULL  WITH DEFAULT,
          STATE CHAR (2) NOT NULL WITH DEFAULT,
          ZIP DEC (5,0) NOT NULL WITH DEFAULT)
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-9. Create table members OD477.0

Notes:

This visual shows you the CREATE TABLE statement for the MEMBERS table.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This visual is the CREATE TABLE statement that we used in an earlier exercise where we discussed how to define various columns in the table.

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Labels and column headings

IBM i

```
For a Table or View:

LABEL ON TABLE CARCLUB/MEMBERS is 'Car Club Members Table'

For Columns:

  LABEL ON COLUMN CARCLUB/MEMBERS
    (MEMBERNUM  IS  'Member Number',
     LASTNAME  IS 'Surname',
     FIRSTNAME  IS  'First Name',
     DUESAMOUNT  IS  'Annual Dues Amount',
     STREET  IS  'Street Address',
     CITY  IS  'City',
     STATE  IS  'State',
     ZIP  IS  'Postal Code')
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-10. Labels and column headings OD477.0

Notes:

The LABEL ON statement adds or replaces labels in the catalog description of tables, views, or columns:

- Labels are enclosed in quotes.
- It is not mandatory to label each column.

Note that labels can not be specified with the CREATE TABLE statement. They are specified separately in the LABEL ON.

Labels are also used to produce column headings that are used when displaying or printing query results.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information — There is a way to make the labels break into tiers for reports (Query Manager) by starting each tier at 20 column intervals. In Query Manager, you can do this with underscores specifying the tier break.

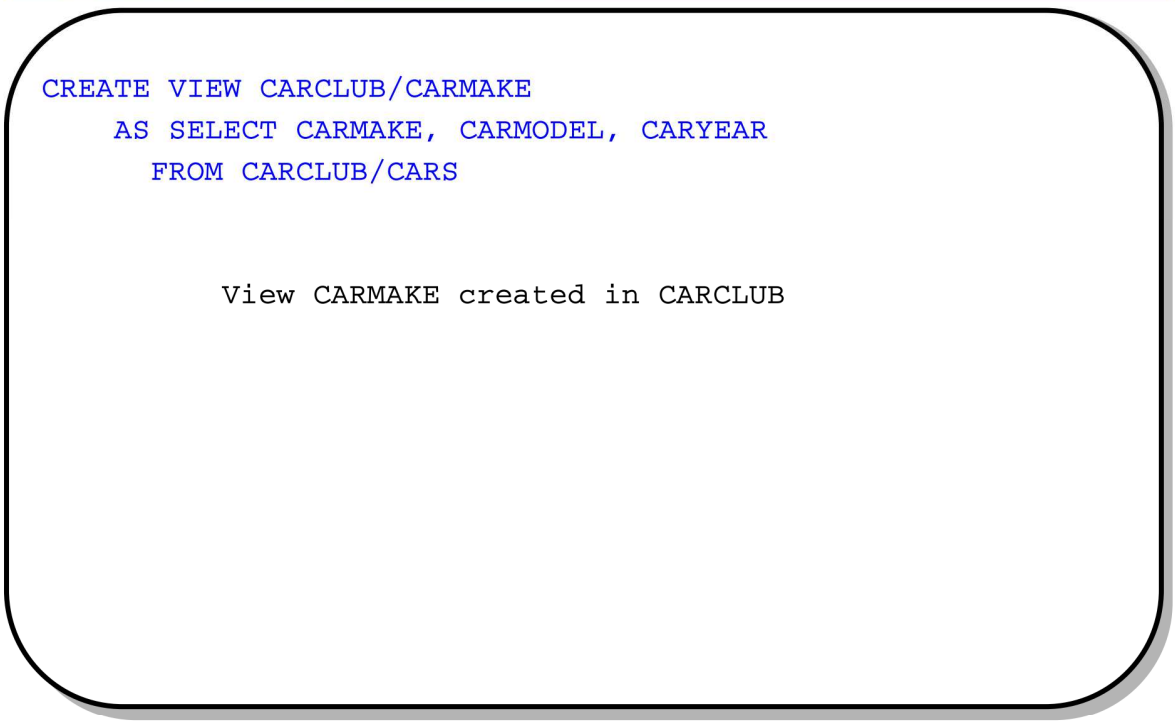
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create view

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-11. Create view OD477.0

Notes:

The name of the view must be unique. It cannot be the same as any table, view, index, or file that already exists on the system.

In the visual, CREATE VIEW is used to create a subset of the table CARCLUB. While you might guess that CREATE VIEW creates a keyed access path, it does not.

Some interesting facts:

- Views are *non-keyed logical files*.
- Views are created in the base table's schema.
- The maximum number of columns in a view = 8000.
- The maximum length of row in a view = 32766 bytes.
- A view can be created over one or more tables or other views.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — In the course schema, you see the view CARMAKE. Use option **5** of PDM to display the file description and emphasize to the students that all views are non-keyed LFs.

Mention that we use the SQL query statement SELECT to limit the columns that will be displayed in this view. We discuss SELECT in a separate topic.

Point out that we are only informing the students of the basics of VIEW and INDEX. We do not cover them extensively in class.

Use DSPFD and point out that the access path for the CARMAKE views is arrival.

Additional information — There are other facilities that can be enabled with a view. For example, the WITH CASCADED CHECK OPTION specifies that every row (of the table) that is inserted or updated through the view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that cannot be retrieved using that view.

For further information, students should reference the CREATE VIEW statement in the *SQL Reference*.

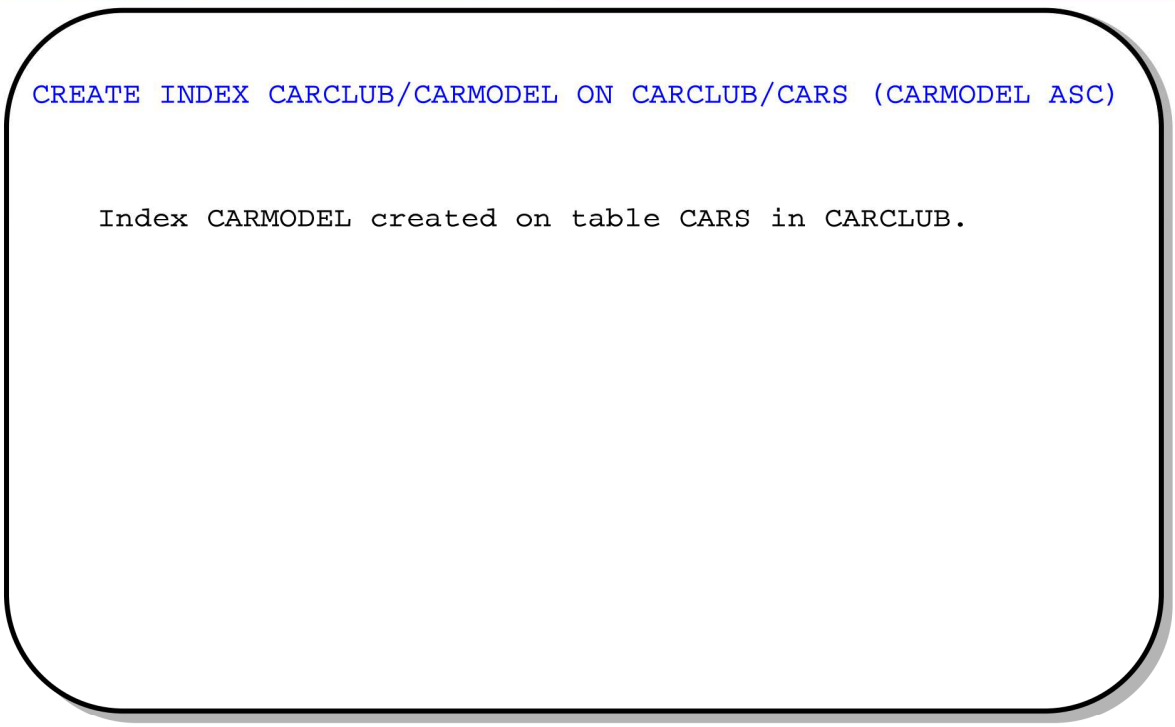
Transition statement — We use the CREATE INDEX statement to create an access path to a table or a view.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Create index

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-12. Create index OD477.0

Notes:

CREATE INDEX creates an access path for a table. Indexes are used to enhance performance:

- They are *not* part of a table.
- They are created as logical files.
- Indexes are created in the table's schema.
- UNIQUE prevents the table from containing two or more rows with the same value of the index key.
- Maximum key length = 2000 bytes.

An *index* is a keyed logical file that the system uses for faster data retrieval. A *key* is one or more columns that are identified as such in the description of an index. The same column can be part of more than one key.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — The students are used to creating physical and logical files that are access paths to data. SQL is different in that you do not specify an access path. DB2 for IBM i does this for you automatically, choosing the best possible access path based on analysis performed during every query by the Query Optimizer. Every index that you create is evaluated by the Optimizer. Where it makes sense to use an index, it will be done automatically without your intervention.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Composite key

IBM i

```
CREATE UNIQUE INDEX CARCLUB/makeyear
                  ON CARCLUB/CARS (CARMAKE, CARYEAR)

Index MAKEYEAR created in CARCLUB on table CARS in CARCLUB.
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-13. Composite key OD477.0

Notes:

A key composed of more than one column is called a *composite key*.

A composite key is an ordered set of columns of the same table. The ordering of the columns is not constrained by their ordering within the table. The term value when used with respect to a composite key denotes a composite value.

In this example, we are creating a composite key based on two columns. We asked that the index be created UNIQUE. This ensures that the index will not contain any duplicates.

When the index is created, it is immediately known to the Database Query Optimizer. You do not specify that a specific index should be used as an access path. The Query Optimizer does that for you.

We can also specify that we want to create the index as an Encoded Vector Index. This form of index is extremely efficient and can enable improved performance of SQL queries.

References:

DB2 for IBM i Database Performance and Query Optimization, The DB2 for IBM i Query Optimizer: Overview

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — While the students should know the purpose of an index at this point, we will not apply them in a practical manner in the exercise for this unit.

Use DSPFD and point out that the access path for this view is keyed.

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

INSERT statement: Add rows to table

IBM i

```
INSERT INTO CARCLUB/MEMTEST VALUES (00001, Slaney, Bob, 35.00, 3325 Golf Course Rd., Atlanta,
GA, 30338)

     Token SLANEY was not valid.  Valid tokens:  :.

INSERT INTO CARCLUB/MEMTEST VALUES (00001, 'Slaney', 'Bob', 3500, '3325 Golf Course Rd.',
'Atlanta', 'GA', 30338)

     Conversion error on assignment to column DUESAMOUNT.

INSERT INTO CARCLUB/MEMTEST VALUES (00001, 'Slaney', 'Bob', 35.00, '3325 Golf Course Rd.',
'Atlanta', 'GA', 30338)

     1 rows inserted in MEMTEST in CARCLUB.


INSERT INTO CARCLUB/MEMTEST VALUES (00001, 'Slaney', 'Bob', 35.00, '3325 Golf Course Rd.',
'Atlanta', 'GA', 30338),(00002, 'Smith', 'Christine', 35.00, '3325 Golf Course Rd.',
'Atlanta', 'GA', 30338)

     2 rows inserted in MEMTEST in CARCLUB.
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-14. INSERT statement: Add rows to table OD477.0

Notes:

The INSERT statement is used to add rows to a table or a view. While this statement is really part of data manipulation, we use it to add some data records to the MEMBERS table.

- For the character data type, the data must be enclosed in quotation marks. If you do not frame a character field in quotes, SQL notifies you that a token is not valid.
- For decimal data, include the decimal point in the field when it is entered. If you do not indicate the location of the decimal point, SQL notifies you with a message: `Conversion error on assignment to column DUESAMOUNT`.
- All tokens are converted to uppercase.

And, remember that comprehensive prompting is available to assist you in both STRSQL and in the SQL assist tool of Run SQL Scripts.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

There are three forms of the INSERT statement:

- The INSERT using VALUES form is used to insert a single row into the table or view using the values provided or referenced.
- The INSERT using SELECT form is used to insert one or more rows into the table or view using values from other tables or views.
- The INSERT using *n ROWS* form is used to insert multiple rows into the table or view using the values provided in a host structure array.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Point out that the first two INSERTs have errors. The third one is successful. Also, note that SQL converts all tokens to uppercase.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

3.2. SQL data types

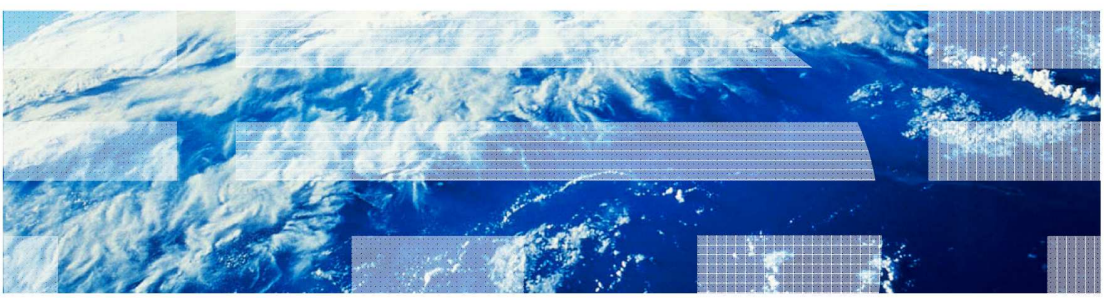
Instructor topic introduction

What students will learn — The students will learn what data types are supported in SQL.

Host Variables, LOBs — We will not cover host variables in this course. Host variables are discussed in OL38, *Developing IBM i Applications using SQL*. We will only introduce Large OBjects (LOBs) as an SQL supported data type. LOBs are covered in more detail in OL39, *DB2 for IBM i Advanced SQL Programming*.

In this topic, we will briefly discuss the data types supported by SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-15. Topic 2: SQL data types OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

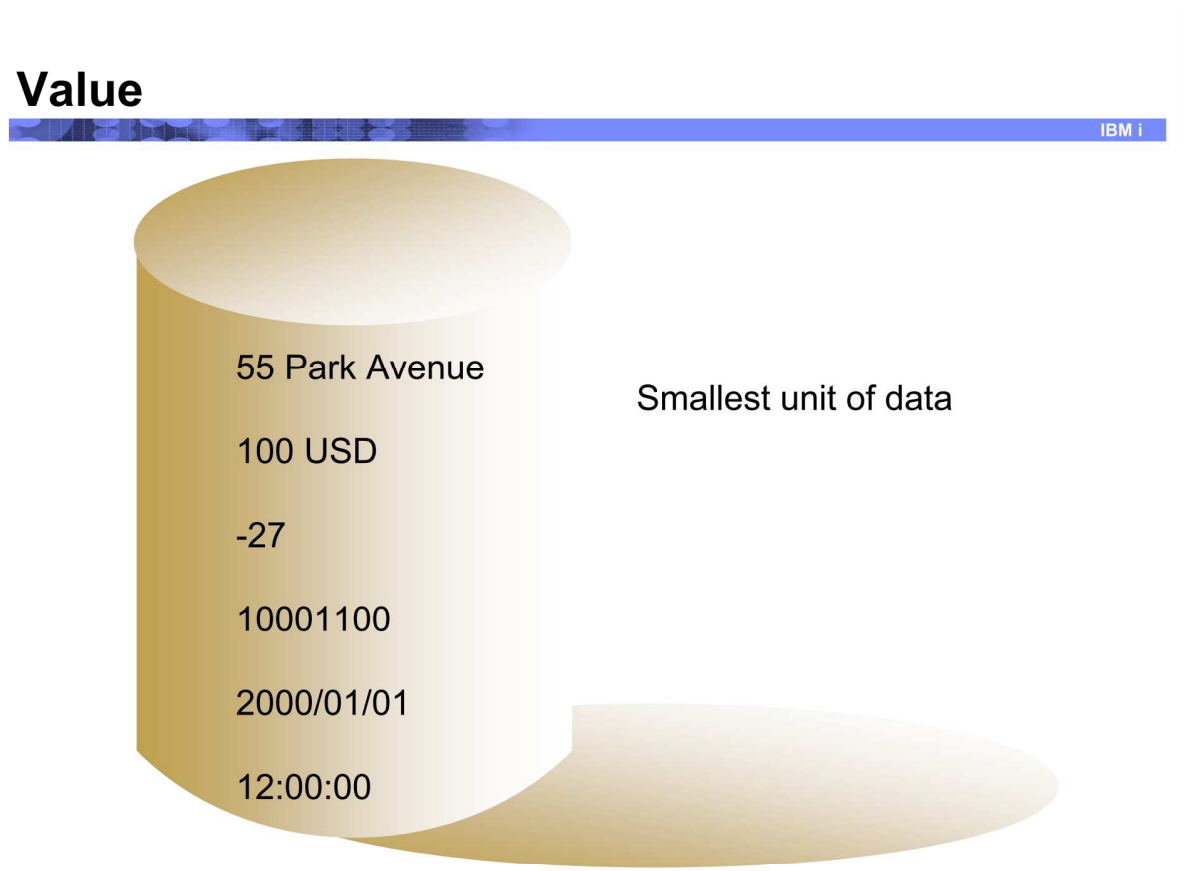
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-16. Value OD477.0

Notes:

The smallest unit of data that can be manipulated in SQL is called a *value*. How values are interpreted depends on the data type of their source.

Some of the sources of values are:

- Columns in an SQL table or view
- Constants in an SQL statement
- Expressions in an SQL statement
- Functions in an SQL statement

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This should be a quick review of some of the basic concepts that most of your students may have covered in other courses.

Transition statement — The following visuals explain the various data types supported by SQL on IBM i.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Nulls

IBM i

- Special value
- Opposite of a non-null value where the column contains a value:
  - 12:00:00 is non-null
- Any column can contain a null value unless defined NOT NULL.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-17. Nulls OD477.0

Notes:

All data types include support of the null value.

The *null value* is a special value that is distinct from all non-null values and thereby denotes the absence of a value.

A value of “ “ (blank) or “0” (zero) is non-null.

Although all data types include support of a null value, columns defined as NOT NULL cannot contain null values. They can be specified NOT NULL WITH DEFAULT and will be initialized based upon the data type of the column. For example, a character variable would be initialized to blanks and a numeric variable would be initialized to zero.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Data types

IBM i

- Every column is assigned a specific data type and size.
- Data types:
  - Numeric
  - Character
  - Graphic
  - Date, time, and timestamp
- Plus:
  - User-defined data types
  - Large objects (LOBs)
  - Datalinks

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-18. Data types OD477.0

Notes:

These are the data types supported by SQL. We will not be covering user-defined data nor large objects (LOBs) in this class.

For each column that you define using SQL, you must assign:

1. Column name
2. Data type and length
3. Null characteristic

References:

DB2 for i SQL Reference, Language Elements

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Numeric data types

IBM i

Data type	Rules
SMALLINT	2-bytes binary -32768 to +32767
INTEGER	4-bytes binary -2,147,483,648 to +2,147,483,647
NUMERIC (n,d)	zoned decimal maximum 63 digits n=length (maximum 63) d=decimals (maximum 63)
DECIMAL (n,d)	packed decimal n=total number of digits (maximum 63) d=number of decimal digits (maximum 63)
FLOAT (n)	very large or very small values single (4-bytes) or double precision (8-bytes)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-19. Numeric data types OD477.0

Notes:

Columns of the numeric data type are usually described using precision and scale. Precision describes the number of digits. Scale describes the number of decimal places and must be less than or equal to the precision.

Normal system rules for the numeric data type apply:

- For large integers, decimal precision and scale are supported by both COBOL and RPG IV. INTEGER can be written as INT.
- A single-precision floating-point number is a 32-bit approximate representation of a real number.
- A double-precision floating-point number is an IEEE 64-bit approximate representation of a real number.
- FLOAT(n) where:
  - n = Precision in bits
  - Single precision (n = 1 to 24)

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Double precision (n = 25 to 53)

- A DECIMAL (or DEC) data type is a packed-decimal number with an implicit decimal point.

  A NUMERIC data type is a zoned-decimal number with an implicit decimal point.

The position of the decimal point is determined by the precision and the scale of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits until V5R2 and 63 digits starting with V5R3.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — The data type is the description of the data of a certain column.

Details — Point out that the ranges for SMALLINT and INTEGER need to be carefully considered. DDS and RPG limit them to +-9999 and +-999,999,999 respectively.

While SQL supports the full range in an INSERT statement, other methods of data access, such as DFU, impose limitations. Make sure you know how the data will be accessed and used.

Additional information — How large a numeric field should be can be determined once you perform some arithmetic operations on it. Doing this will help you to decide on the data type to use.

For instance, a SMALLINT can store whole numbers between -32768 and +32767 only.

Not mentioned is the BIGINT data type, a binary number composed of 8 bytes (64 bits) with a precision of 19 digits. The range of big integers is -9223372036854775808 to +9223372036854775807.

Transition statement — You cannot perform arithmetic operations on alphanumeric fields.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Alphanumeric data types

IBM i

Data type	Rules
CHAR(n) or CHARACTER(n)	STRING (fixed length) maximum 32766
VARCHAR(n)	STRING (variable length) maximum 32740

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-20. Alphanumeric data types OD477.0

Notes:

Alphanumeric values must be enclosed in quotes when you are retrieving data from or inserting data into a table.

All values of a fixed-length character-string column have the same length. This is determined by the length attribute of the column. The length attribute must be between one and 32766 inclusive.

All values of a varying-length character-string column have the same maximum length, which is determined by the length attribute of the column. The length attribute must be between one and 32740 inclusive.

The length of the variable length field is stored in a 2 byte prefix.

You can use ALLOCATE(n) to specify VARCHAR and VARGRAPHIC, which determines the space to be reserved for the column in each row.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Advise the students that they should consider carefully when to use VARCHAR versus CHAR defined columns. CHAR stores data in a fixed-length, padding for the bytes not used. VARCHAR uses variable-length storage, but has overhead consisting of 2 bytes for field length and 16 bytes of pointers to the data.

Additional information — The programmer's guide has a discussion about using fixed versus variable-length columns that is interesting. If some students are interested in a discussion, you might use this as background information or review the *Programmers' Reference* document. Read the topic tips for using VARCHAR and VARGRAPHIC data types.

Variable-length column (VARCHAR or VARGRAPHIC) support allows you to define any number of columns in a table as variable length. If you use VARCHAR or VARGRAPHIC support, the size of a table can usually be reduced.

Data in a variable-length column is stored internally in two areas: A fixed-length or ALLOCATE area and an overflow area. If a default value is specified, the allocated length is at least as large as the value. The following points help you determine the best way to use your storage area.

When you define a table with variable-length data, you must decide the width of the ALLOCATE area. If the primary goal is:

- Saving space: Use ALLOCATE(0).
- Performance: The ALLOCATE area should be wide enough to incorporate at least 90% to 95% of the values for the column.

It is possible to balance space savings and performance. If you consider an example of an electronic phone book, the following data is used:

- 8600 names identified by last, first, and middle name.
- The Last, First, and Middle columns are variable length.
- The shortest last name is two characters; the longest is 22 characters.

In this case, we can show how space can be saved by using variable-length columns. The fixed-length column table uses the most space.

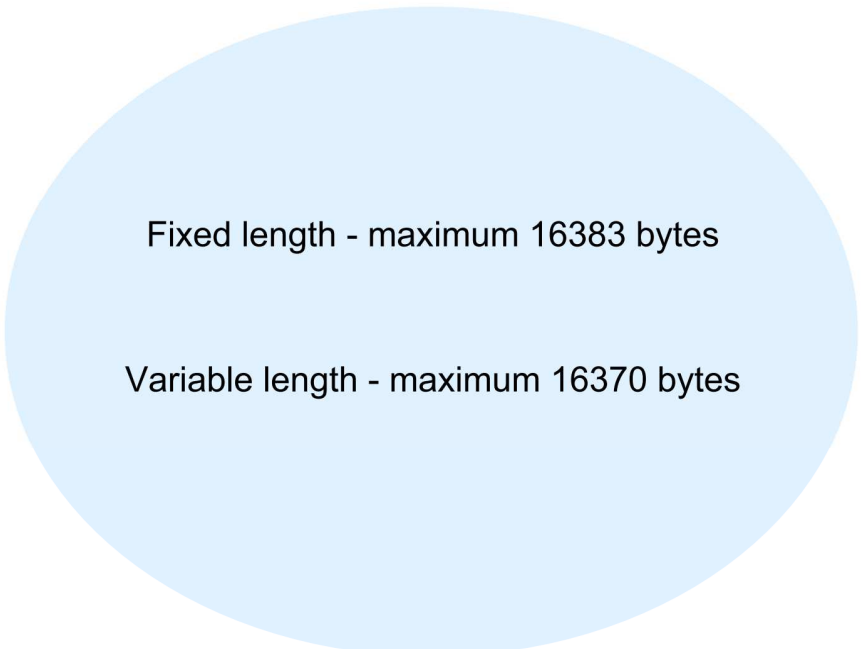
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Graphics data type

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-21. Graphics data type OD477.0

Notes:

- **Fixed length**

  All values of a fixed-length graphic-string column have the same length, which is determined by the length attribute of the column. The length attribute must be between one and 16383 inclusive.

- **Variable length**

  The values of a varying-length graphic-string column can have different lengths. The maximum length is determined by the length attribute of the column. The length attribute must be between one and 16370 inclusive.

The graphic data type is a double byte character data type.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date data type

IBM i

Data type	Rules		
Date	Internal Length: 4 bytes Format: Scalinger		
	External		
	USA	mm/dd/yyyy	(10 bytes)
	EUR	dd.mm.yyyy	(10 bytes)
	ISO	yyyy-mm-dd	(10 bytes)
	JIS	yyyy-mm-dd	(10 bytes)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-22. Date data type OD477.0

Notes:

Everyone should be aware that date support on the i SQL includes the ability to determine durations. We discuss this facility later.

The formats available follow:

***USA**	USA Standard (mm/dd/yyyy)
***EUR**	European Standard (dd.mm.yyyy)
***ISO**	International Standards Organization (yyyy.mm.dd): This is the default.
***JIS**	Japanese Industrial Standard (yyyy.mm.dd)
***MDY**	mm/dd/yy
***DMY**	dd/mm/yy
***YMD**	yy/mm/dd
***JUL**	yy/ddd
***JOB**	The format specified for the job. Check DSPJOB for the format.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — You might get some questions about the storage required for the date data type. The following information should help you answer any questions.

Note that the date (and time) column size depends on the format that is chosen. However, this is the “externalized” view of the date. Internally (in a database file), dates are stored as 4-byte values.

Additional information — Dates are stored in the database as Scalinger numbers. This method (of Julian Day Numbering) was introduced by Joseph Scalinger (1540-1609). For more history, you can browse:

http://www.friesian.com/numbers.htm

The external column sizes for dates are the resolved (buffer) lengths only. Even DSPPFM will *expand* date (and time) columns to a formatted form.

Transition statement — For the data type TIME, you have similar representation choices as with date fields (columns of data type DATE).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Time data type

IBM i

Data type	Rules		
Time	Internal Length: 3 bytes		
	External		
	USA	hh:mm AM or PM	(8 bytes)
	EUR	hh.mm.ss	(8 bytes)
	ISO	hh.mm.ss	(8 bytes)
	JIS	hh:mm:ss	(8 bytes)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-23. Time data type OD477.0

Notes:

The formats available follow:

***USA**	USA Standard
***EUR**	European Standard
***ISO**	International Standards Organization
***JIS**	Japanese Industrial Standard
***HMS**	Hour: Minutes: Seconds

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Times are stored in three bytes with each byte holding two digits in the form *hhmmss*.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Timestamp data type

IBM i

Data type	Rules
Timestamp	Internal Length: 10 bytes
	External yyyy-mm-dd-hh.mm.ss.nnnnnn

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-24. Timestamp data type OD477.0

Notes:

TIMESTAMP is a seven part value composed of year, month, day, hour, minute, seconds, and microseconds.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Timestamps are stored in 10-byte composite numbers composed of the 4-byte date, the 3-byte time, and the final 3-bytes containing the microseconds in packed form.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Other data types

IBM i

Large objects (size limit = 2 GBs)

- BLOB (binary large object)
  - Images, sound, and so forth
- CLOB (character large object)
  - Text and documents
- DBCLOB
  - Double byte character set large object
- Access to large objects
- Datalink
  - Used to access large objects from tables
- UDT (user defined type)
  - Usually based on existing SQL data types

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-25. Other data types OD477.0

Notes:

There are other data types that are not used in this class that you should be aware of when you use SQL to access data on your own systems:

- **BLOBs** and **CLOBs**: Large binary and character objects that contain data, such as images, sound, video, or large text documents.
- **Datalinks**: Are *pointers* stored in your regular data files that point to large objects. Datalinks can make access to large objects easier and faster.
- **UDTs**: Are user-defined types that are created by you to meet special business needs. They are usually based on an existing data type. For example, the data types *dollars* and *yen* would be based on a numeric data type. The data types would enable international currency exchange and would avoid confusion between items sold in Japan versus items sold in the US from the same company.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — These data types are explained in the OL39 *Advanced SQL Programming* course. Introduce these data types to the students and explain that we cover them in a more advanced class (which is for programmers only).

Details —

Additional information —

Transition statement — Now you know how tables are created and what data types can be defined in SQL. Next, we discuss how you query SQL objects with the SELECT statement.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

3.3. The SELECT statement

Instructor topic introduction

What students will learn — In this topic, we focus our attention on the capabilities of the SQL SELECT statement. We cover some of the functions that are available within the SELECT statement and its clauses.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

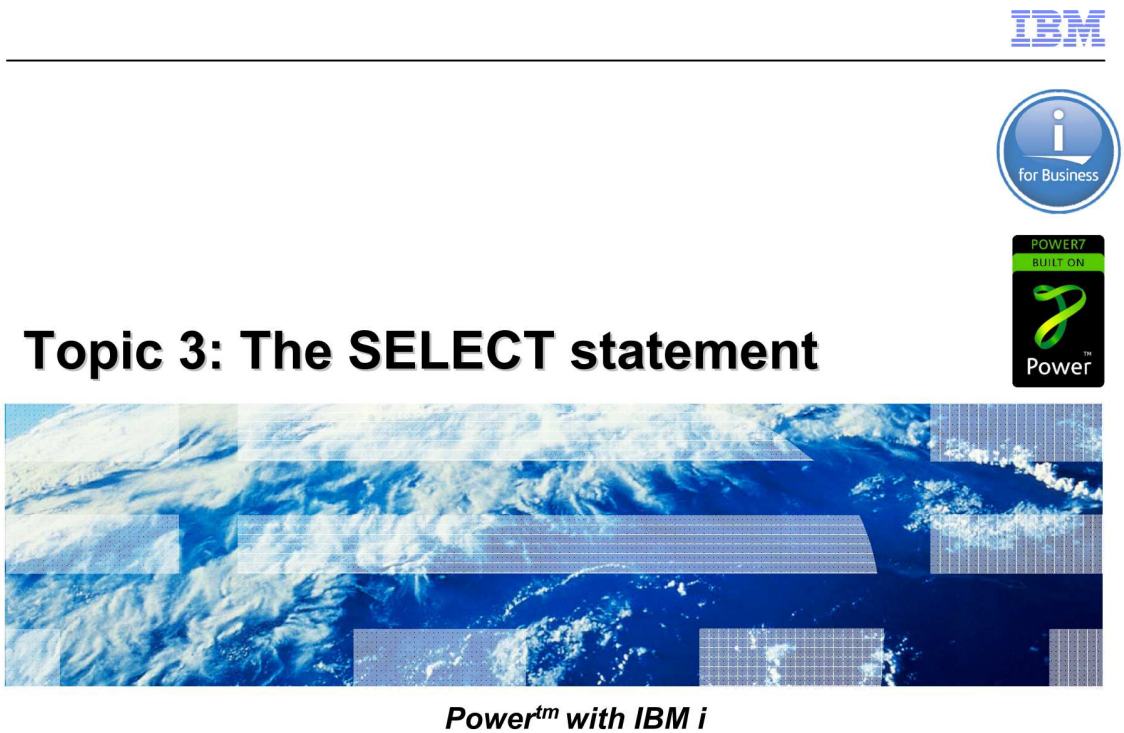


Figure 3-26. Topic 3: The SELECT statement OD477.0

Notes:

Let's look at the SELECT statement.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

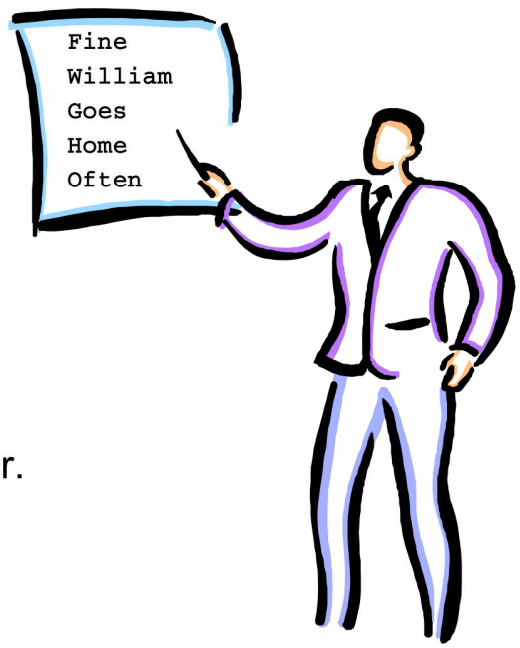
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT statement

IBM i

```
SELECT  ....
        FROM  ....
        WHERE  ....
        GROUP BY  ....
        HAVING  ....
        ORDER BY  ....
```



- Clauses *must* be used in this order.
- Output of SELECT = Result set

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-27. SELECT statement OD477.0

Notes:

The SELECT statement is the data retrieval statement in SQL. We will be using it in its simplest form with the clauses that it supports.

SELECT statements can also be nested within a SELECT statement using a form known as subselects. A *subselect* is a component of the *fullselect*, the CREATE VIEW statement, or the INSERT statement. A *fullselect* specifies a result table is to be generated. The *subselect* is also a component of certain predicates, which in turn, are components of a subselect. A subselect that is a component of a predicate is called a *subquery*.

A *predicate* specifies a condition that is true, false, or unknown about a given row or group.

1. **FROM**

   The FROM clause specifies that an intermediate result table will be produced. If only one table or view is specified, the intermediate result table is simply that table or view.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

2. **WHERE**

   The WHERE clause specifies that the intermediate result table will consist of those rows of a result set from the SELECT for which the search condition is true.

3. **GROUP BY**

   The GROUP BY clause specifies that an intermediate result table will be produced. It will consist of a grouping of the rows of the result set of the SELECT.

4. **HAVING**

   The HAVING clause specifies that an intermediate result table will be produced. It will consist of those groups of the result set of the SELECT for which the search condition is true.

5. **ORDER BY**

   The ORDER BY clause specifies an ordering of the rows of the result table. If a single column is identified, the rows are ordered by the values of that column. If more than one column is identified, the rows are ordered by the values of the first identified column, then by the values of the second identified column, and so on. Each column specified can be sorted in either ascending or descending order.

Order of clauses

While it is not necessary to code all clauses in a SELECT statement, the clauses that you do code *must* be coded in this order. When using interactive SQL, the F4 key will prompt in a different order than shown, but the generated statement will be in the correct order. For example, the system will prompt for schemas before tables before columns, as you would want it to do.

Tip

When coding using the interactive STRSQL interface, you should place each keyword of the SELECT statement on a separate entry line and indent it as illustrated in the visual. Your query will be easier to maintain in the future.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — In this topic, we cover the SELECT statement and its clauses in further detail.

Details — Point out to the students that the required structure of the SELECT statement,

```
SELECT   <columns>
         FROM   <tables>
         WHERE  <condition>
         etcetera
```

does not match the sequence followed by the prompting facility of STRSQL. The prompting, understandably, requests the tables first, before the specification of columns.

The phrase “Fine William goes home often” can be used to help students to remember the order of clauses in the SELECT statement. (I know that this will need to be changed for non-English speaking countries.)

Transition — Before we discuss SELECT further, let’s introduce our sample database for the lecture, which is based upon a car club, its cars, and its members.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Table: CARS

IBM i

Column

CARNUM	MEMBERNUM	CARMAKE	CARMODEL	CARYEAR
1	1	Oldsmobile	Cutlass 442	1967
2	1	Chevrolet	Corvette	1967
3	2	Ford	Mustang	1965
4	3	Chevrolet	Corvette	1985
5	5	Chevrolet	Corvette	1965
6	6	**Ferrari**	308	1984
7	7	Chevrolet	El Camino	1977
8	1	Ford	Taurus SHO	1995
10	1	Maserati	Diablo	1971
11	10	Chevrolet	Chevelle SS454	1970

Row

VALUE = intersection of row and column

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-28. Table: CARS OD477.0

Notes:

We have a sample database composed of several tables, one of which is the CARS table.

This database supports a car club. The club is made up two tables, CARS, and MEMBERS.

A table:

- Has an unique name, in this case **CARS**.
- Consists of rows and columns.
  - The intersection of a row and a column is known as a *value*.
  - All values in a column are the same data type.
  - **0**, **( )** (blank), and **NULL** are unique and different values.
  - Every column is available to an SQL statement.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Is created as physical file:
  - Maximum number of columns is 8000
  - Maximum length of a row is 32766 bytes
  - Update/Add/Delete capable
  - Size = *NOMAX
  - Contiguous = *NO
  - Allocate = *NO
  - Name Length = 30

In this table, CARS, the columns are defined as follows:

COLUMN NAME	*DATA TYPE*	*LENGTH*	*SCALE*
CARNUM	DECIMAL	3	0
MEMBERNUM	DECIMAL	3	0
CARMAKE	CHAR	15	
CARMODEL	CHAR	15	
CARYEAR	DECIMAL	4	0

In the lecture, we use a car club organization as a source of examples. Our club is made up of members who are interested in owning and enjoying old cars. For the purposes of examples, there are two tables in this application:

1. **MEMBERS**: Is a table that contains names and addresses of members of the car club. An individual can be a member of the club even if he or she does not own a car that qualifies as an old car. Interest is sufficient.
2. **CARS**: Is a table that contains information on the vehicles owned by club members. It cross-references to the MEMBERS table using the member number. Each member can own zero or many old cars. A car qualifies as an old car if it is more than ten model years old.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This visual describes the terms you need to know when viewing a table. Focus the student's attention on the terms: *row*, *column*, and *value*. Mention that this table is one that we use as a sample table in the class.

CARS has a related table named MEMBERS. Each row in the MEMBERS table identifies a unique individual of a car club (people who enjoy and show or parade their cars). Each row in the CARS table identifies a unique car that is owned by a MEMBER of the club. Each member may own no cars or many cars.

In subsequent visuals, we use CARS and MEMBERS to illustrate key capabilities of SQL.

Details — Give the students a brief introduction to the car club application (CARCLUB) that we are using as a sample database. We use two tables, CARS and MEMBERS, throughout the class as samples.

Since 1978, we have been calling tables (the relational term) physical files. Each table has a unique name.

A table contains a description of the data, as well as the data itself. A table consists of rows and columns.

Each row of a table has the same columns as all other rows. Each column contains values of the same data type. Data values are found at the intersection of rows and columns. There is only one data value at each intersection.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Table: Members

IBM i

Member number	Last name	First name	Dues amount	Street	City, State	Zip
1	Slaney	Bob	35.00	3325 Golf Course Rd.	Atlanta, GA	30338
2	Humphries	Ernest	35.00	67 Elsemere Ave.	Norfolk, VA	20248
3	Shrauger	Mary	35.00	490 Grand Ave.	Owego, NY	30101
4	Moline	Jim	35.00	125 Welch Road	Ithaca, PA	30303
5	Armstrong	Doug	35.00	25 Slate Bottom Creek Dr.	West Seneca, PA	30556
6	Jamison	Annette	35.00	9 White Pine Road	Bethesda, NY	20350
7	Williams	Diane	35.00	923 Eastridge Road	Timonium, PA	10216
10	Simpson	Gregory	40.00	33 Mystic Lane	Williamsville, PA	13309

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-29. Table: Members OD477.0

Notes:

This is the MEMBERS table. It contains name and address information, along with a member number (MEMBERNUM). The MEMBERNUM column is used to associate members with any cars they own that are in the club.

A member might not own a car, might own one car, or might own more than one car.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Review this second table that we use during the lecture to illustrate the use of SELECT statements.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT all data

IBM i

```
SELECT   * FROM   CARS

 CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

    1         1      Oldsmobile       Cutlass 442        1,967
    2         1      Chevrolet        Corvette           1,967
    3         2      Ford             Mustang            1,965
    4         3      Chevrolet        Corvette           1,985
    5         5      Chevrolet        Corvette           1,965
    6         6      Ferrari          308                1,984
    7         7      Chevrolet        El Camino          1,977
```

- Select ALL columns

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-30. SELECT all data OD477.0

Notes:

The * tells the DB2 for IBM i database to select all rows from the named table or view.

In the example, we see a display of all rows in the CARS table that we used in the previous unit.

Performance

SELECT * selects all columns from a table. In a production environment, you should be aware of the performance implications of using a query of this form. Wherever possible, avoid SELECT * and try to SELECT only those columns and rows that you need.

Question

Notice that the headings for the values of the CARS table are the column names. Can you guess what SQL statement could be used to make the column headings more meaningful? Refer to the *SQL Reference Manual* to find this information.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Use the following URL to access the IBM i Information Center, *SQL Reference Manual:*

http://publib.boulder.ibm.com/iseries/

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — In general, using SELECT * is not a good SQL programming technique for performance. Instead, the SQL programmer should list only those columns that SQL needs to process. SELECT * maximizes the amount of data that the system has to move and limits the implementation choices of the query optimizer.

Note that you might see the SELECT ALL statement in the documentation. SELECT ALL has another meaning in SQL that we cover in the unit titled Functions. You must use the asterisk to select all rows for your result set.

Answer to question: The LABEL ON statement could be used to make the column headings more meaningful. If the documentation is not readily available, find a copy in the IBM i Information Center and show the students the reference manual:

`http://publib.boulder.ibm.com/iseries/`

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT some columns

IBM i

```
SELECT CARYEAR, CARMAKE, CARMODEL FROM cars


CARYEAR  CARMAKE          CARMODEL

 1,967   Oldsmobile       Cutlass 442
 1,967   Chevrolet        Corvette
 1,965   Ford             Mustang
 1,985   Chevrolet        Corvette
 1,965   Chevrolet        Corvette
 1,984   Ferrari          308
 1,977   Chevrolet        El Camino
```

- Can change relative position of columns

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-31. SELECT some columns OD477.0

Notes:

When you use this form of the SELECT statement, you specify exactly those columns that you want to place in your result set.

You can specify only a subset of the columns you want to SELECT or you can use this form to resequence some or all columns in the result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Deriving a new column

IBM i

```
select caryear, carmake, carmodel, (2000 - caryear) from cars

CARYEAR             CARMAKE      CARMODEL      (2000 - CARYEAR)
1,967               Oldsmobile   Cutlass 442          33
1,967               Chevrolet    Corvette             33
1,965               Ford         Mustang              35
1,985               Chevrolet    Corvette             15
1,965               Chevrolet    Corvette             35
1,984               Ferrari      308                  16
1,977               Chevrolet    El Camino            23
1,995               Ford         Taurus SHO            5
1,971               Maserati     Diablo               29
1,970               Chevrolet    Chevelle SS 454      30
1,965               Austin       Healey               35
1,973               Alfa Romeo   Targa                27
1,969               Plymouth     Roadrunner           31
1,968               Dodge        Superbee             32
1,970               Plymouth     Hemi Superbird       30
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-32. Deriving a new column OD477.0

Notes:

You can also derive a column from existing columns in the table. This is a simple example that calculates the age of each car in the year 2000.

In this example, notice the use of an expression. Also notice that the label of the heading of the derived column is based upon the expression used to derive the column.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — We are introducing the derive capability at this point. Do not spend a lot of time beyond pointing out that you can derive a new column and that it will become an element of your result set. We will have the students derive a simple column in the exercise at the end of this unit.

Transition — Next we cover each of the main components of the SELECT statement, beginning with ORDER BY.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

ORDER BY (1 of 2)

IBM i

```
  SELECT  *
    FROM  cars
    ORDER BY  caryear

CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

   18          2     Ford             Thunderbird         1,957
    3          2     Ford             Mustang             1,965
    5          5     Chevrolet        Corvette            1,965
   12          7     Austin           Healey              1,965
   16          7     Plymouth         GTX                 1,966
    1          1     Oldsmobile       Cutlass 442         1,967
    2          1     Chevrolet        Corvette            1,967
```

- Order rows
- ORDER BY must be last clause in SELECT

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-33. ORDER BY (1 of 2) OD477.0

Notes:

The ORDER BY clause specifies that for the selected columns, rows are to be ordered by one or more columns in ascending or descending order. Ascending order is the default.

In this example, cars are being ordered by model year.

If ORDER BY is not specified in a SELECT statement, then the order in which the rows are displayed cannot be guaranteed. As a matter of fact, the order in which the rows are presented might change from invocation to invocation. The order changes because the Query Optimizer dynamically decides on an access path to retrieve the data.

ORDER BY also supports the coding of an expression in the clause.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Remind the students that SELECT clauses must be coded in a specific order. ORDER BY is always the last clause that is coded.

We do not discuss performance in this class.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

ORDER BY (2 of 2)

IBM i

```
select * from OL37XXCOL/cars
       order by carmake, caryear
OR

select * from OL37XXCOL/cars
        order by 3,5

CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

   13          6     Alpha Romeo      Targa              1,973
   12          7     Austin           Healey             1,965
    5          5     Chevrolet        Corvette           1,965
    2          1     Chevrolet        Corvette           1,967
   11         10     Chevrolet        Chevelle SS454     1,970
    7          7     Chevrolet        El Camino          1,977
    4          3     Chevrolet        Corvette           1,985
```

- May specify more than one column for ORDER BY
- Ordered by second column within first and so on

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-34. ORDER BY (2 of 2) OD477.0

Notes:

The ORDER BY clause specifies an ordering of the rows of the result table. If a single column is identified, the rows are ordered by the values of that column. If more than one column is identified, the rows are ordered by:

1. The values of the first identified column, then by
2. The values of the second identified column,
3. And so on.

In the ORDER BY clause, a column may be specified using its relative column number as defined when you define the result set.

The following statement is the same as the statement in the visual.

```
SELECT * FROM cars ORDER BY 3, 5
```

Generally, you should avoid using this form because it is not as readable as when you use the column name.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

However, when the result set has a calculated column, using the column numbers rather than the name is necessary.

Expressions in ORDER BY

ORDER BY can also support expressions since V4R4 and higher releases. In prior releases, you had to code an ORDER BY like this:

```
select empno, (salary + bonus + comm)  from ol37XXcol/employee
       order by 2
```

Since V4R4 or higher, you can code the same statement like this:

```
select empno, (salary + bonus + comm) from ol37XXcol/employee
       order by (salary + bonus + comm)
```

Column numbers can still be coded in an ORDER BY clause.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The ability to code an expression in an ORDER BY does not add any new function since we could always use an integer to reference a derived column. However, one benefit might be that it is clearer how the result set is to be ordered.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Ascending or descending sequence

IBM i

```
select * from OL37XXCOL/cars
       order by carmake, caryear desc


CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

   13          6     Alpha Romeo      Targa              1,973
   12          7     Austin           Healey             1,965
    4          3     Chevrolet        Corvette           1,985
    7          7     Chevrolet        El Camino          1,977
   11         10     Chevrolet        Chevelle SS454     1,970
    2          1     Chevrolet        Corvette           1,967
    5          5     Chevrolet        Corvette           1,965
   15          1     Dodge            Superbee           1,969
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-35. Ascending or descending sequence OD477.0

Notes:

You can specify ascending or descending sequence for ORDER BY by adding ASC or DESC next to the specific column.

For example:

```
select *
      from cars
      order by carmake desc, caryear
```

would list all columns in ascending sequence of CARYEAR within CARMAKE in descending sequence. Ascending sequence is the default.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *How SELECT works*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-36. Exercise OD477.0

Notes:

Before we continue our lecture about the SELECT statement, let us take the opportunity to learn how the SELECT statement works and what happens when the various clauses are executed.

Perform the exercise *How SELECT works*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — While the students have not yet discovered all of the things that can be done with the SELECT statement, this guided lab exercise teaches them that the clauses must be processed in a specific order. They also see the intermediate results produced by executing specific clauses.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DISTINCT

IBM i

```
SELECT DISTINCT caryear FROM OL37XXCOL/cars

 CARYEAR
  1,967
  1,965
  1,985
  1,984
  1,977
  1,971
  1,970
  1,973
  1,969
  1,966
  1,957
  1,974
```

- Eliminates duplicate rows

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-37. DISTINCT OD477.0

Notes:

The DISTINCT clause eliminates all but one of each set of any duplicate rows of the final result set. In the example, we can see all years of cars that are represented in the CARS table.

The DISTINCT clause is allowed only once for a SELECT statement. SELECT DISTINCT performs an implicit ORDER BY for all the columns that follow it and then suppresses duplicate rows. Columns are presented in ascending order.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This SELECT statement answers the question, “What years of cars are represented in your club?”

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

WHERE column-name clause

IBM i

```
SELECT CARYEAR, CARMAKE, CARMODEL FROM cars
       WHERE caryear = 1967

CARYEAR   CARMAKE          CARMODEL

 1,967    Oldsmobile       Cutlass 442
 1,967    Chevrolet        Corvette
```

- Select rows from a table or view
- Can specify any columns for WHERE
- All Rows Selected if WHERE Clause Not Used

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-38. WHERE column-name clause OD477.0

Notes:

The WHERE clause limits the scope of the SELECT statement based on a certain condition being true.

In our example, **caryear = 1967** is known as a search condition. A search condition must conform to the following rules:

- Each column name in the condition must unambiguously identify a column of the result set or be a correlated reference. A column name is a correlated reference if it identifies a column of a table or view identified in an outer subselect.

  We will discuss correlated references further in the unit titled SubSELECTs.
- A column function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

It follows logically that if no WHERE clause is coded, all rows are selected.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Conditions in WHERE clause

IBM i

```
SELECT CARYEAR, CARMAKE, CARMODEL FROM cars
     WHERE  carmodel = 'corvette'


SELECT CARYEAR, CARMAKE, CARMODEL FROM cars
     WHERE  caryear = 1965
```

- Character Data Enclosed in Single Quotes
- Numeric Data Must NOT be in Quotes

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-39. Conditions in WHERE clause OD477.0

Notes:

Note that character data must be enclosed in quotes and that the value is case sensitive and must match the column value exactly:

For example, corvette *is not* equal to Corvette.

In the first example, no rows will be selected into the result set.

Numeric data is not enclosed in quotes.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Selecting on conditions in the WHERE clause

IBM i

- Compares two values.

```
   =, ¬ =, <>, <, ¬ <, <=, >, ¬ >, >=

One of 2 Conditions .......... OR
Both of 2 Conditions ......... AND

Between 2 Values ............. BETWEEN
Equals Value in Set .......... IN
Similar to Another Value ..... LIKE

NOT BETWEEN,  NOT IN  &  NOT LIKE
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-40. Selecting on conditions in the WHERE clause OD477.0

Notes:

The result of a search condition is derived by the application of the specified logical operators (AND, OR, and NOT) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

A predicate specifies a condition that is true, false, or unknown about a given row or group. The following rules apply to all types of predicates:

- All values specified in a predicate must be compatible.
- The value of a host variable must not be a string longer than 32,766 bytes.

A search condition specifies a condition that is true, false, or unknown about a given row or group.

This visual shows some of the operators that can be used when you code conditions.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Tip

The hex value for the ¬ symbol varies by keyboard. For this reason, the ¬ symbol is not recommended. The <>, <=, and >= symbols are preferred when coding conditions.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — In this visual, we introduce the WHERE statement. Tell the students that WHERE is used to limit the rows selected from a table. WHERE contains a condition that can be made up of one or more expressions that make use of logical, arithmetic, string, and Boolean operators. In addition, you can use certain SQL functions that we discuss in the following unit.

Transition — BETWEEN, IN, and LIKE are predicates that we discuss with examples in this topic. First, we show you some simple operations that can be performed. Then, we add some complexity to the examples and show you a few WHERE expressions that include numeric and Boolean operators.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

WHERE: Other operators

IBM i

```
select * from cars where (substring(carmake,1,4) = 'Chev')

select * from cars where (caryear+10) < 2000
```

Arithmetic:

- + (Add)
- - (Subtract)
- / (Divide)
- * (Multiply)
- ** (Exponentiate)

String:

- || (Concatenate)

Logical:

- AND
- OR
- NOT

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-41. WHERE: Other operators OD477.0

Notes:

Many other operators can be used in an expression in a WHERE clause. We will not cover all of them with examples, but WHERE search conditions can include arithmetic, string, and logical operators in addition to the relational operators that we covered in the previous visual.

The concatenation operator (||) can also be written as CONCAT.

Functions, such as substring, are also available. We cover functions in detail in the next unit.

References:

- *DB2 for i SQL Reference*, Language Elements

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Less than, greater than, equal

IBM i

```
SELECT  *
  FROM  OL37XXCOL/cars
  WHERE  caryear >= 1970
  ORDER BY  caryear

CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

   11         10     Chevrolet        Chevelle SS454     1,970
   17          3     Plymouth         Hemi Superbird     1,970
   10          1     Maserati         Diablo             1,971
   13          6     Alfa Romeo       Targa              1,973
   19         10     Mercedes         450SL              1,974
    7          7     Chevrolet        El Camino          1,977
    6          6     Ferrari          308                1,984
    4          3     Chevrolet        Corvette           1,985
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-42. Less than, greater than, equal OD477.0

Notes:

In this example, we use the greater than or equal operator to select only those cars of model year 1970 and later.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

AND, OR

IBM i

```
SELECT  *
          FROM  cars
          WHERE  caryear >= 1960 and carmake='Chevrolet'
          ORDER BY  caryear

CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR
   5         5      Chevrolet        Corvette          1,965
   2         1      Chevrolet        Corvette          1,967
  11        10      Chevrolet        Chevelle SS454    1,970
   7         7      Chevrolet        El Camino         1,977
   4         3      Chevrolet        Corvette          1,985
```

- Can code multiple conditions
- Character values are case sensitive

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-43. AND, OR OD477.0

Notes:

Logical AND and OR can be used to specify multiple conditions that are to be tested for a true or false value. In this visual, we are looking for all Chevrolet cars built in 1970 or later.

Question

Write the statement that answers the question, “What Chevrolet and Ford cars in the car club have a model year after 1980?”

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Keep the students involved by asking them to code the SQL query to answer the question in their notes:

Correct Answer

```
select * from cars where (carmake = 'Chevrolet' or carmake = 'Ford')
       and (caryear > 1980)
```

```
CARNUM  MEMBERNUM  CARMAKE          CARMODEL         CARYEAR
    4         3     Chevrolet        Corvette          1,985
```

Transition — Some students may suggest this answer:

```
select * from cars where carmake = 'Chevrolet' or carmake = 'Ford'
       and caryear > 1980
```

This produces an incorrect result:

```
CARNUM  MEMBERNUM  CARMAKE          CARMODEL         CARYEAR
    2         1     Chevrolet        Corvette          1,967
    4         3     Chevrolet        Corvette          1,985
    5         5     Chevrolet        Corvette          1,965
    7         7     Chevrolet        El Camino         1,977
   11        10     Chevrolet        Chevelle SS454    1,970
```

Explain why it is incorrect. This SELECT includes all years of Chevrolet, plus only those Fords built after the 1980 model year. That is not what we wanted to know. This discussion should lead directly into the use of parentheses in the next visual.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Parentheses with multiple conditions

IBM i

```
SELECT  *
        FROM  OL37XXCOL/cars
        WHERE  (caryear >= 1960) and (carmodel =
                                          'Corvette')
        ORDER BY  caryear

CARNUM   MEMBERNUM   CARMAKE        CARMODEL        CARYEAR

   5         5       Chevrolet      Corvette        1,965
   2         1       Chevrolet      Corvette        1,967
   4         3       Chevrolet      Corvette        1,985
```

- Good coding technique
- Helps readability / maintainability

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-44. Parentheses with multiple conditions OD477.0

Notes:

Expressions within parentheses are evaluated first. When the order of evaluation is not specified by parentheses, exponentiation is applied after prefix operators (such as -) and before multiplication and division. Multiplication and division are applied before addition and subtraction. Operators at the same precedence level are applied from left to right. The following table shows the priority of all operators.

Table 1: Priority of Operators

Priority	Operators	Notes
1	+, -	when used for signed numeric values
2	**	
3	*, /, ||	
4	+, -	when used between two operands

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using parentheses when coding SQL will make your code more readable. It is also easier to code because you force yourself to break up complex expressions into their basic components.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — Emphasize the value of using parentheses. RPG IV programmers who have coded free form expressions can confirm the usefulness of parentheses.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Other WHERE clause considerations

IBM i

Incorrect:

```
SELECT  *
  FROM  ol37XXcol/cars
  WHERE  (caryear >= 1960 and <= 1990)
  ORDER BY  caryear
```

Correct:

```
SELECT  *
  FROM  ol37XXcol/cars
  WHERE  (caryear >= 1960) and (caryear <= 1990)
  ORDER BY  caryear
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-45. Other WHERE clause considerations OD477.0

Notes:

When coding conditions, make sure that they always are complete predicates such that there is a column name or a value on each side of the operator. Use parentheses to define each condition precisely.

The first example in the visual will result in an invalid token error when the statement is being prepared for execution.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

WHERE NOT (1 of 2)

IBM i

```
SELECT  *
         FROM  OL37XXCOL/cars
         WHERE  NOT carmake='Ferrari'

CARNUM   MEMBERNUM   CARMAKE          CARMODEL         CARYEAR

   1         1      Oldsmobile       Cutlass 442       1,967
   2         1      Chevrolet        Corvette          1,967
   3         2      Ford             Mustang           1,965
   4         3      Chevrolet        Corvette          1,985
   5         5      Chevrolet        Corvette          1,965
   7         7      Chevrolet        El Camino         1,977
```

- Select all rows except those where the car is made by Ferrari.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-46. WHERE NOT (1 of 2) OD477.0

Notes:

Combining NOT with the WHERE clause excludes the selection of rows from the result set based on the condition defined against the column. In this example, we are not interested in Ferraris.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

WHERE NOT (2 of 2)

IBM i

Correct:

WHERE NOT (caryear = 1967)

is the same as

WHERE (caryear ¬= 1967)

Incorrect:

WHERE (caryear NOT = 1967)

WHERE ¬(caryear = 1967)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-47. WHERE NOT (2 of 2) OD477.0

Notes:

NOT does not necessarily equate to ¬. Remember earlier that we discouraged the use of the ¬ because it may be represented differently on different keyboards.

Using the **NOT** in the first example is a better way to code the condition to be tested.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

BETWEEN x AND y

IBM i

```
SELECT  *
          FROM  OL37XXCOL/cars
          WHERE CARYEAR BETWEEN 1960 AND 1970

   CARNUM   MEMBERNUM   CARMAKE          CARMODEL        CARYEAR

      1         1       Oldsmobile       Cutlass 442      1,967
      2         1       Chevrolet        Corvette         1,967
      3         2       Ford             Mustang          1,965
      5         5       Chevrolet        Corvette         1,965
```

- Values within a range
- Limits are inclusive
- Cannot code BETWEEN 1970 AND 1960
- Same as (CARYEAR >=1960 and CARYEAR <=1970)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-48. BETWEEN x AND y OD477.0

Notes:

The BETWEEN predicate compares a value with a range of values.

The BETWEEN predicate:

```
value1 BETWEEN value2 AND value3
```

is logically equivalent to the search condition:

```
value1 >= value2 AND value1 <= value3
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IN (x, y, z)

IBM i

```
SELECT  *
          FROM  OL37XXCOL/cars
          WHERE CARYEAR IN (1965, 1967, 1977)


   CARNUM   MEMBERNUM   CARMAKE          CARMODEL       CARYEAR

      1         1      Oldsmobile       Cutlass 442     1,967
      2         1      Chevrolet        Corvette        1,967
      3         2      Ford             Mustang         1,965
      5         5      Chevrolet        Corvette        1,965
      7         7      Chevrolet        El Camino       1,977
     12         7      Austin           Healey          1,965
```

- Any match of a value in the list
- Order of items in list NOT important

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-49. IN (x, y, z) OD477.0

Notes:

The IN predicate compares a value with a set of values. In our example, we are checking to see which cars were built in model years 1965 or 1967 or 1977.

If the values we are checking against are character, remember that the case must exactly match the case of the value in the row in order to be selected.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — Other predicates that you will see used in SQL are ALL, SOME, and ANY. These predicates are used with subselects which we will discuss in a later unit.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

LIKE (1 of 3)

IBM i

- Values are similar to character string
- Uses wildcard characters
  - Underscore (_) represents any single character.
  - Percent (%) represents a string of zeros or more characters.
  - Any other character represents itself.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-50. LIKE (1 of 3) OD477.0

Notes:

The LIKE predicate searches for strings that have a certain pattern. The pattern is specified by a string in which the underscore and percent sign have special meanings.

The LIKE predicate may also be written NOT LIKE to exclude selection from the result set.

Since LIKE works with strings, you need to be careful of case-sensitivity.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

LIKE (2 of 3)

IBM i

```
      select * from OL37XXCOL/cars
        where carmake like '%hev%'

CARNUM   MEMBERNUM   CARMAKE          CARMODEL          CARYEAR

    2          1     Chevrolet        Corvette           1,967
    4          3     Chevrolet        Corvette           1,985
    5          5     Chevrolet        Corvette           1,965
    7          7     Chevrolet        El Camino          1,977
   11         10     Chevrolet        Chevelle SS454     1,970
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-51. LIKE (2 of 3) OD477.0

Notes:

If the values we are checking against are character, remember that the case must exactly match the case of the value in the row in order to be selected.

In the example above, if we coded **like %Hev**, we would get incorrect results. String comparisons are case sensitive.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Ask the class what this SELECT statement says.

The statement will select all rows where the column CARMAKE includes the string “hev” anywhere in it.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

LIKE (3 of 3)

IBM i

```
select * from CARCLUB/cars
  where carmake not like '__r%'
```

CARNUM	MEMBERNUM	CARMAKE	CARMODEL	CARYEAR
1	1	Oldsmobile	Cutlass 442	1,967
2	1	Chevrolet	Corvette	1,967
4	3	Chevrolet	Corvette	1,985
5	5	Chevrolet	Corvette	1,965
7	7	Chevrolet	El Camino	1,977
10	1	Maserati	Diablo	1,971
11	10	Chevrolet	Chevelle SS454	1,970
12	7	Austin	Healey	1,965
13	6	Alpha Romeo	Targa	1,973

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-52. LIKE (3 of 3) OD477.0

Notes:

In this example, we want to exclude from our result set any car that has an “r” in the third character of the *carmake* column. To specify this, we use *not like __r%* where the first two characters are underscores (_). The underscore means that we will accept any character in the first or second position. We are looking for an “r” in the third position and we do not care about the rest of the characters in this column so we use “%” to handle the rest of the string.

The ESCAPE clause of the LIKE predicate allows the definition of patterns you want to match to values that contain the percent or underscore characters or both.

Further function is available in the LIKE predicate. Expressions can be coded in a LIKE on the LIKE itself as well as the Escape. We will cover an example of this function later in the class.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Ask the class what this SELECT statement says.

The statement will select all rows where the column CARMAKE does not include the string “r” in the third position.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

GROUP BY

IBM i

```
select carmake, count(carmake)
       from cars
       group by carmake

CARMAKE          COUNT ( CARMAKE )
Alfa Romeo                       1
Austin                           1
Chevrolet                        5
Dodge                            1
Ferrari                          1
Ford                             2
Maserati                         1
Mercedes                         1
Oldsmobile                       1
Plymouth                         3
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-53. GROUP BY OD477.0

Notes:

A GROUP BY clause first orders all rows based on the GROUP BY expression. Then, for each row, a single row is produced. A GROUP BY orders rows like an ORDER BY but a GROUP BY results in a single summary row whereas an ORDER BY results in a detailed set of individual rows.

In its simplest form, a GROUP BY clause contains a grouping expression. A grouping expression is an expression used in defining the grouping of the desired result set. Each column name included in a grouping expression must unambiguously identify a column of the result set.

In this example, we are grouping our result set by the make of car. Regardless of how many cars we have of the same make in the table CARS, we will see only one row in the result set for each unique CARMAKE.

The result of the GROUP BY clause is a set of groups of rows. In each group of more than one row, all values of each grouping expression are equal, and all rows with the same set of values of the grouping expressions are in the same group. For grouping, all null values for a grouping expression are considered equal.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

HAVING

IBM i

```
select membernum, count(carmake) as num_cars
       from cars
       group by membernum
       having count(carmake) > 1
       order by membernum

MEMBERNUM        NUM_CARS
     1                  3
     2                  2
     3                  2
     6                  2
     7                  3
    10                  2
```

- HAVING produces intermediate table from table/view selected

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-54. HAVING OD477.0

Notes:

The clause HAVING defines a search condition on the column functions specified in the SELECT, for a GROUP of rows selected with the GROUP BY clause. The HAVING clause causes an intermediate result table to be built that consists of those groups for which the search condition is true.

The example shown extracts, from the CARS table that lists those members who own more than one old car in the car club.

The resulting table will be sequenced according to the ORDER BY clause, based on the value of the first column, MEMBERNUM, in ascending order.

TIP

Where possible use the WHERE clause instead of the HAVING clause. The WHERE does the selection of rows prior to the GROUP BY rather than after it. This minimizes the number of rows processed for the selection criteria. However, you can only code simple conditions

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

in a WHERE. You must use HAVING in situations where a WHERE cannot be used. This will become even more significant in a later unit.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — Use this opportunity to review the order in which the clauses in a SELECT must be coded.

Note that we could use a JOIN to the MEMBERS table to add LASTNAME and FIRSTNAME to the result set. You could ask the students if they have any ideas how to add these columns to the result set. I did not use a JOIN because we will not explain it until a later unit.

Transition — COUNT is a function that counts all those situations where there is more than one car. It cannot be used in a WHERE clause. We will see more about functions in the next unit.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

FETCH FIRST n ROWS ONLY

IBM i

```
select empno, rtrim(firstnme)||' '||rtrim(lastname)
       as Employee_Name, salary+bonus+comm as Total_Pay
       from employee
       order by 3 desc
       fetch first 10 rows only

EMPNO   EMPLOYEE_NAME                    TOTAL_PAY
000010  Christine Haas                   57,970.00
000110  Vicenzo Lucchesi                 51,120.00
000020  Michael Thompson                 45,350.00
000050  John Geyer                       44,189.00
000030  Sally Kwan                       42,110.00
000070  Eva Pulaski                      39,763.00
000400  Henry Wilson                     38,000.00
000060  Irving Stern                     35,330.00
000220  Jennifer Lutz                    32,827.00
000090  Eileen Henderson                 32,740.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-55. FETCH FIRST n ROWS ONLY OD477.0

Notes:

You can specify that you want to return or display a certain number of rows of the result set. The new clause is **FETCH FIRST n ROWS ONLY**.

In this example, we want to know who are the top ten money earners We can also use this feature to determine the top money earner by coding:

```
select empno, rtrim(firstnme)||' '||rtrim(lastname)
   as Employee_Name, salary+bonus+comm as Total_Pay
   from employee
   order by 3 desc
   fetch first row only
```

in the company.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

3.4. Using System i Navigator

Instructor topic introduction

What students will do — In this topic, we will demonstrate again how to use System i Navigator to perform SQL queries.

We will use the SQL Scripts feature as well as show the class how System i Navigator can be used to display the contents of an SQL table (or any data file).

System i Navigator is the preferred tool. Some of the figures revisit what we briefly did in Lab exercise 2,How SELECT Works, but, move through them as quickly as you want.

The students should be given the option of performing the following exercises using System i Navigator as well as the interactive STRSQL interface.

The purpose of this topic is not to teach everything there is to know about System i Navigator. Only discuss it briefly so that the students can see how it flows and where various functions that relate to running SQL scripts are found.

Rather than using the electronic presentation, you may perform a live demo of the same steps; the tables that are used can be found in the student master schema, OL37XXCOL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

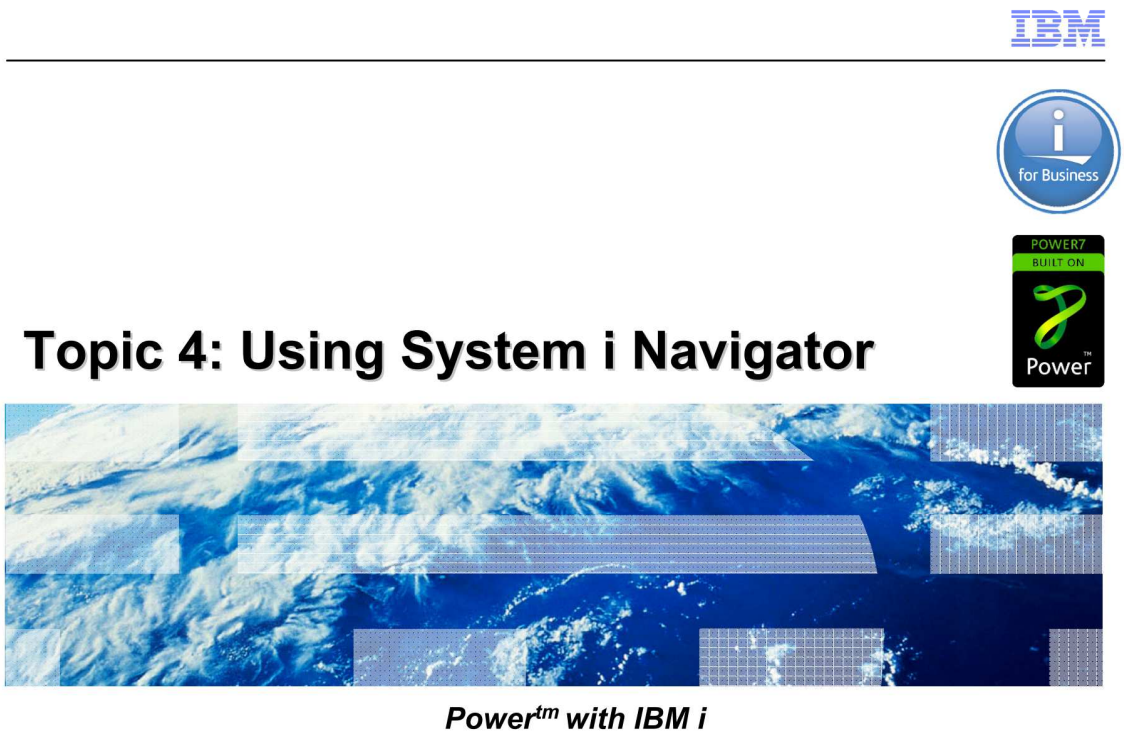


Figure 3-56. Topic 4: Using System i Navigator OD477.0

Notes:

System i Navigator is a powerful tool that provides the user a graphical user interface (GUI) to SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

System i Navigator overview

IBM i

- Uses your Windows skills
  - Uses familiar tools such as Explorer view, properties sheets, context menus, drag and drop
  - Reduces the need to learn a command interface
  - Uses easier to understand terminology and concepts
  - Uses graphics; not just icons
- Wizards
  - High degree of function integration, not standalone utilities
- Provides support for:
  - User
  - System operator
  - System administrator

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-57. System i Navigator overview OD477.0

Notes:

We are going to take a brief tour of System i Navigator. While this product can be used for many purposes, we are only interested in using it to explore schemas, and the SQL objects contained in schemas.

When you perform the machine exercise at the end of this unit, we recommend that you try using the **SQL script** capability of System i Navigator.

Start System i Navigator by clicking the icon that is either on the desktop or in the System i Access folder.

A number of System i connections are displayed to which you have access.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

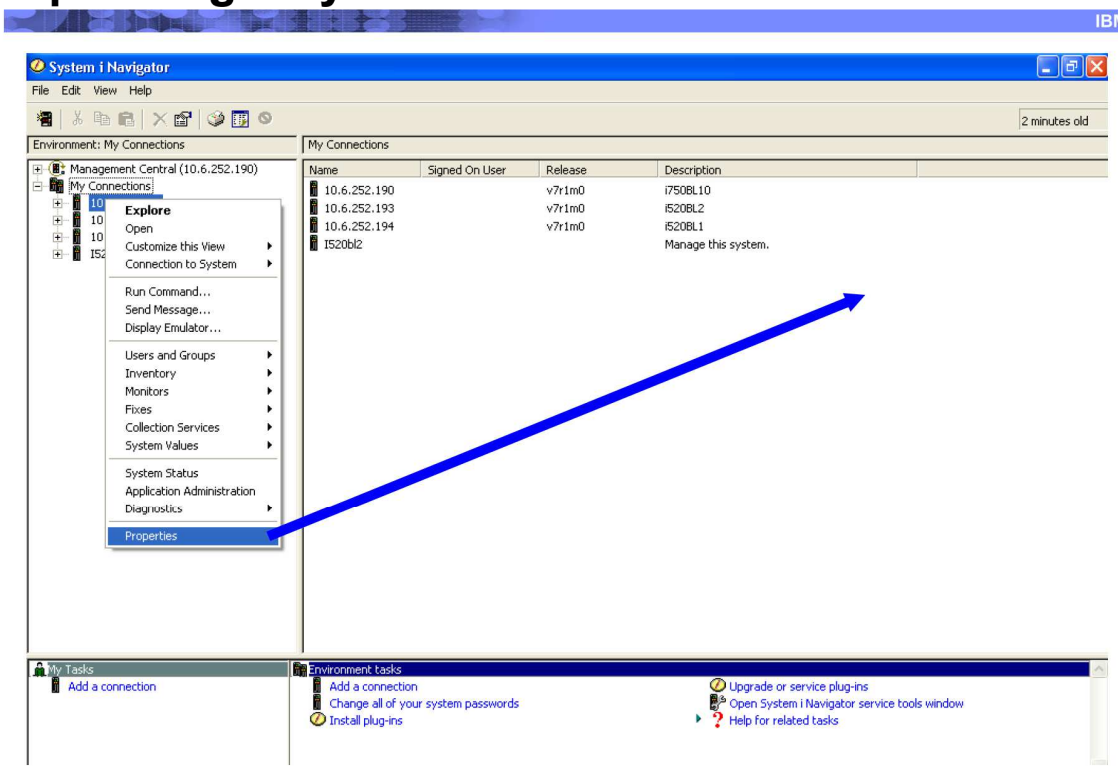
Details — If necessary, discuss adding a System i connection:

1. Right-click **My Connections**.
2. Select **Add Connection**.
3. Enter the host name (must be in the HOSTS table) or the IP address of the class System i.
4. In the next window, click **Prompt Every Time**.
5. In the next window, click **Verify Connection**. If this works fine, you are connected. If not, either your HOST name is not in the HOSTS table or the IP address you entered is invalid.
6. When the connection is successful, click **Finish**.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Open target system



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-58. Open target system OD477.0

Notes:

To access the resources of the system you wish to work with, you should expand it by clicking the plus (+) sign or use right-click and select **Open**.

As you can see, there are a number of other options that we will not cover in this class.

One of the options we should explain is the Connections Properties notebook. To open the notebook, select **Properties**. The next visual shows you the notebook.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

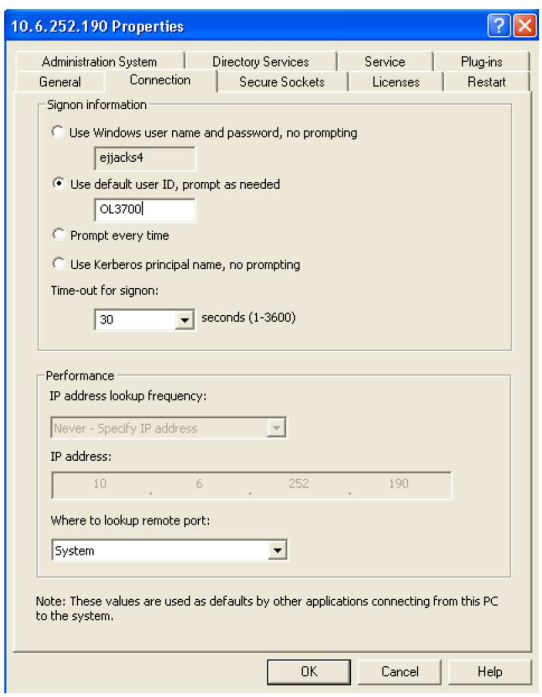
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Properties notebook for System i connection

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-59. Properties notebook for System i connection OD477.0

Notes:

Click the **Connection** tab. You can now modify sign-on information. Normally, you would enter your System i user profile name. When you sign-on, System i Navigator will pull any information about your library list that is available during the signon process.

Sign-on information: Specifies the settings the System i Navigator uses when it connects to the server. You can specify a user ID and password to be used when you connect to the server. In the Time-out for signon field, you can specify the number of seconds that System i Navigator will continue trying to connect to the server.

Possible values for user ID and password are:

Use Windows user name and password, no prompting: Specifies to use your Windows log on user ID and password for all connections to the server. You will never be prompted. Your Windows log on user ID and password must follow i5/OS conventions in order for this to work. For example, the maximum length of an i5/OS user ID is 10 characters.

Use default User ID, prompt as needed: Specifies that you must enter a user ID. A blank is not allowed. You will be prompted for the password only if it is not cached.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If you use the default user ID and password caching, the first time you connect to the server you will be prompted; subsequent connections will not prompt. This user ID and password will remain active until you reboot Windows.

Prompt every time: Specifies that you must enter the user ID and password on every connection. Select this if you want to be prompted for every session.

Use Kerberos principal name, no prompting: Specifies to use your Kerberos principal name instead of a user name and password for all connections to the server. You will never be prompted. This option is available only when Kerberos is running on the server. Kerberos is a third-party authentication method that uses a centralized, secure server that authenticates users to resources in a particular network. A Kerberos principal is a user within a Kerberos-enabled network.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

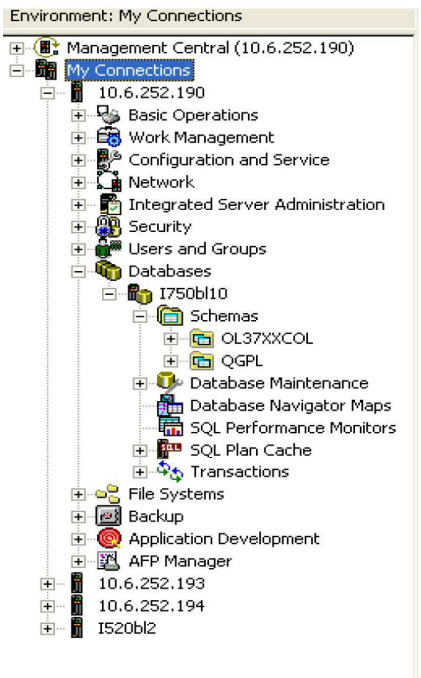
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Connection supports database

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-60. Connection supports database OD477.0

Notes:

When you have signed on to a system for the first time, System i Navigator will scan that system for functions it supports (varies by release of OS/400 or IBM i) and for the functions to which you have access depending upon your level of authority. System i Navigator can support servers with plus or minus two release level of software than what is installed on the client PC.

Notice in the visual that the libraries to which we have access are shown. This information is displayed when you expand the **Databases** plus (**+**) sign followed by the **Schemas** plus (**+**) sign.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

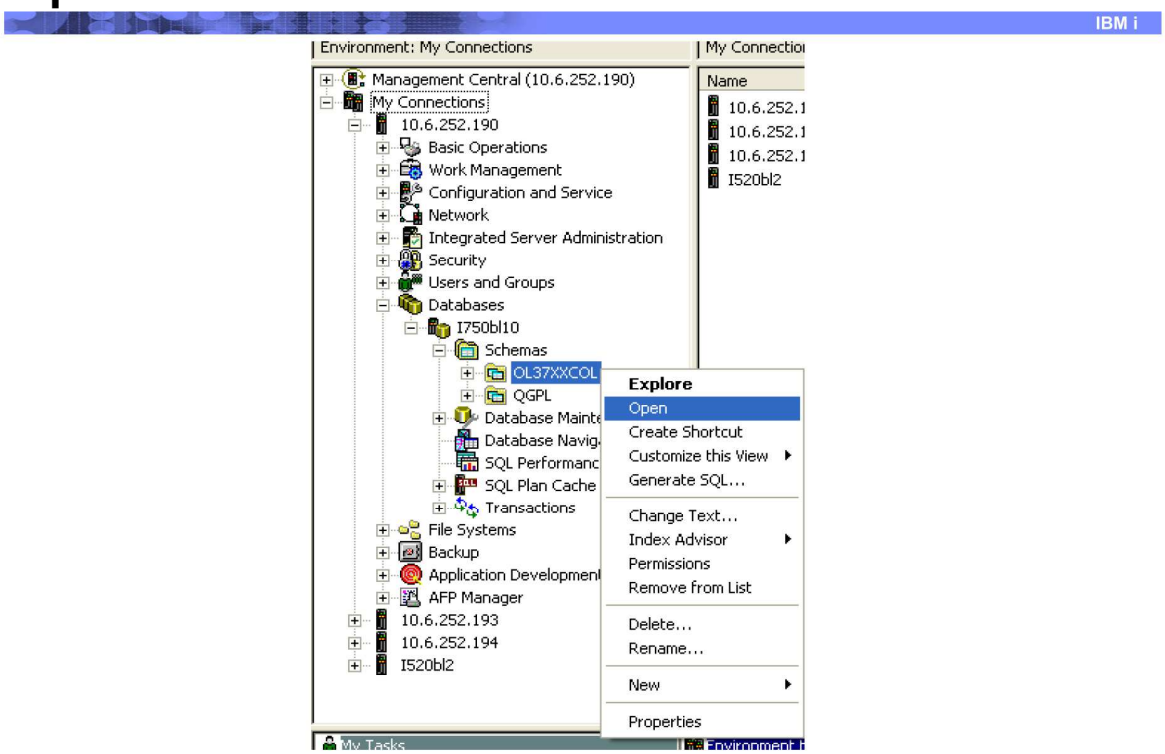
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Open schema



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-61. Open schema OD477.0

Notes:

You can display the objects in a schema by double-clicking the library name or by right-clicking the library name and selecting **Open**. You can then select which objects you'd like to see such as **All Objects**, **Aliases**, **Constraints**, etcetera. The easiest selection to make is **All Objects**. Let's take a look at the schema contents in the next figure.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

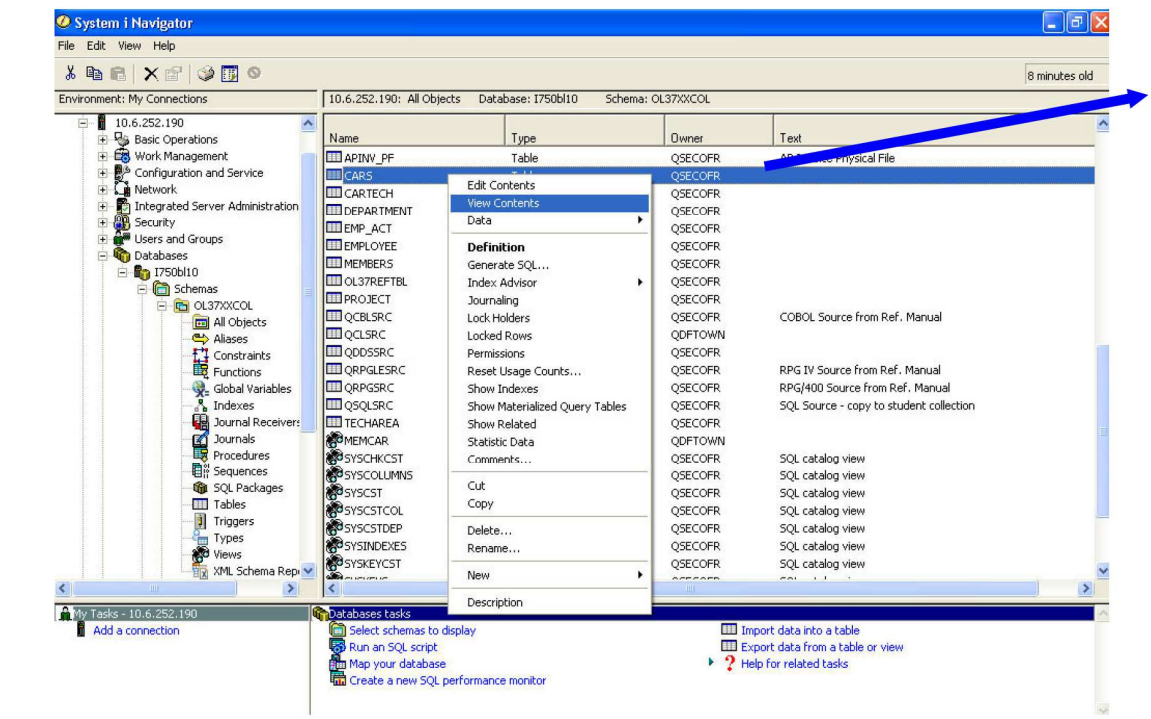
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Display schema objects



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-62. Display schema objects OD477.0

Notes:

Under Schemas, either expand the OL37XXCOL plus (**+**) sign or double-click the **OL37XXCOL** schema icon. In the right panel, you can see the different categories by which the different objects have been grouped: Aliases, Constraints, Distinct Types, Functions, Indexes, Journal Receivers, Journals, Procedures, Sequences, SQL Packages, Tables, Triggers, and Views.

To see all the database objects of the schema, double-click the **All Objects** icon. Note in the right panel, you can see the catalog views, a journal, and a journal receiver that were automatically created in your schema.

You can double-click each of the icons to see aliases, tables, sequences, indexes, packages, procedures, triggers, journal, a journal receiver, and some catalog views.

Right-click and select **View Contents** for the **CARS** table

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

View table contents

IBM i

Contents of OL37XXCOL.CARS - 10.6.252.190(I750bl10)

	CARNUM	MEMBERNUM	CARMAKE	CARMODEL	CARYEAR
1	1	1	Oldsmobile	Cutlass 442	1967
2	2	1	Chevrolet	Corvette	1967
3	3	2	Ford	Mustang	1965
4	4	3	Chevrolet	Corvette	1985
5	5	5	Chevrolet	Corvette	1965
6	6	6	Ferrari	308	1984
7	7	7	Chevrolet	El Camino	1977
8	10	1	Maserati	Diablo	1971
9	11	10	Chevrolet	Chevelle SS454	1970
10	12	7	Austin	Healey	1965
11	13	6	Alfa Romeo	Targa	1973
12	14	8	Plymouth	Roadrunner	1969
13	15	11	Dodge	Superbee	1969
14	17	3	Plymouth	Hemi Superbird	1970
15	16	7	Plymouth	GTX	1966
16	18	2	Ford	Thunderbird	1957
17	19	10	Mercedes	450SL	1974

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-63. View table contents OD477.0

Notes:

We right-clicked and selected View Contents for the **CARS** table. All rows and columns are displayed in tabular format. Notice that the results are exactly the same as if we had entered **SELECT * FROM CARS**.

There is a difference between what you can do with View Contents versus Edit Contents. View Contents opens the table and displays it as a SELECT will do. Edit Contents also enables you to insert, update, or delete rows from the table using a window if you have appropriate authority.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

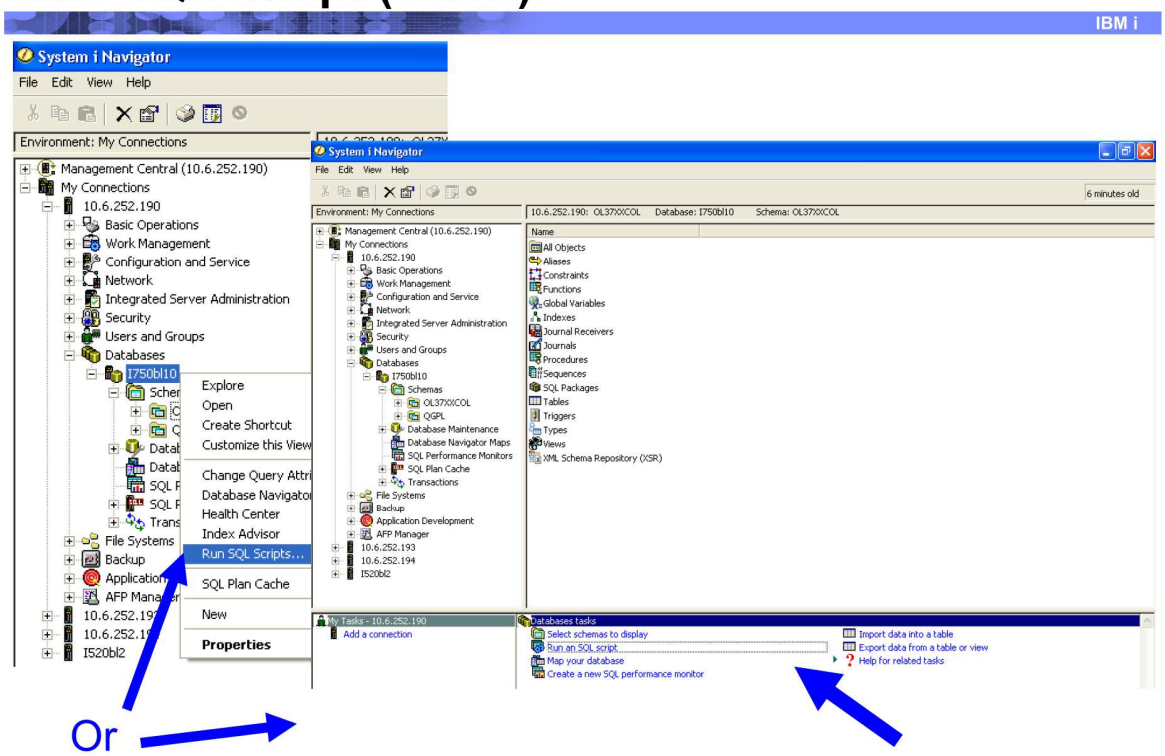
Details — We use this window to “display the table.” The main purpose we want to emphasize is that you can display the content of a table by right-clicking it and selecting view contents.

You can also display a table by right-clicking it and selecting **Edit Contents**.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Run SQL Script (1 of 2)



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-64. Run SQL Script (1 of 2) OD477.0

Notes:

The really significant feature of System i Navigator that we want you to use is **Run SQL Scripts**.

We click the database icon and select **Run SQL Scripts**, or from the lower right portion of the window we can click **Run an SQL Script**. Note that the contents of this part of the window will vary depending on what you click. Clicking **Databases** or one of its subcategories will display the **Database tasks** options.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

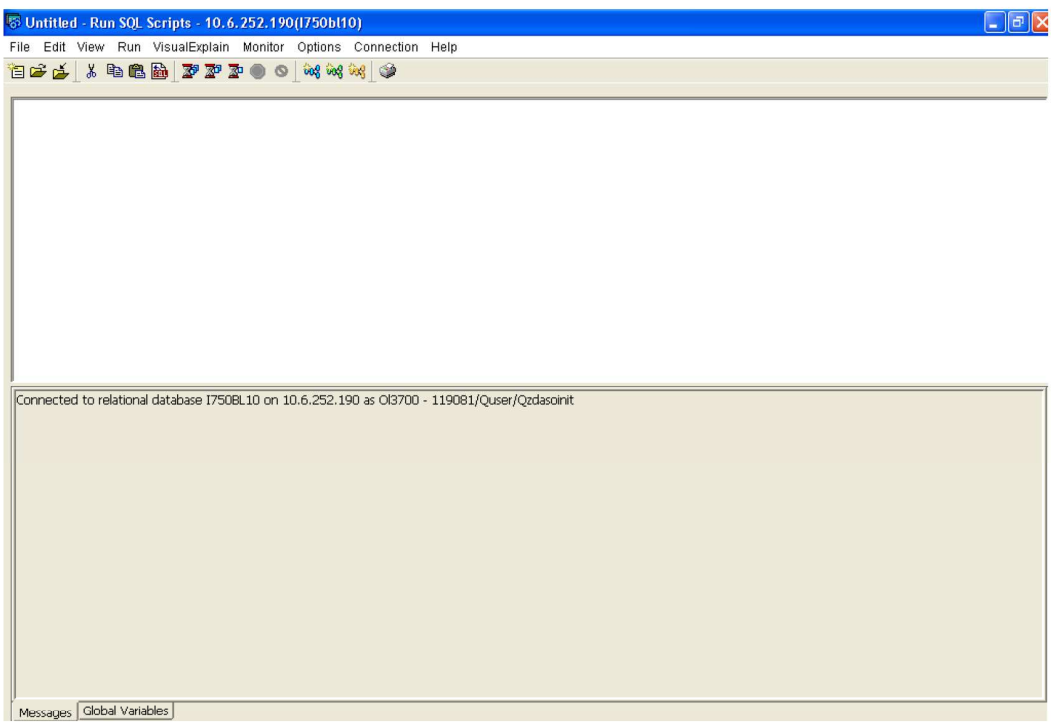
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Run SQL Script (2 of 2)

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-65. Run SQL Script (2 of 2) OD477.0

Notes:

When we select **Run SQL Scripts** or **Run an SQL Script**, a window pops up like the one in the visual. We can enter SQL statements one or many at a time. In other words, we can use this facility as a prototyping tool, like the 5250 STRSQL interface, or we could run a series of statements in the same fashion as the `RUNSQLSTM` command. We will discuss both in the next topic.

System i Navigator provides you with these facilities and much, much more than what is available in 5250 emulation.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

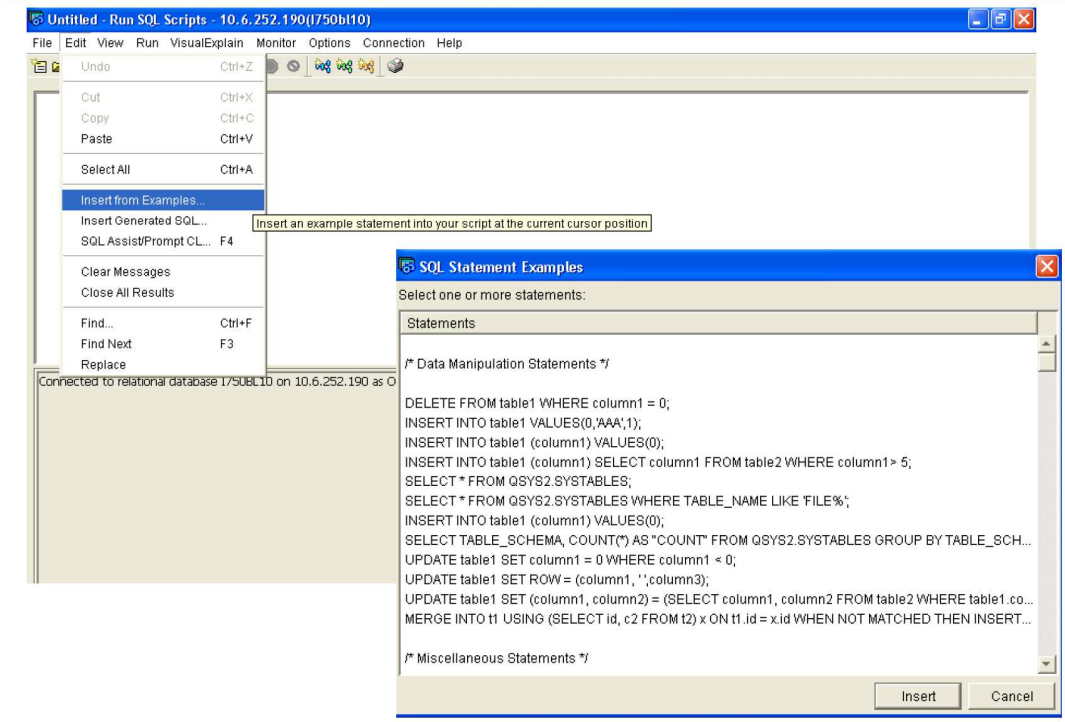
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL statement examples



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-66. SQL statement examples OD477.0

Notes:

If you click the **Edit** > **Insert from Examples** menu bar option, we are offered sample templates for almost every SQL statement. These can be selected, inserted and then modified to construct the statement desired.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

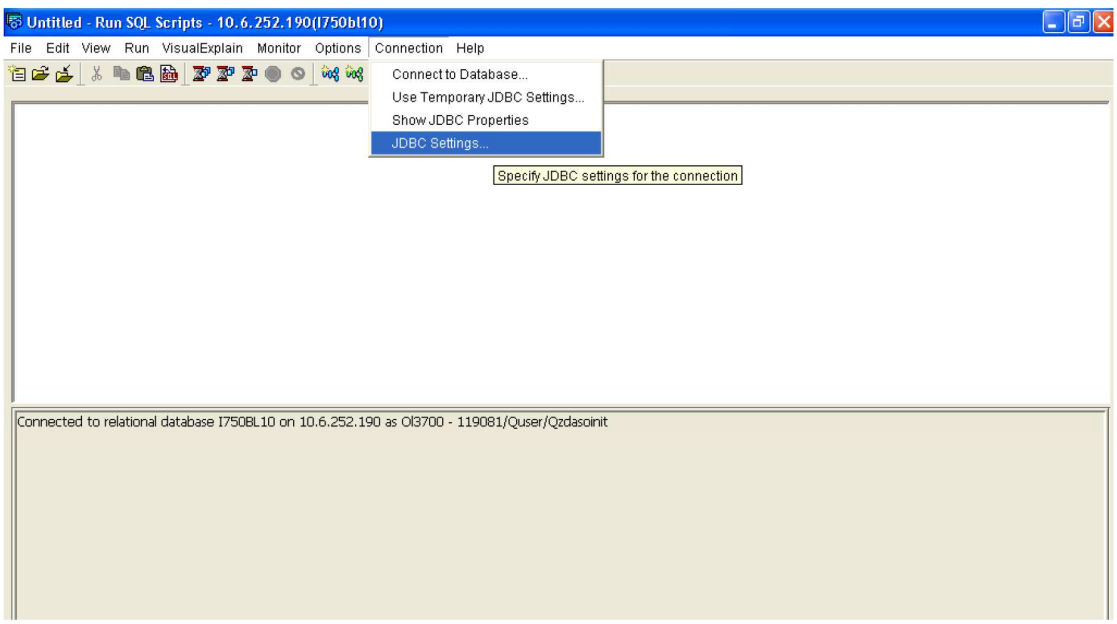
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Open JDBC properties notebook

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-67. Open JDBC properties notebook OD477.0

Notes:

Before we code any statements, we need to set some options for our session. There are several things that you should know:

1. IBM i Access uses a JDBC connection to the database.
2. You can access and change connection attributes and parameters by clicking **Connection** in the menu bar and selecting **JDBC Settings.**

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

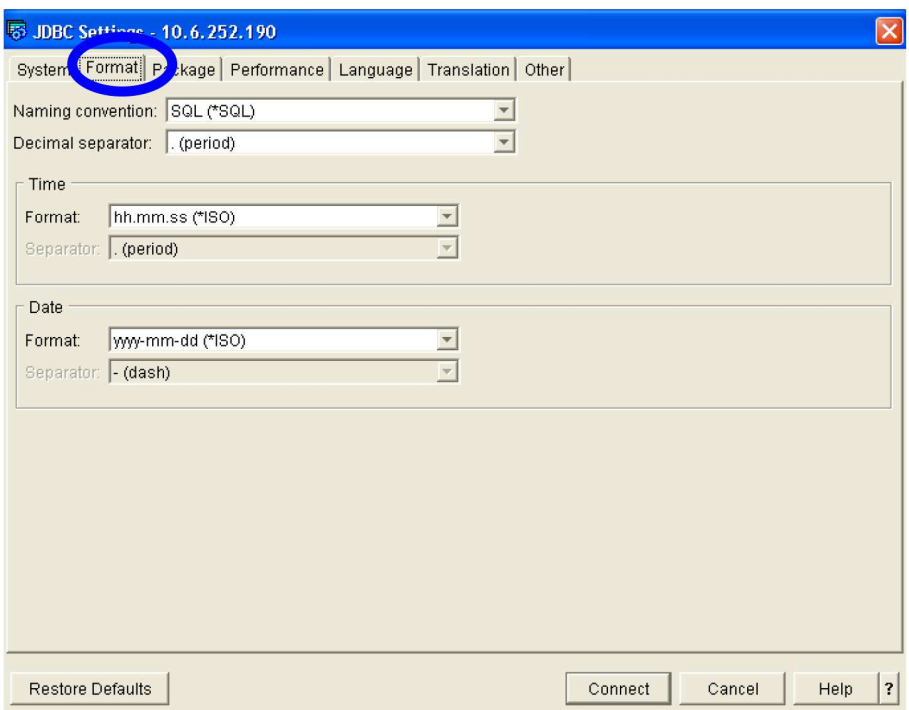
Details — The visual was captured on a V7R1 release so the connection is JDBC. V4 was the last version to use ODBC.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Set naming convention

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-68. Set naming convention OD477.0

Notes:

Here we select the **Format** tab and specify the default of **SQL Naming Convention (*SQL)** instead of **System Naming Convention (*SYS)**. SQL convention uses a period (**.**) between the schema and the object names. SYS convention uses a forward slash (/) between the schema and object names. Be sure to click **Connect** if any changes are made to this display.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

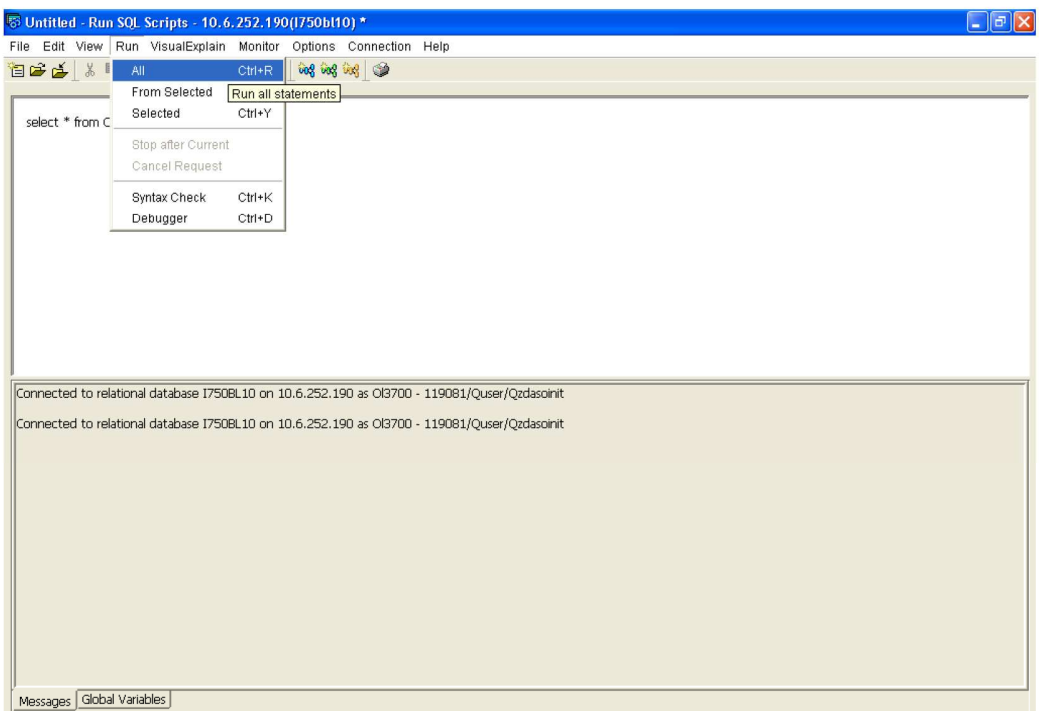
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Enter SQL statement and run

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-69. Enter SQL statement and run OD477.0

Notes:

We enter an SQL statement **select * from OL37XXCOL.cars;** and the results are displayed in the Output pane.

This is only a sample of an SQL statement. We can enter and run all the forms of the Select statement that we have discussed thus far.

The steps to follow are:

1. Enter the SQL statement.
2. Follow each statement that you enter with a semicolon (**;**). This terminates individual statements and, since you can code and run a series of statements, you must code the semicolon; it is also necessary for the last statement.
3. There are several ways to run the statements but the easiest is to click **Run** on the menu bar and select **All**.

   If the statement is successful, messages tell you so. In the case of a Select, the result set is displayed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If a statement is unsuccessful, an error message is displayed and the statement is highlighted. You must correct the statement in error and then select it and others that follow it to process and run them.

SQL script summary of steps

To create an SQL script, do the following:

1. In the System i Navigator window, expand the system you want to use.
2. Click Databases and select **Run an SQL Script** from the lower right part of the window under Database tasks. The **Run SQL Scripts** window is displayed.
3. If you have existing SQL statements that you want to use, you can copy those statements and paste them into the Run SQL Scripts window.
4. If you do not have existing SQL statements, type your statements into the window or insert examples from the SQL statement examples list and modify these template in your window.
5. After you have finished, you can syntax check your SQL script by selecting **Syntax check** from the **Run** menu. If any errors occur, descriptions of the errors appear in the Run History pane.
6. When you are done syntax checking your SQL script, select **Save from the File** menu. You are prompted for the location and name of the script.

Additional tips for using SQL scripts

The following are additional tips for working with the Run SQL Scripts window:

- You can use the SQL scripts function without starting System i Navigator. Once you have saved a script file, you can double-click the script file to edit it without starting System i Navigator.
- Use a semicolon (;) to separate statements.
- Use two dashes (--) to make the rest of the line a comment.
- To make longer comments, start the comment with /* and use */ to end it. Comments made in this manner can be any length.
- Double-click a statement to select it, or place the cursor in a statement and the statement is automatically selected when run.
- Files can be up to 1 MB in size.
- CL commands can be executed by placing CL: At the beginning of a statement. You can use any CL command that can be submitted from a batch job.
- Review the Help Topics for additional information.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

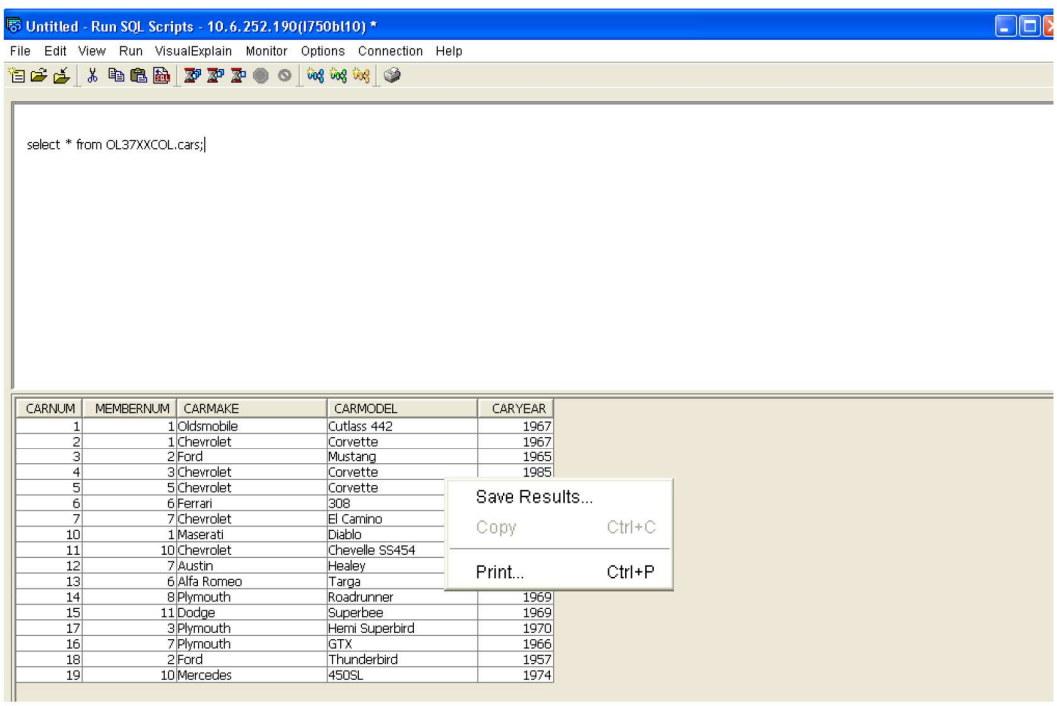
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Results viewer

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-70. Results viewer OD477.0

Notes:

We enter an SQL statement **select * from OL37XXCOL.cars;** and the results are displayed in the Output pane.

This is the same table that was displayed when we right-clicked and viewed contents of the CARS table earlier in this topic.

System i Navigator allows for saving or printing of your result set. If the **Options** > **Allow Save Results** menu bar option is chosen prior to running your query, you can save the results of the query by right-clicking the displayed result set and selecting **Save Results** > **Printing** is also accessed in the same fashion.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — There is a feature of Run SQL Scripts that you should demonstrate at this point. It is the SQL Builder. Follow these steps:

1. Using your existing Run SQL Scripts window, click **Edit > SQL Assist/Prompt CL...**.
2. The SQL Assist Window pops up. Point out the various statements that can be built by SQL Assist. We will use SELECT which is the default.
3. Select **From** and then expand the OL37XXCOL collection. A list of course tables appears. Click the **CARS** table and then press the **>** button.
4. Select **Select (Result Columns)**. Expand **OL37XXCOL> CARS**.
5. Select CARMAKE, CARMODEL and CARYEAR using **Ctrl** and the left mouse button.
6. Press the **>** button.
7. Click **OK**. The statement is built in the Run SQL Scripts window.
8. Click **Run > All**. The results viewer displays the result set.

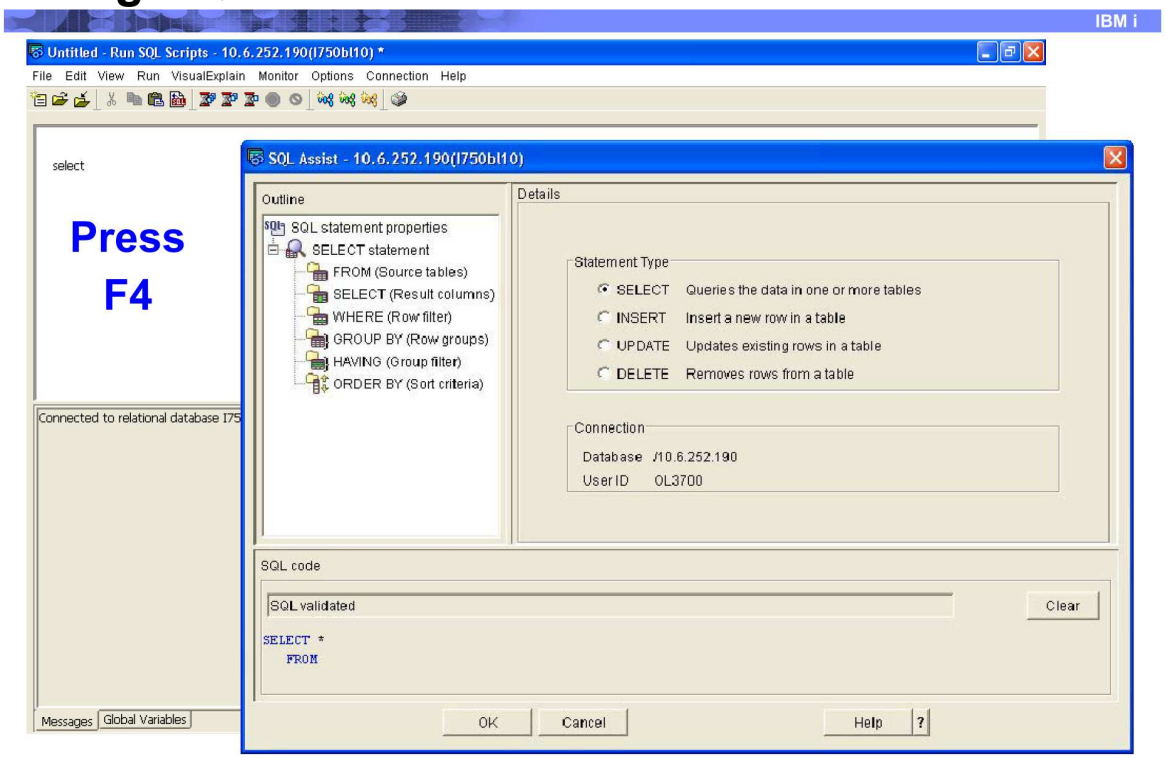
The students may use this tool in their exercises. Remind them that SQL Assist uses the *SQL naming convention and the format tab settings should be set accordingly.

Transition statement — This was a very brief overview of System i Navigator. Next we will quickly look at the 5250 legacy tool, STRSQL. The students may use either tool in the exercises.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using SQL Assist



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-71. Using SQL Assist OD477.0

Notes:

Use SQL Assist to create SQL statements. With SQL Assist and some knowledge of SQL, you can create SQL SELECT, INSERT, UPDATE, or DELETE statements.
The SQL Assist main window is comprised of three areas:

Outline view: The Outline view contains a high-level representation of the current SQL statement. With the Outline view, you can visually examine an outline of the SQL statement and navigate through the steps of building the SQL statement. Select a node in the Outline view to view details for this element in the Details area.

Details area: Use the Details area to add elements to the SQL statement. The Details area changes based on what node you select in the Outline view. When you select a node in the Outline view and make changes in the Details area, the SQL code is generated in the SQL code view.

SQL view: The SQL code view contains the SQL code that is generated based on the contents of the Outline view and changes that you made in the Details area. The code is syntax-highlighted. In some environments, you can edit the SQL code in the SQL code view.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

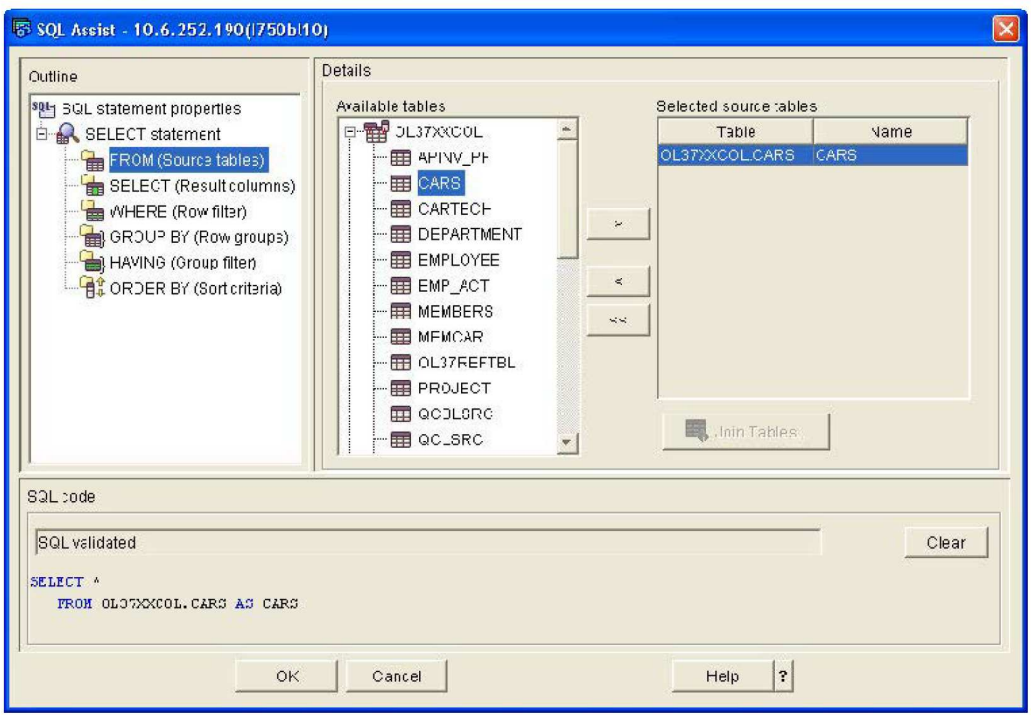
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

FROM clause

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-72. FROM clause OD477.0

Notes:

To select tables for a SELECT statement:

1. In the Outline view, click the **FROM (Source tables)** node. A list of available tables is displayed in the Details area.
2. Select tables from the **Available tables** list.
3. Click **>** to add selected tables to the **Selected tables** list.
4. Optional: Edit the alias (correlation name) for the table. Type a new name in the **Name** field.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

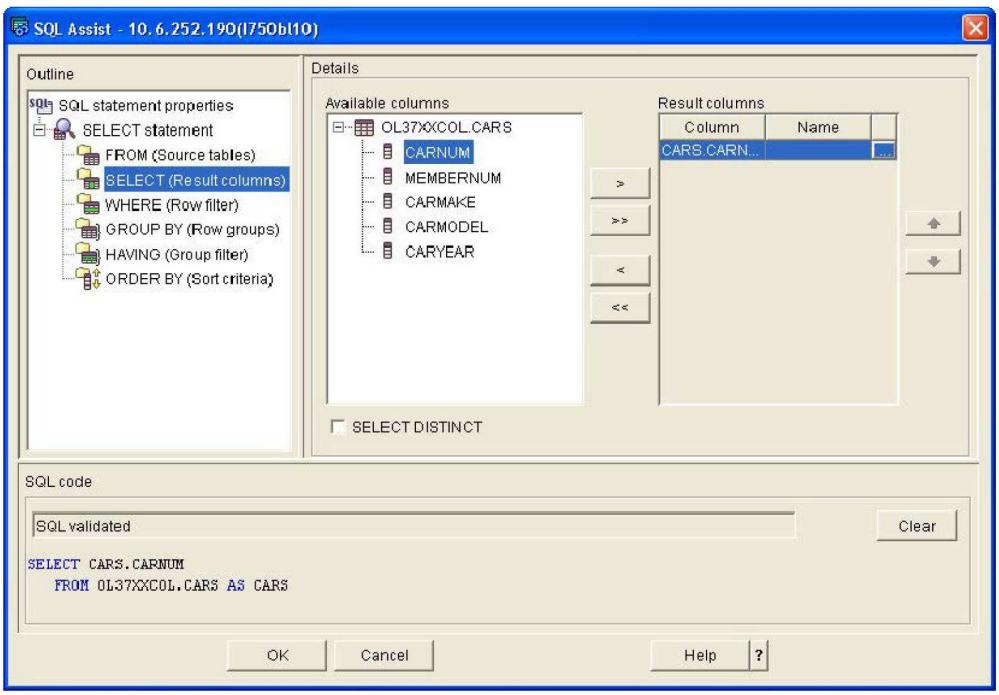
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT clause

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-73. SELECT clause OD477.0

Notes:

You must select tables for a SELECT statement before you can select result columns.

If you do not select individual result columns for the SELECT statement, all columns are returned by default.

These steps are part of the larger task of creating SELECT statements. When you complete the steps for selecting result columns (SELECT), return to Create a SELECT statement.

To select result columns for a SELECT statement:

1. In the Outline view, click the **SELECT (Result columns)** node. A list of columns from the tables that you selected in the **FROM (Source tables)** node is displayed in the Details area.
2. Click a table node in the Available columns list to see the columns in that table.
3. Select columns from the Available columns list.
4. Click **>** to move selected columns to the Selected columns list.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

5. Optional: Click the **>>** button to add all columns in the Available columns list to the Result columns list.
6. Optional: Click the **<<** button to remove all columns in the Result columns list.
7. Optional: Select the **Remove duplicate rows (SELECT DISTINCT)** checkbox to exclude duplicate rows from the result set.
8. Optional: Type a new column name in the **Name** field next to a selected column, then press **Enter** to add the new name to the SQL code in the SQL code view.
9. Optional: Click**...**next to a selected column to open a window where you can create an expression.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

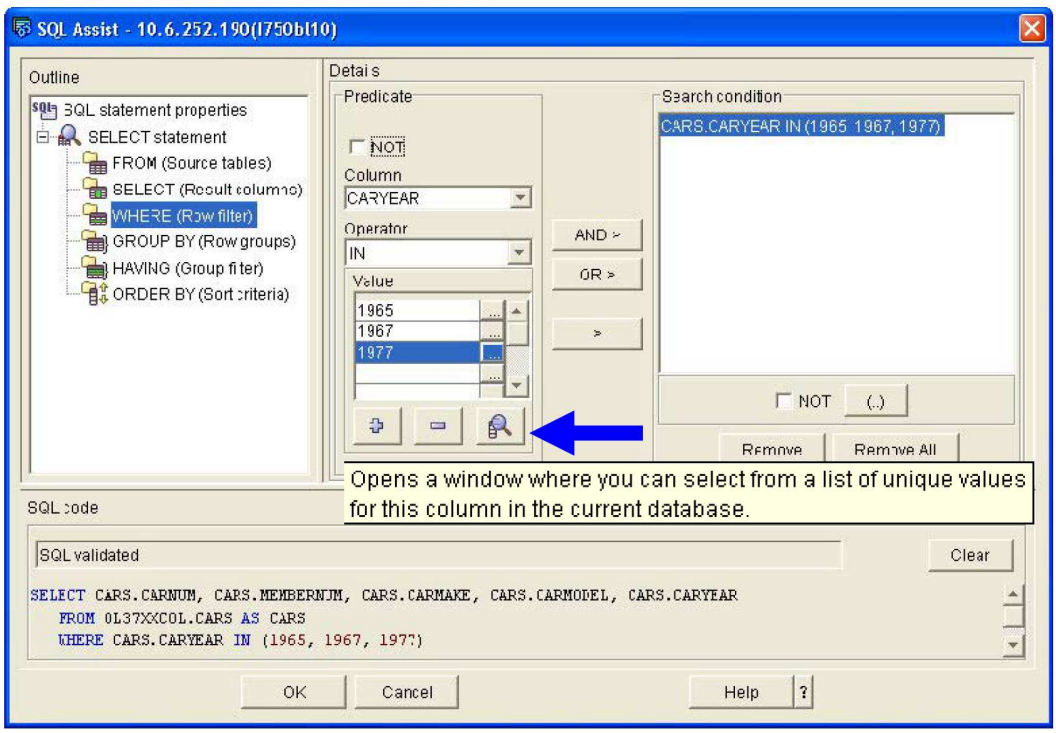
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

WHERE clause



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-74. WHERE clause OD477.0

Notes:

You must select tables for a SELECT, UPDATE, or DELETE statement before you can add a search condition in the WHERE clause.

For a SELECT statement, you should select grouping columns before you add a search condition in the HAVING clause.

A search condition specifies a condition that is true, false, or unknown about a given row. The result of a search condition is derived by application of the specified logical operators (AND, OR, NOT) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified individual predicate.

Unique values for a column can be searched using the **search** icon button.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

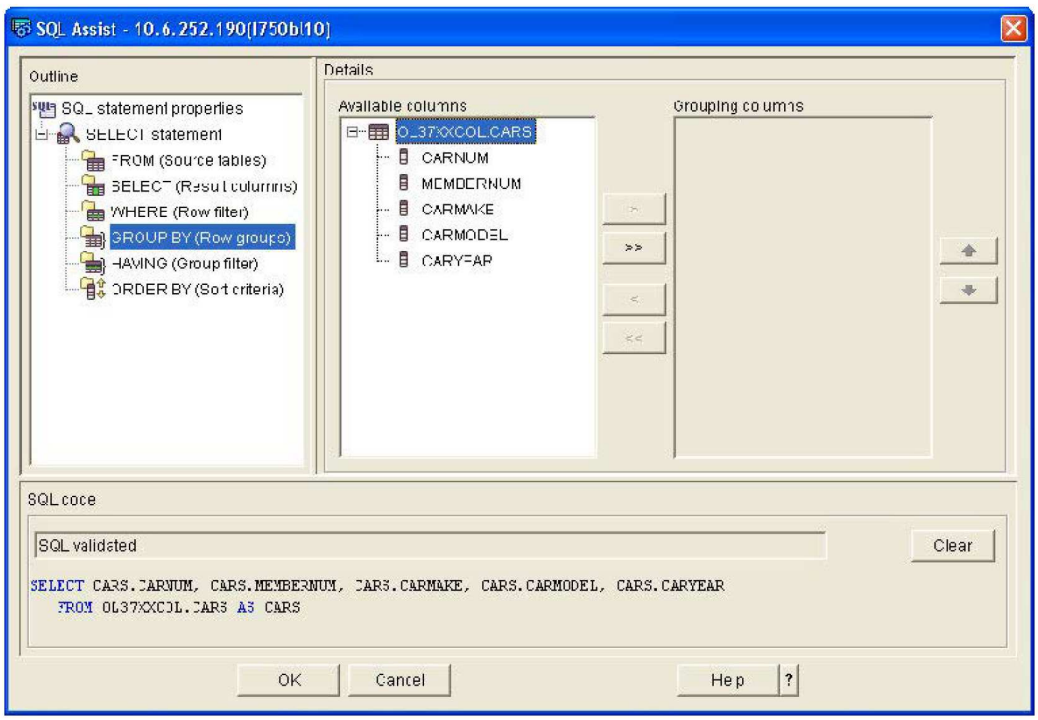
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

GROUP BY clause

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-75. GROUP BY clause OD477.0

Notes:

You must select source tables for an SQL statement before you can select grouping columns.

The GROUP BY clause specifies that rows retrieved by the WHERE clause should be grouped and condensed in such a way that all rows having the same values for the grouping columns will be treated as a single row in the result set.

SQL Assist will automatically add all columns in the SELECT clause (other than column function columns) to the grouping column list if there are any column function columns in the SELECT clause.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

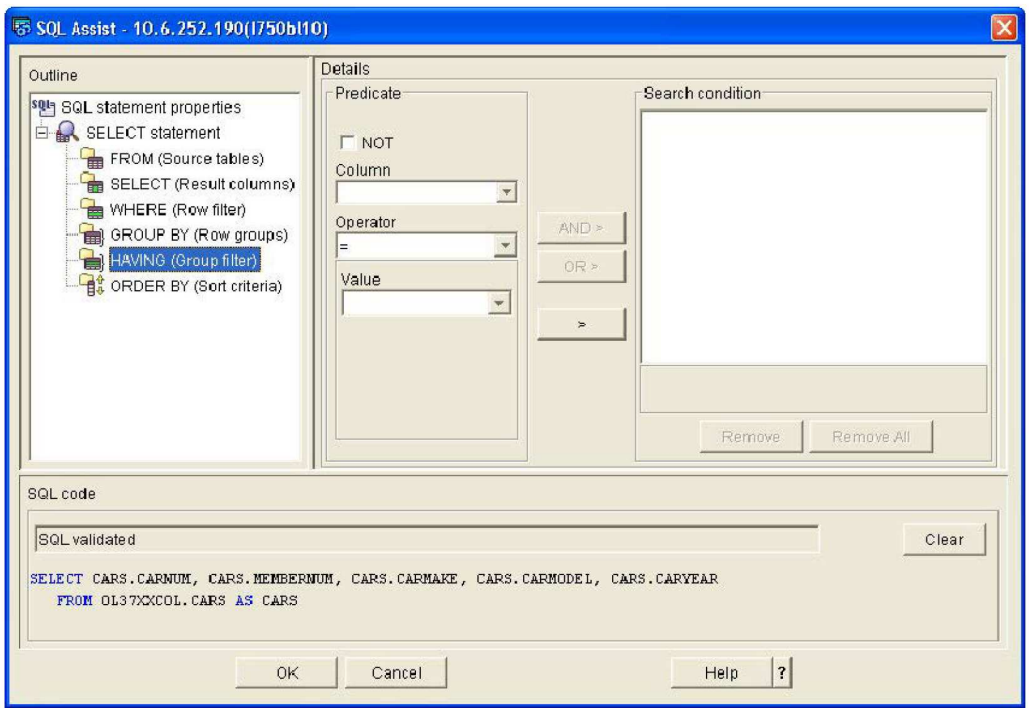
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

HAVING clause

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-76. HAVING clause OD477.0

Notes:

You must select tables for a SELECT, UPDATE, or DELETE statement before you can add a search condition in the WHERE clause.

For a SELECT statement, you should select grouping columns before you add a search condition in the HAVING clause.

A search condition specifies a condition that is true, false, or unknown about a given row. The result of a search condition is derived by application of the specified logical operators (AND, OR, NOT) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified individual predicate.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

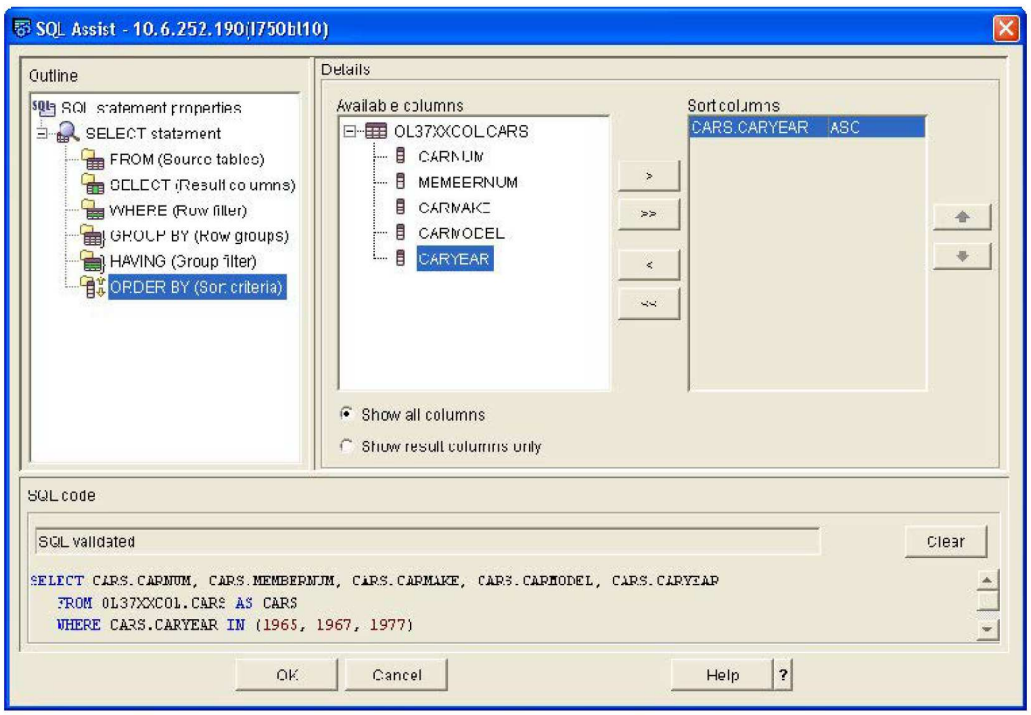
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

ORDER BY clause

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-77. ORDER BY clause OD477.0

Notes:

You must select source tables for an SQL statement before you can specify sort columns.

The ORDER BY clause specifies an ordering of the rows of the result table. If a single sort specification (one sort key with associated direction) is identified, the rows are ordered by the values of that sort specification. If more than one sort specification is identified, the rows are ordered by the values of the first identified sort specification, then by the values of the second identified sort specification, and so on.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

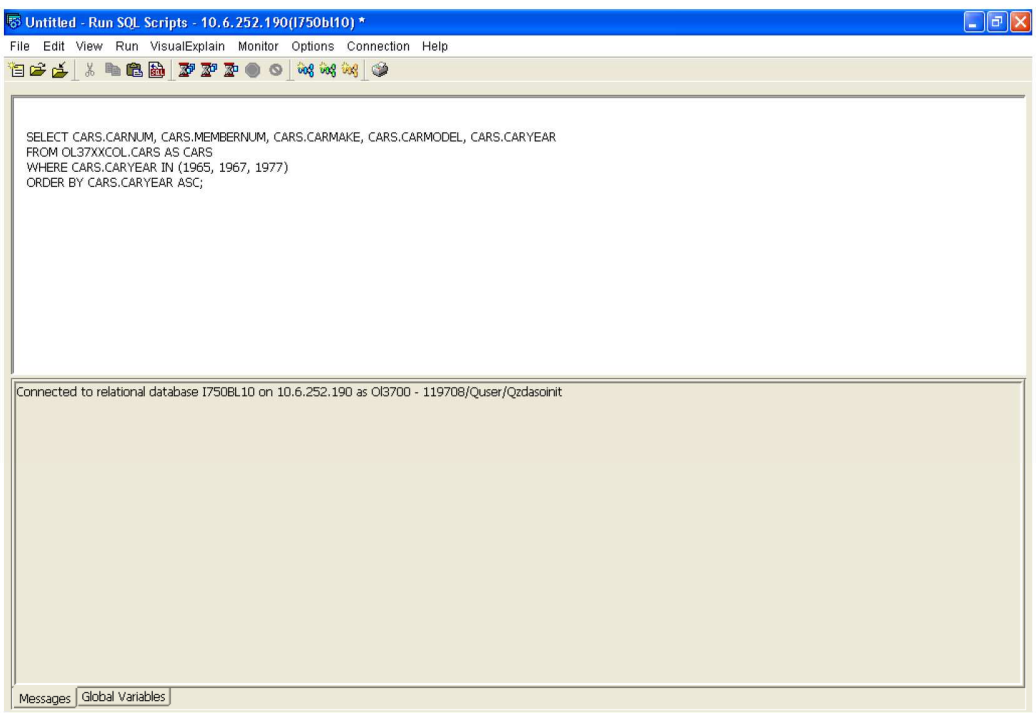
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The SELECT statement that was generated

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-78. The SELECT statement that was generated OD477.0

Notes:

This is the statement that got created by the SQL assist wizard. Highlight the statement, then click **Run** > **Selected** to execute this SQL statement.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

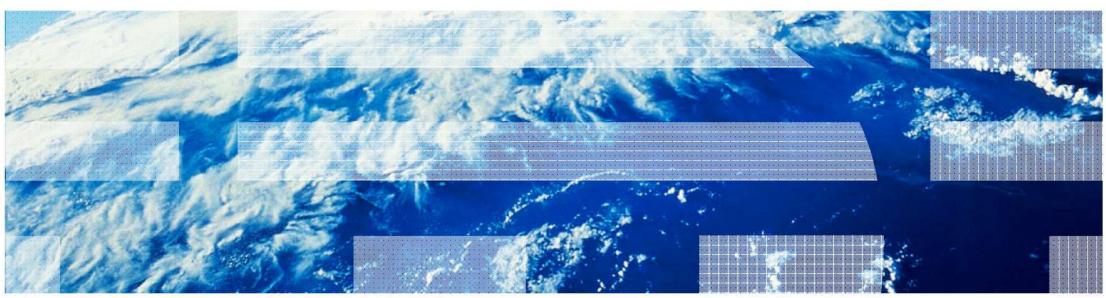
3.5. Using the 5250 STRSQL interface

Instructor topic introduction

Because 5250 emulation is still widely used, we include a topic showing this interface to the students. We will show the class how to set session options, how to use the STRSQL entry screen, how to save session data, and how to edit it to use with the `RUNSQLSTM` command.

The best way to present this topic is for you to demonstrate STRSQL on the IBM i. The file that is used, **APINV_PF**, can be found in the OL37XXCOL student master schema.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Topic 5: Using the 5250 STRSQL interface

Power^{tm} with IBM i

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

8.1

Figure 3-79. Topic 5: Using the 5250 STRSQL interface OD477.0

Notes:

As an alternative to System i Navigator's graphical interface, the STRSQL interface provides a green-screen based interface to SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

STRSQL: Prompting with F4

IBM i

```
                    Start SQL Interactive Session (STRSQL)

Type choices, press Enter.

Commitment control . . . . . COMMIT         *NONE
Naming convention  . . . . . NAMING         *SYS
Statement processing . . . . . PROCESS      *RUN
Library option . . . . . . . . LIBOPT       *LIBL
List type  . . . . . . . . . . LISTTYPE     *ALL
Data refresh . . . . . . . . . REFRESH      *ALWAYS
Allow copy data  . . . . . . . ALWCPYDTA    *YES
Date format  . . . . . . . . . DATFMT       *JOB
Date separator character . . . . DATSEP     *JOB
Time format  . . . . . . . . . TIMFMT       *HMS
Time separator character . . . . TIMSEP     *JOB
Decimal point  . . . . . . . . DECPNT       *JOB
Sort sequence  . . . . . . . . SRTSEQ       *JOB
  Library  . . . . . . . . . .
Language identifier  . . . . . LANGID       *JOB

                                                                    Bottom
F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
F24=More keys
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-80. STRSQL: Prompting with F4 OD477.0

Notes:

When you enter the `STRSQL` command and press the **F4** key, you can see, and modify, the session defaults for the interactive SQL interface.

This interface is not a production interface. It is a prototyping tool and is useful for testing individual statements. It does not offer the facilities of System i Navigator but still is a valuable tool.

One of its features is the ability to save your session in a source file that can be used to cut and paste SQL statements during the edit process. Another use for your saved your source file is that editing it to produce "**5250 based SQL Scripts**" that can be run as a group using the `RUNSQLSTM` command on the IBM i. Since this command is available as part of IBM i on all systems, you can create the SQL source members on a development system and then run them on any system. RUNSQLSTM is packaged with IBM i and does not require the licensed SQL Development Kit.

At this point, let us explain the importance of some of the parameters in the `STRSQL` command session defaults:

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Naming convention

This parameter specifies the naming convention used for objects in SQL statements.

`*SYS`	The system naming convention (library-name/file-name) is used. If you use this convention and the schema (library) can be found in the library list, you can omit the library name. This is the default value.
`*SQL`	The SQL naming convention (schema-name.table-name) is used. If you use this convention, the schema will usually need to be specified. It will search for a library/schema matching the user's authorization ID (the user profile).

Statement processing

This parameter determines how your SQL statements will be prepared and run.

`*RUN`	The statements are syntax checked, data checked, and then run. This is the default.
`*VLD`	The statements are syntax checked and data checked, but not run.
`*SYN`	The statements are syntax checked only.

Library option

This parameter specifies the list of libraries that are shown when you request list support from the Enter SQL Statements window (F4, F16, F17, and F18). If you do not have *EXECUTE authority for a specified library, that library is not shown.

`*LIBL`	All libraries in the job's library list are searched until the first match is found. This is the default value.
`*CURLIB`	The job's current library is shown.
`*USRLIBL`	Only the libraries in the user portion of the job's library list are shown.
`*ALL`	All of the libraries in the system, including QSYS, are shown.
`*ALLUSR`	All user libraries are searched.

Session defaults

These are important. When you have problems using STRSQL, you can often solve problems by changing a session option. For example, Naming Convention of `*SQL` preforms a different search than `*SYS`. Also, it is often best to specify the specific date format that you want to use. For example, `*JOB` will use a 6-digit date while `*ISO` will use an 8-digit date.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Explain that these parameters are very important. Remind them, for example, that the naming convention (`*SYS` or `*SQL`) is important. The IBM i `STRSQL` command defaults to using `*SYS` naming convention. However, if you use System i Navigator which uses JDBC, its default is the `*SQL` naming convention. If you try to access SQL data and receive an error using one of the naming conventions, try the command again using the other naming convention.

Transition — At this point you can choose to lead the demonstration with or without class participation. If you choose to perform the demonstration yourself, proceed. If you choose to have the class participate and use their terminals, then you must explain a few things to the class.

It will be necessary to explain to the students that each of them will be using the unique schema that they already created.

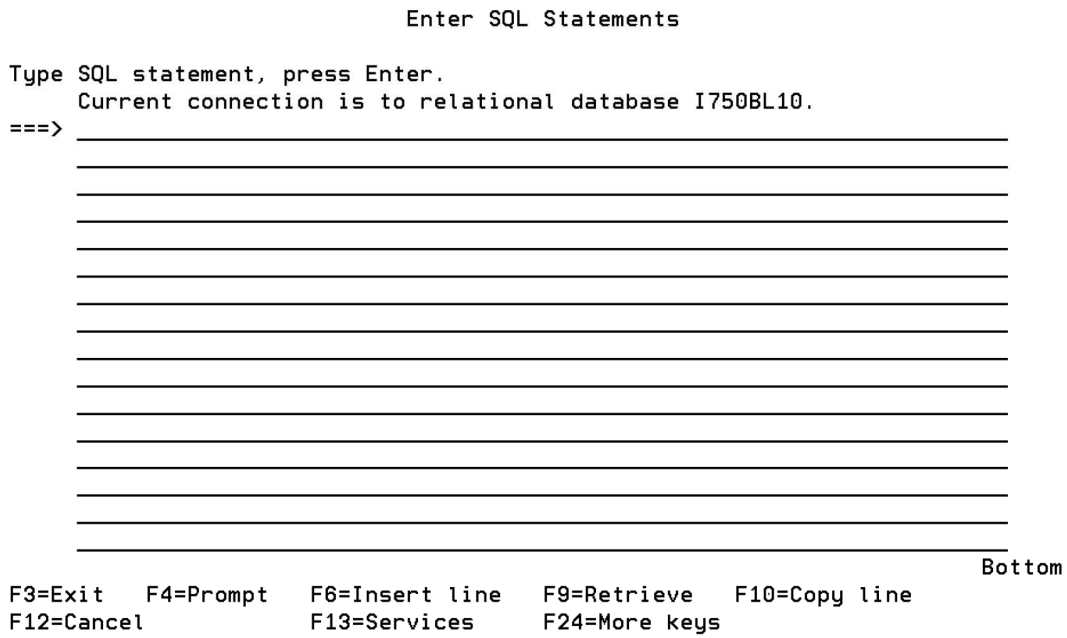
Remember, **OL37nnCOL** is the schema created by team **OL37nn,** for example, **OL3701COL** is created by team **OL3701**.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

STRSQL entry

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-81. STRSQL entry OD477.0

Notes:

This is your entry screen. SQL statements are entered and run *one SQL statement at a time*.

1. You key the statement and press **Enter**.
2. The statement is syntax-checked.
3. If the statement is syntactically correct, it is executed.

This process is one type of *Dynamic SQL*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Because SQL views the local and remote systems that can be accessed simply as databases, point out that the local database on this system is known by SQL as the IBM i system name.

Demo

If the class is participating in the demonstration, they must now do the following:

1. On the SQL command display, they should enter

   ```
   SELECT PONBR, VNDNBR, APINVNBR, APDATE, APNETPAID, APDATEPAID FROM
       apinv_pf
   ```

   They may need to fully qualify the schema and the table using the student master schema name, OL37XXCOL.

2. Point out to the class that the APINV_PF is a physical file, not an SQL table. It is located in the student master schema if they did not copy it to their own schema. We can query it as if it were a table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Session defaults are important

IBM i

SELECT PONBR, VNDNBR, APINVNBR, APDATE, APNETPAID, APDATEPAID FROM OL37XXCOL/apinv_pf

```
                                  Display Data
                                                Data width . . . . . . :      57
Position to line  . . . . .  ______      Shift to column  . . . . . .   _____
....+....1....+....2....+....3....+....4....+....5....+..
 Purch      Vend   Vendor    Date        Net     Date
 Order      Num    Invoice   Compl       Amount  Paid
 Number            Number    YYYYMMDD    Paid    YYYYMMDD
300,071   10,005   5000      04/01/96    950.00  04/11/96
300,072   10,005   1005      06/01/96       .00  ++++++++
300,073   10,015   1580      04/01/98   2850.00  04/11/98
300,074   10,020   2007      05/15/98   4000.00  ++++++++
300,075   10,025   25697     03/15/98       .00  ++++++++
300,076   10,030   301       08/03/98       .00  ++++++++
300,077   10,035   350001    02/01/98    630.00  03/15/98
300,078   10,040   40079     09/03/98       .00  ++++++++
300,079   10,045   45090     06/03/98       .00  ++++++++
300,080   10,050   500777    07/22/98       .00  ++++++++
300,081   10,005   5001      06/26/98       .00  ++++++++
300,082   10,010   1006      05/23/98       .00  ++++++++
300,083   10,015   1583      09/02/98   3800.00  09/30/98
300,084   10,020   2013      01/31/98   4750.00  03/15/98
                                                                  More...
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-82. Session defaults are important OD477.0

Notes:

That was simple. With a few moments work, we were able to produce a display of the complete table. We knew nothing about the description of the data itself. That information was obtained from the DDS used to create the physical file.

APINV_PF

This file was created, using DDS to describe it, many years ago for an RPG based application. Since there was no null support, the programmer used a different technique. Today, we would design the table such that the date would be null until the invoice was paid.

The file has been converted to use date data types but null support has not yet been implemented in this application.

Notice that:

- Default editing is applied to each numeric field, depending upon its data type.
- For the date data type, the date format is included in the heading.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Note the **++++** in certain date fields. This means that the value of the date is not what the system expects.

Date Value of +++++

Can you explain why there is a problem with some of the values for dates?

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — Teach students how to set the date format for a 4 digit date field.

Details — The **+++++** for the date field is because the year is 9999. For the date data type, (where a four digit year is valid), this is a valid date, but for this SQL query, the year value of **9999** causes a date overflow condition because the default date format for the interactive SQL session is *MDY where the year is only two digits. Valid years are between 1940 and 2039. Any year outside that range is invalid. This also applies when using the *JUL, *DMY, and *YMD date formats.

Transition — Now, let us correct the problem with the date format. Press **F3** to return to the ENTER SQL STATEMENTS display. Then press **F13**.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Change defaults

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-83. Change defaults OD477.0

Notes:

In order to change session defaults, we press **F13** from the Enter SQL Statements display. Then we select option **1**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

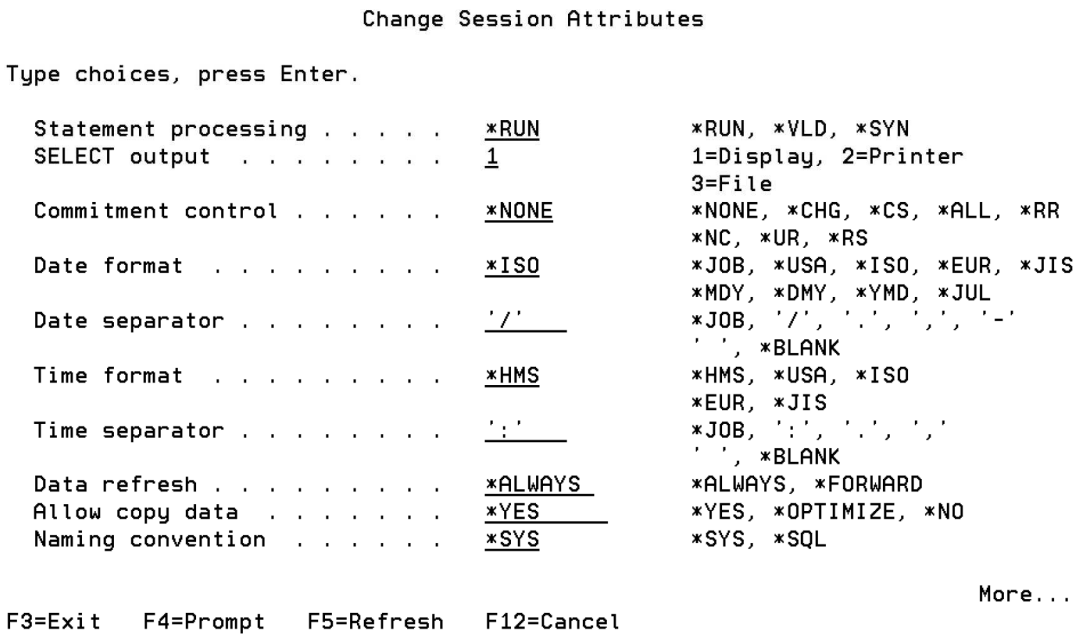
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Change session attributes

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-84. Change session attributes OD477.0

Notes:

Change the date format from *MDY to *ISO or the format standard for a four digit year in your country. Note the other options that can be changed. While you are viewing this display, review the options that you can modify.

Tip

Many of these attributes, including the date format, can be specified when you begin your interactive SQL session.

When you are finished reviewing the various options, press **ENTER** until you return to the Interactive SQL Services display. Then, press **F3** or **F12** to return to the ENTER SQL STATEMENTS display. Rerun the previous `SELECT` command to review the change.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — If you exit your SQL session and save it, you will save the existing session defaults, which normally specify `*JOB` as the default for the date format. If you `STRSQL` again and prompt it, you can change the `DATFMT` parameter. However, as soon as you press **enter**, your previous session defaults are restored, negating the change you just made.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Rerun query

IBM i

```
                                  Display Data
                                                Data width . . . . . . :      61
Position to line  . . . . .  ______      Shift to column  . . . . . .  _____
....+....1....+....2....+....3....+....4....+....5....+....6.
 Purch      Vend   Vendor    Date          Net     Date
 Order      Num    Invoice   Compl         Amount  Paid
 Number            Number    YYYYMMDD      Paid    YYYYMMDD
300,071   10,005   5000      1996-04-01    950.00  1996-04-11
300,072   10,005   1005      1996-06-01       .00  9999-12-31
300,073   10,015   1580      1998-04-01   2850.00  1998-04-11
300,074   10,020   2007      1998-05-15   4000.00  9999-06-01
300,075   10,025   25697     1998-03-15       .00  9999-12-31
300,076   10,030   301       1998-08-03       .00  9999-12-31
300,077   10,035   350001    1998-02-01    630.00  1998-03-15
300,078   10,040   40079     1998-09-03       .00  9999-12-31
300,079   10,045   45090     1998-06-03       .00  9999-12-31
300,080   10,050   500777    1998-07-22       .00  9999-12-31
300,081   10,005   5001      1998-06-26       .00  9999-12-31
300,082   10,010   1006      1998-05-23       .00  9999-12-31
300,083   10,015   1583      1998-09-02   3800.00  1998-09-30
300,084   10,020   2013      1998-01-31   4750.00  1998-03-15
                                                                   More...
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-85. Rerun query OD477.0

Notes:

Now we have our date formatted correctly. Press **F3** to return to the ENTER SQL STATEMENTS display. Press **F9** to retrieve your query and then press **ENTER** to rerun it.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

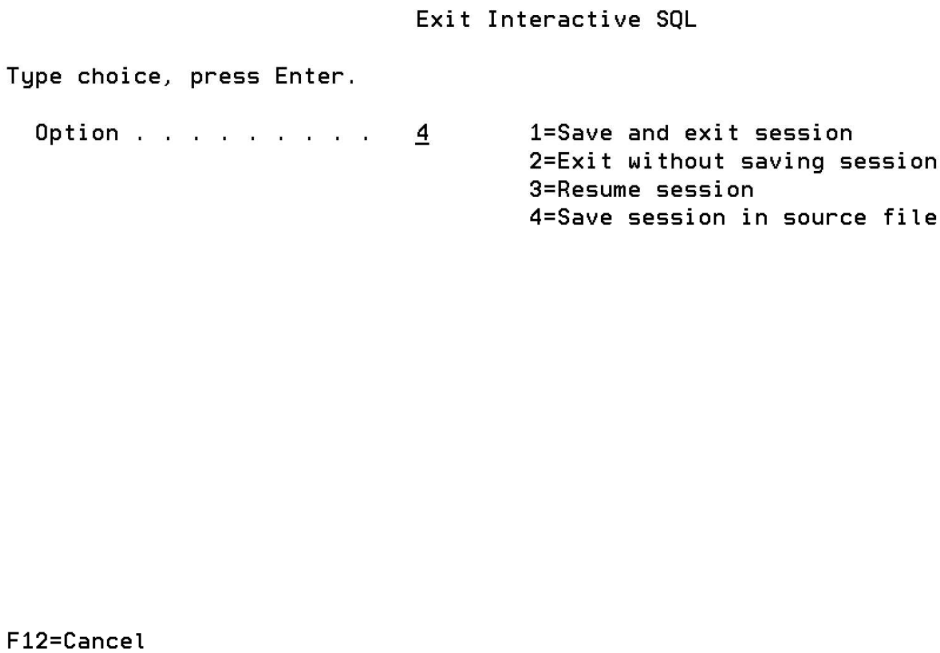
Details — This is the end of the demonstration part of this topic.

Transition statement — Next you will show the class how to save their session data for later cut and paste or for running with RUNSQLSTM.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exit and save session in source file

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-86. Exit and save session in source file OD477.0

Notes:

Pressing **F3** from the ENTER SQL STATEMENTS will get you to this display. By default, Option **1**, Save and exit session, is selected. If you wish, you may save the SQL Query you have created in a source file.

To be certain of saving your SQL from a query session correctly, you should save it a source file. If a single user has multiple 5250 sessions going with a STRSQL session in each, it is possible that the internal system area to store the user's interactive SQL will be unreliable. So option **4** is the only way to be sure that your session has been correctly saved.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

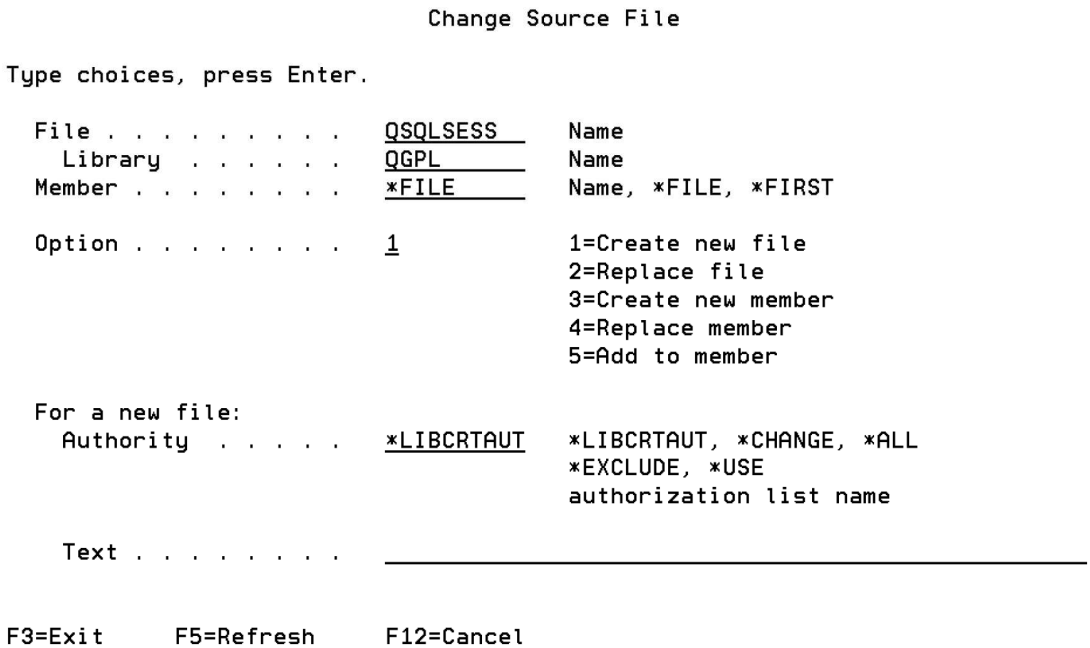
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Saving in source file

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-87. Saving in source file OD477.0

Notes:

When you save your session, you may use the default file (and member) name of QSQLSESS or specify a name of your choice.

Notice the options available to you when you save your latest session. The default is to create a new file. Be sure to specify a library of choice, like OL37nnCOL, in place of the default, QGPL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

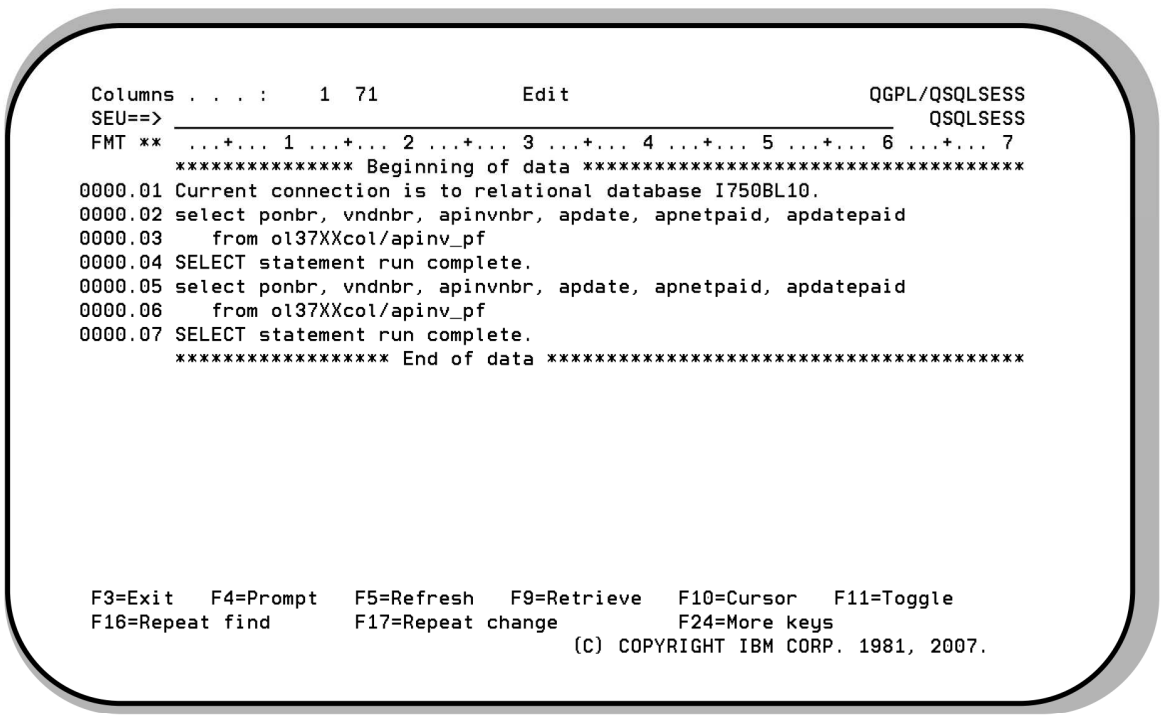
Instructor notes:

Transition statement — Next we will show the students how to edit the session source file and clean it up.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Contents of source file

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-88. Contents of source file OD477.0

Notes:

Here are the contents of our simple query session. All data from your session has been saved, our two SELECT statements, including any messages. We can use this source member to cut and paste our statements for later use. If we had non-SELECT statements in the source member, we could use the SQL statements to create a script the can be processed by RUNSQLSTM (the SQL statement processor). To do so you must:

1. Delete all messages.
2. Delete any statements that you do not want to keep.
3. Delete any statements that cannot be run by the SQL statement processor. SELECT *cannot* be run using this tool so they must be deleted.
4. Add a semicolon (**;**) at the end of every SQL statement. The semicolon acts as a delimiter for individual statements.

Notice that there is no syntax checking in SEU for SQL statements. You should have already done this by prototyping individual statements using STRSQL. There is no SEU type for SQL and it is not necessary to enter one.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

In contrast, the **Run** pull-down of the System i Navigator menu bar includes the options to **Syntax Check**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Remind the students that even though a SELECT statement is used in the example on this visual, SELECT statements are not executable from a source member utilizing the `RUNSQLSTM` command.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Cleanup and create SQL script

IBM i

1. Remove messages.
2. Delete unwanted statements.
3. Add a semicolon (;) at the end of each SQL statement.

```
        ********** Beginning of data *************************************
0001.00 INSERT INTO OL37XXCOL/CARS VALUES(00010, 'slaney', 'bob',
0002.00 'chevrolet', 'corvette', 1967, 35.00);
0003.00 INSERT INTO OL37XXCOL/CARS VALUES(00020, 'slaney', 'amy',
0004.00 'oldsmobile', 'cutlass 442', 1967, 35.00);
0005.00 INSERT INTO OL37XXCOL/CARS VALUES(00030, 'eaton', 'john',
0006.00 'chevrolet', 'corvette', 1970, 35.00);
0007.00 INSERT INTO OL37XXCOL/CARS VALUES(00030, 'eaton', 'adele',
0008.00 'ferrari', '308', 1985, 35.00);
0009.00 INSERT INTO OL37XXCOL/CARS VALUES(00040, 'dingman', 'daryl',
0010.00 'chevrolet', 'el camino', 1977, 35.00);
0011.00 INSERT INTO OL37XXCOL/CARS VALUES(00050, 'bradley', 'stan',
0012.00 'chevrolet', 'corvette', 1985, 35.00);
0013.00 INSERT INTO OL37XXCOL/CARS VALUES(00060, 'vamone', 'bruce',
0014.00 'dodge', 'super bee', 1969, 35.00);
0015.00 INSERT INTO OL37XXCOL/CARS VALUES(00060, 'shiller', 'george',
0016.00 'chevrolet', 'impala', 1965, 35.00);
```

4. Save the source. You can use the same name or a new name for the member.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-89. Cleanup and create SQL script OD477.0

Notes:

This visual shows a saved source member being edited during the process of cleaning up the data from that saved session.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

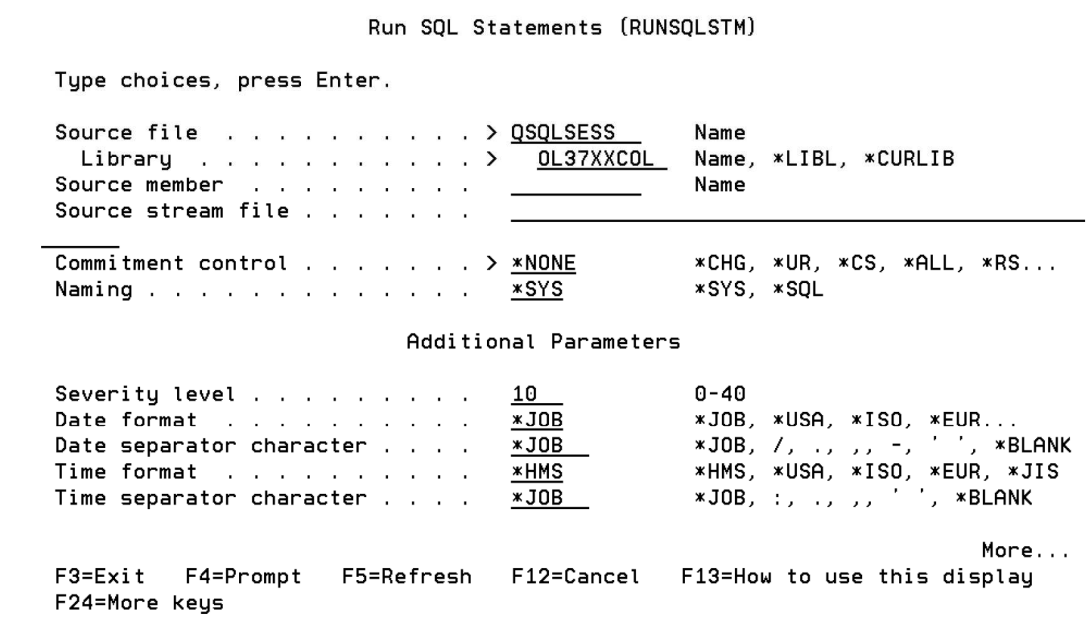
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

RUNSQLSTM

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-90. RUNSQLSTM OD477.0

Notes:

Notice that **RUNSQLSTM** has parameters that require careful attention. We will not look at all of them, but notice:

- Commitment control: If not implemented, specify *NONE.
- Naming: *SYS or *SQL
- Date format: Select the format you use.

Specify the location and name of the source file that holds the SQL script data.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

RUNSQLSTM steps

IBM i

1. SQL statements are processed one statement at a time.
2. Fix errors in the source.
3. Try again.

Always test statements interactively one at a time.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-91. RUNSQLSTM steps OD477.0

Notes:

This visual summarizes the steps that you should follow when creating an SQL Script. Before you create your SQL Script, you should always test individual statements interactively, using System i Navigator or the 5250 STRSQL interface.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

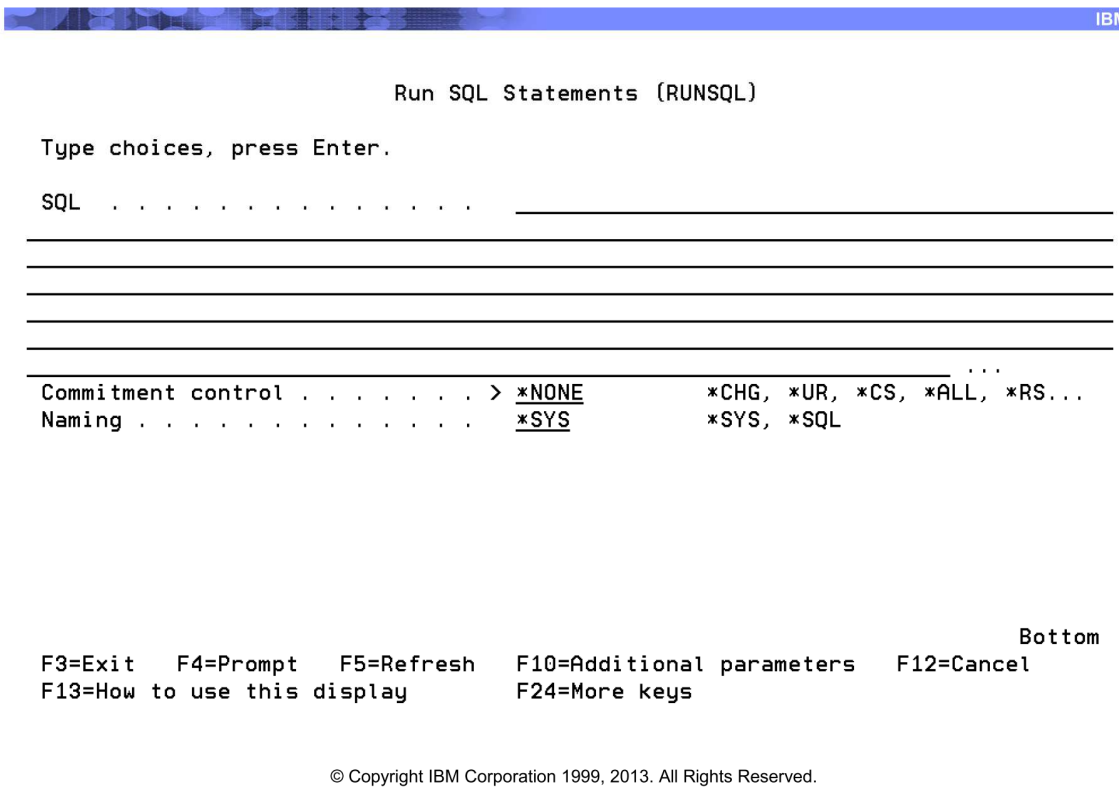
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

RUNSQL

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-92. RUNSQL OD477.0

Notes:

The `RUNSQL` CL command (shown prompted from the IBM i command line) allows an SQL statement to be run from within a CL program or command line without needing a source file. The SQL (parameter) statement string can be up to 5000 characters long and it must not end with a semicolon. SQL statements such as INSERT, DROP, DELETE and UPDATE are just a few of the allowed statements. SELECT is not allowed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U02** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U02** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U02** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint (1 of 2)

IBM i

1. Place the following clauses in the order in which they must be coded with a SELECT statement.

   ___ HAVING
   ___ WHERE
   ___ GROUP BY
   ___ FROM
   ___ ORDER BY

2. True or False: In order to SELECT all columns from a table, you can code SELECT * or SELECT ALL.

3. True or False: When ORDER BY specifies multiple columns, you must remember that the result set is sequenced by the last column, within the next to last column, and so on.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-93. Checkpoint (1 of 2) OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions (1 of 2)

IBM i

1. Place the following clauses in the order in which they must be coded with a SELECT statement.

   _4_ HAVING
   _2_ WHERE
   _3_ GROUP BY
   _1_ FROM
   _5_ ORDER BY

   The answers are 1 – FROM, 2 – WHERE, 3 – GROUP BY, 4 – HAVING, 5 – ORDER BY.

2. True or False: In order to SELECT all columns from a table, you can code SELECT * or SELECT ALL.

   The answer is False. You can code SELECT *. Also, you can code each individual field. If you want to resequence the selected columns, you must code them individually.

3. True or False: When ORDER BY specifies multiple columns, you must remember that the result set is sequenced by the last column, within the next to last column, and so on.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint (2 of 2)

IBM i

4. When coding a condition that includes a character literal, what should you be careful to do when coding the value?

5. Which lines of code below are valid?
   a. WHERE quantity > 1000 and quantity < = 10000
   b. Where (quantity > = 1000) and (season = ‘Christmas’)
   c. WHERE (year < = 2000 and > = 1998)

6. Code the SQL WHERE clause that will limit the selection of rows to those columns of the name 'COUNTRY' that have the string 'Amer' anywhere in the value.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-94. Checkpoint (2 of 2) OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions (2 of 2)

IBM i

4. When coding a condition that includes a character literal, what should you be careful to do when coding the value?

   The answer is Enclose the value in quotes and be aware of case.

5. Which lines of code below are valid?

   a. WHERE quantity > 1000 and quantity < = 10000
   b. Where (quantity > = 1000) and (season = 'Christmas')
   c. WHERE (year < = 2000 and > = 1998)

   The answers are WHERE quantity > 1000 and quantity < = 10000 and Where (quantity > = 1000) and (season = 'Christmas').

6. Code the SQL WHERE clause that will limit the selection of rows to those columns of the name 'COUNTRY' that have the string 'Amer' anywhere in the value.

   The answer is WHERE COUNTRY LIKE '%Amer%'.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Using the SELECT statement*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-95. Exercise OD477.0

Notes:

Perform the exercise *Using the SELECT statement*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Use the CREATE statement to build a table, view, or an index
- List the data types supported by SQL
- Write SQL queries using SELECT
- Use predicates to define the scope of your query
- Use the 5250 STRSQL tool to prototype SQL statements
- Use the RUNSQLSTM facility to run a group of SQL statements
- Use System i Navigator to create and Run SQL Scripts
- Use System i Navigator to display tables

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 3-96. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 4. Functions

Estimated time

01:50

What this unit is about

This unit describes how to manipulate data using some of the column functions such as AVG, COUNT, MAX, MIN, and SUM.

We will also discuss scalar functions such as ABSVAL, CHAR, CURDATE, and CURTIME. While not all of the more than one hundred and eighty (180+) scalar functions will be discussed, the students will learn how to use some of the main ones and where to find information on how to use the others.

In this unit, we will use the familiar CARCLUB database as well as the Corporate database that we used in the previous workstation exercise.

What you should be able to do

After completing this unit, you should be able to:

- Describe the purpose of column and scalar functions
- Use column functions in SQL queries
- Use scalar functions in SQL queries
- Code CASE and CAST specifications

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercise

  There are many functions available in SQL. The students will perform a workstation exercise to demonstrate their understanding of column and scalar functions.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe the purpose of column and scalar functions
- Use column functions in SQL queries
- Use scalar functions in SQL queries
- Code CASE and CAST expressions

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-1. Unit objectives OD477.0

Notes:

The notes contain many of the column and scalar functions that you might use in SQL. We will not be able to cover all of them, but, they are in the notebook and you may reference them there.

Also, the functions are described in the *SQL Reference Manual*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Cover this unit as quickly as possible as you will need at least half a day tomorrow for the database maintenance unit. Tell the class that any functions not covered in lecture can be reviewed by the students on their own time or, that when the last unit is covered, those who are not interested in it can come back to this unit and review those functions that were not covered in lecture. In addition, you will have to try to hurry up the lab and give the user students time later in the course to complete the lab.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

4.1. Column functions

Instructor topic introduction

What students will learn — Students will learn how to use column functions in SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

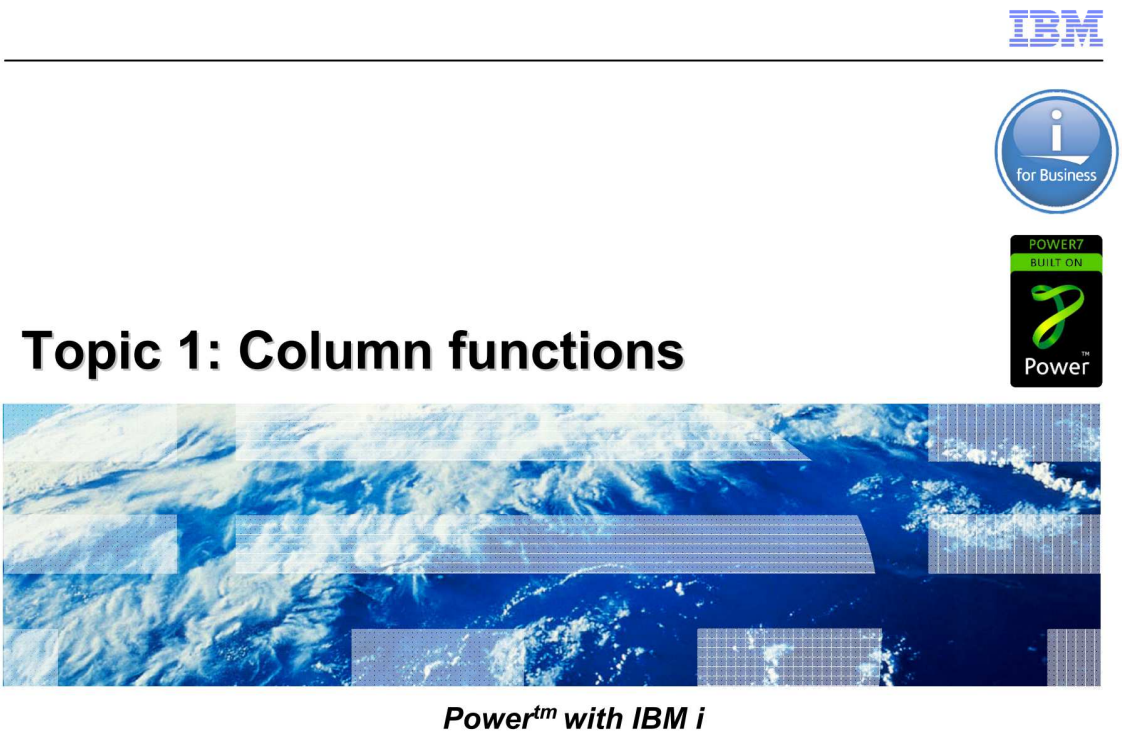


Figure 4-2. Topic 1: Column functions OD477.0

Notes:

Let's learn how to use column functions in SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Column functions

IBM i

```
AVG(x) ............. Average Value

COUNT(*) ........... Count Selected Rows

SUM(x) ............. Sum of a set of numbers

MIN(x) ............. Minimum Value

MAX(x) ............. Maximum Value

STDDEV(x) .......... Standard Deviation

VAR(x) ............. Variance from Mean
```

- Where *x* = column name or expression
- Operates on a set of values (column)
- Can be used wherever the expression is valid in SQL

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-3. Column functions OD477.0

Notes:

Functions are keywords in SQL, used to manipulate values within columns for output purposes. A function is always used in conjunction with a column name or expression. The syntax for a function is the function name followed by one or more arguments that are enclosed in parentheses. Most functions have a single argument that is specified by an expression. The result of a function is a single value derived by applying the function to the result of the expression.

SQL functions are classified as scalar functions or column functions. The argument of a column function is a set of values in a column. All rows are preprocessed to determine the value of the function.

In the syntax of SQL, the term function is used only in the definition of an expression. Thus, a function can be used only where an expression can be used.

We will discuss several examples of commonly used functions. STDDEV and VAR (or VARIANCE) are not shown in further detail. These two functions are related. VAR will find the variance from the biased variance for a set of numbers and STDDEV will find the standard deviation from this variance.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — We will discuss some of these functions, not all of them. While there are many covered in the notes, some will be for student reference and will not be covered in lecture.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Column functions and calculated values

IBM i

- Used only with numeric values:
  - SUM
  - AVG
- Used with numeric or character values:
  - MIN
  - MAX
  - COUNT

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-4. Column functions and calculated values OD477.0

Notes:

The argument of a column function is a set of values derived from one or more columns, except for COUNT (*). The result of the set of values is a group or an intermediate result table.

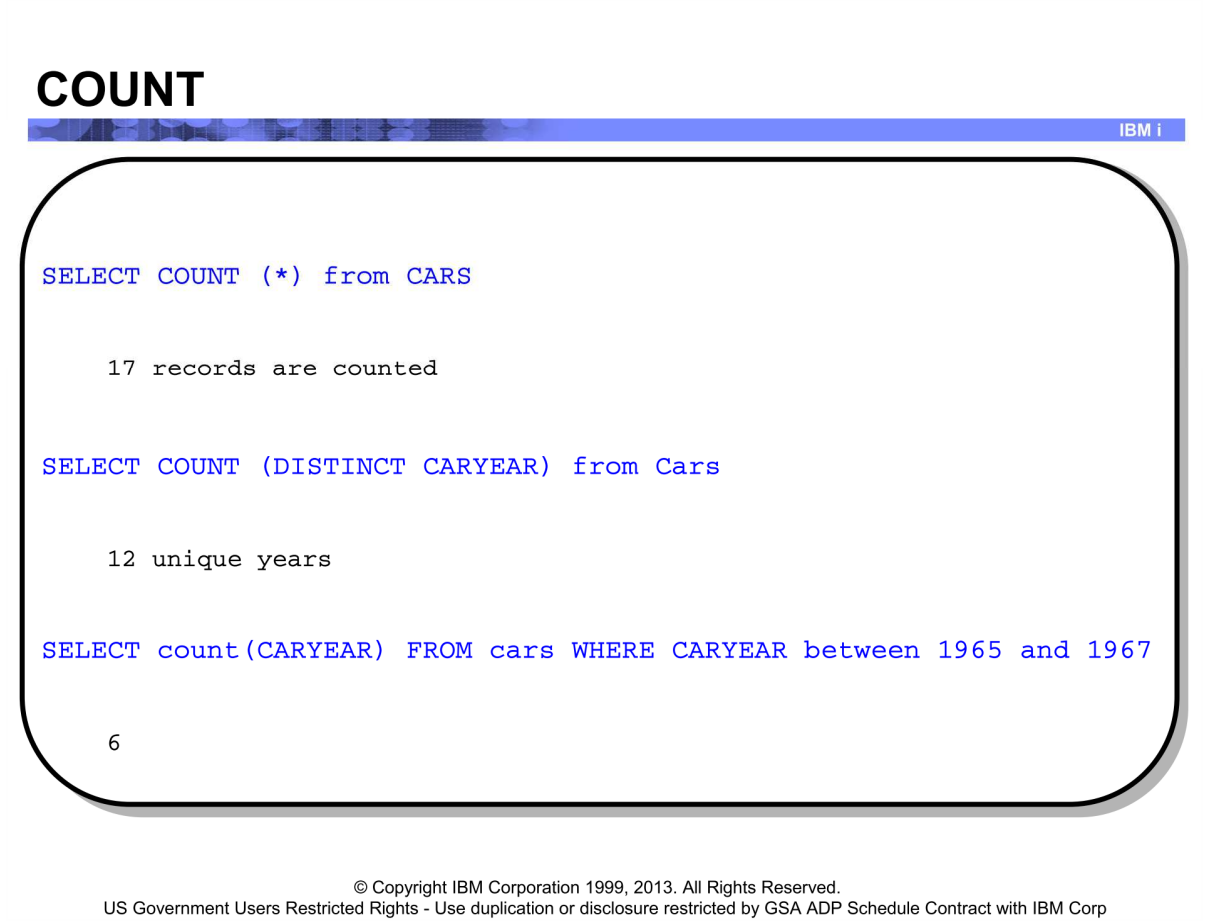
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition — Let us take a look at the COUNT function.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-5. COUNT OD477.0

Notes:

The **COUNT** function returns the number of rows or values in a set of rows or values.

In the first example, the COUNT (*) counts all columns.

In the second example, the keyword DISTINCT is not an argument of the COUNT function. It is a specification of an operation that is performed before the COUNT is performed. Since DISTINCT is specified, duplicate values of the column CARYEAR are eliminated.

In the last example, we are counting how many cars were built in the 1965, 1966, and 1967 model years.

Note that the format shown for the COUNT function is common for all column functions.

1. COUNT(ALL CARYEAR) is the same as COUNT(CARYEAR). The ALL is implied.
2. You *cannot* use any column functions in a WHERE clause.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Run this statement for the class:

```
select *
       from cars
       where count(*) > 1
```

Point out that column functions cannot be done in a WHERE.

Also point out that ALL is implied in these functions. ALL is the opposite of DISTINCT.

I added the verbiage to the values of the COUNT function in this visual. When using STRSQL or System i Navigator, you see the column heading and the value only.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

AVG

IBM i

```
select carmake, avg(caryear) from cars
       group by carmake

CARMAKE                          AVG ( CARYEAR )

Alfa Romeo       1,973.0000000000000000000000000000
Austin           1,965.0000000000000000000000000000
Chevrolet        1,972.8000000000000000000000000000
Dodge            1,968.0000000000000000000000000000
Ferrari          1,984.0000000000000000000000000000
Ford             1,972.3333333333333333333333333333
Maserati         1,971.0000000000000000000000000000
Mercedes         1,974.0000000000000000000000000000
Oldsmobile       1,967.0000000000000000000000000000
Plymouth         1,968.3333333333333333333333333333
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-6. AVG OD477.0

Notes:

The **AVG** function returns the average value of a set of numbers. The argument values must be numbers and their sum must be within the range of the data type of the result. The result can be null.

The data type of the result is the same as the data type of the argument values, except:

- The result is double-precision floating point if the argument values are single-precision floating point.
- The result is large integer if the argument values are small integers.
- The result is decimal if the argument values are nonzero scale binary.

If the data type of the argument is decimal (zoned or packed) or nonzero scale binary with precision p and scale s, the precision of the result is 31 and the scale is 31-p+s. In our example, CARYEAR is packed (4,0). Thus, the scale of the result is 27 decimal places.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

MIN/MAX

IBM i

```
select carmake, Min(caryear) as oldest,
       max(caryear) as newest
       from cars
       group by carmake order by carmake

CARMAKE          OLDEST   NEWEST

Alfa Romeo        1,973    1,973
Austin            1,965    1,965
Chevrolet         1,965    1,985
Dodge             1,969    1,969
Ferrari           1,984    1,984
Ford              1,957    1,965
Maserati          1,971    1,971
Mercedes          1,974    1,974
Oldsmobile        1,967    1,967
Plymouth          1,966    1,970
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-7. MIN/MAX OD477.0

Notes:

MIN and **MAX** find the minimum and maximum values in the argument. If, for each make of car, we want to know the oldest and newest model year, MIN and MAX can tell us.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SUM

IBM i

```
SELECT SUM (BONUS)

       FROM EMPLOYEE

       WHERE JOB = 'CLERK'


SUM (BONUS)

     2,800.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-8. SUM OD477.0

Notes:

The **SUM** function returns the sum of a set of numbers.

In the example, we want to determine the total bonus paid to all clerks in the company.

The argument values must be any built-in numeric data type and their sum must be within the range of the data type of the result.

The data type of the result is the same as the data type of the argument values except in certain cases. Please see the reference document for more information.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

4.2. Scalar functions

Instructor topic introduction

What students will learn — Students will learn how to code scalar functions in SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

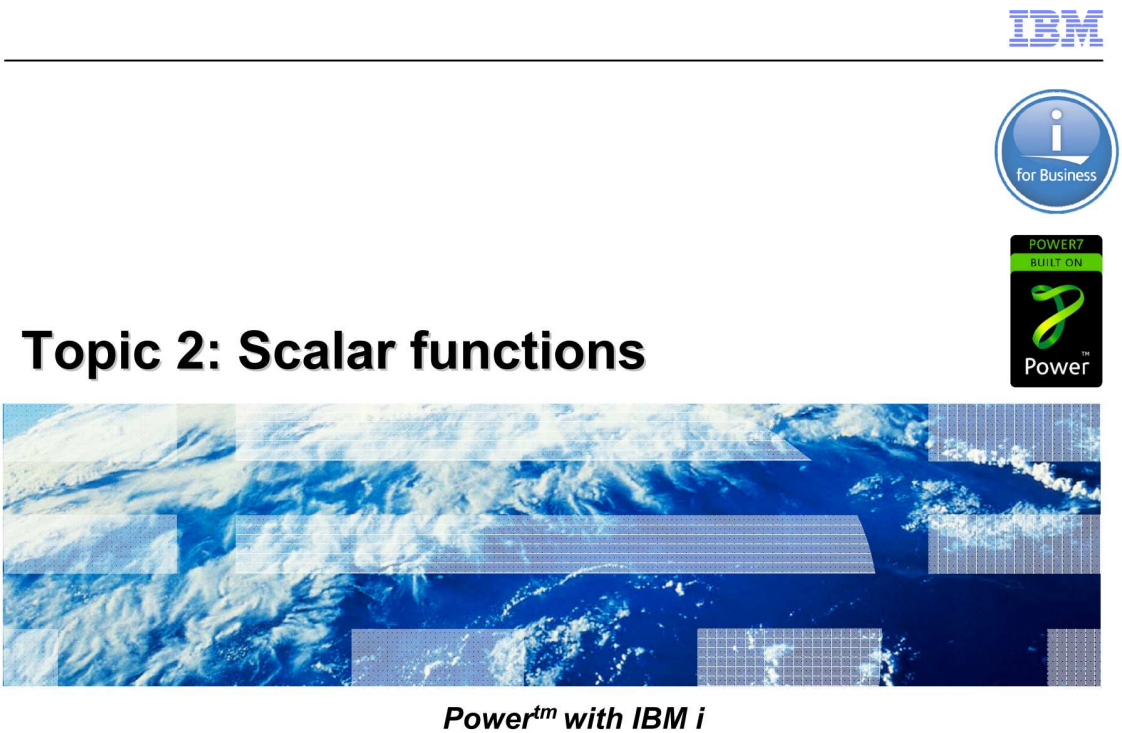


Figure 4-9. Topic 2: Scalar functions OD477.0

Notes:

A scalar function is applied to a single value rather than a set of values like a column function. Let's learn how to use them in SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Scalar functions

IBM i

- Operate on individual values
  - String functions
  - Conversion functions
  - Date and time functions
- Can be nested
- Can use the result of another function as an argument

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-10. Scalar functions OD477.0

Notes:

There are many scalar functions available in SQL. For example, you can:

- Perform date operations such as fetch the values of the current date and the current time
- Determine the absolute value or the sine of a number

In this topic, we will cover some of the most commonly used functions.

A *scalar function* can be used wherever an expression can be used. A column function cannot be used in the same way.

A *scalar function* is applied to a *single value* rather than a *set of values* like a column function.

Therefore, for example, the argument of a scalar function can be another function. However, the restrictions that apply to the use of expressions and column functions also apply when an expression or column function is used within a scalar function.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

For example, the argument of a scalar function can be a column function only if a column function is allowed in the context in which the scalar function is used.

If additional information regarding the existence and the results delivered by scalar functions that are not in this topic is desired, please review the documentation.

References:

- *DB2 for i SQL Reference Manual*, Built-in Functions

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This topic will describe a variety of scalar functions with an emphasis on date functions. At V7R1, there are over one hundred eighty (180+) different scalar functions available in SQL. Since the discussion in class can become very lengthy, we will discuss the most commonly used functions. For any others, the reference document or the Information Center website are comprehensive references.

Transition — First, we will cover the string handling functions.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Substring character columns

IBM i

```
        SELECT  caryear, SUBSTR(carmake, 1, 4),
                SUBSTR(caryear, 3, 2) as YY
          FROM  cars
          WHERE  caryear > 1970

CARYEAR  SUBSTR  YY

 1,985    Chev   85
 1,984    Ferr   84
 1,977    Chev   77
 1,995    Ford   95
 1,971    Mase   71
 1,973    Alfa   73
 1,974    Merc   74
```

- Must be numeric or string data type

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-11. Substring character columns OD477.0

Notes:

The **SUBSTR** or **SUBSTRING** function operates on an expression. The expression must be any built-in numeric or string data type. A numeric argument is automatically cast to a character string before evaluating the function.Three parameters are passed:

- The *expression* can be fixed or variable length.
- The *start value* must be an integer value or variable. It specifies where the substring function is to start.
- The *length value*, which is optional, must be an integer value or an integer variable. If this value is omitted, the substring operation will extract every byte from the start byte to the end of the expression argument.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

The expression must be any built-in numeric or string data type. A numeric argument is automatically cast to a character string before evaluating the function.

© Copyright IBM Corp. 2010, 2013
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CONCAT

IBM i

```
select concat(firstname, lastname) as member_name
       from members

MEMBER_NAME
Bob             Slaney
Ernest          Humphries
Mary            Shrauger
Jim             Moline
Doug            Armstrong
Annette         Jamison
Diane           Williams
Gregory         Simpson
Joyce           Brown
David           Hall
Romaine         Rockefeller
Margaret        Brown
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-12. CONCAT OD477.0

Notes:

The concatenation function, **CONCAT**, combines two or more compatible strings. All operands must be either character or graphic. The result of the expression is a string of the same data type.

Another way to code the example, using ||, is:

```
select firstname || lastname as member_name from members

select firstname concat lastname as member_name from members
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — You could also code:

```
select firstname concat lastname as member_name from members
```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

STRIP

IBM i

```
select strip (empno, leading, '0') as EMP#,
       projno, actno, emptime, emstdate, emendate
       from emp_act

 EMP#  PROJNO  ACTNO EMPTIME  EMSTDATE    EMENDATE
 10    AD3100     10     .50   1982-01-01  1982-07-01
 70    AD3110     10    1.00   1982-01-01  1983-02-01
 230   AD3111     60    1.00   1982-01-01  1982-03-15
 230   AD3111     60    1.00   1982-01-01  1982-03-15
 230   AD3111     60     .50   1982-03-15  1983-04-15
 230   AD3111     70     .50   1982-03-15  1982-10-15
 230   AD3111     80     .50   1982-04-15  1982-10-15

                                                More...
```

- In EMP_ACT table, EMPNO column has leading zeros

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-13. STRIP OD477.0

Notes:

The **STRIP** function removes blanks or another specified character from the end(TRAILING), beginning(LEADING) or BOTH of a string expression.

The first argument must be a string expression.

The second argument, if specified, indicates whether characters are removed from the end or beginning of the string. If the second argument is not specified, then blanks are removed from both the beginning and the end of the string.

The third argument, if specified, is a single-character constant that indicates the SBCS or DBCS character that is to be removed. The character must be enclosed in single quotes.

In our example, we have a field named **EMPNO** that is six bytes in length.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Here is the column layout of the table EMP_ACT that is used in the example:

COLUMN NAME	DATA TYPE	COLUMN LENGTH	SCALE
EMPNO	CHAR	6	
PROJNO	CHAR	6	
ACTNO	BINARY	4	0
EMPTIME	DECIMAL	5	2
EMSTDATE	DATE	10	
EMENDATE	DATE	10	

Remember that:

1. The expression must be a character string.
2. The result of a scalar function can be the argument of another scalar function.

If the value of EMPNO is '00010', the **STRIP** function removes any zeros unless another character is specified as the strip character. You can see that the value has been left aligned if you look at other values in the column.

The **TRIM** function performs the same function as **STRIP** but the order of the arguments is different.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — The column label “STRIP function” is what is usually generated. We replaced it with something better using an AS clause to designate “EMP#” as a heading.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

LTRIM and RTRIM

IBM i

- Trim blanks from the left or right side of a string expression

  RTRIM(expression)
  LTRIM(expression)

- If FIRSTNAME = ***'Robert '***

  SELECT RTRIM(firstname) FROM members

  results in ***'Robert'***

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-14. LTRIM and RTRIM OD477.0

Notes:

The **LTRIM** function removes blanks from the beginning of a string expression. The **RTRIM** function removes blanks from the end of a string expression.

The result of the function is a varying-length string with the same maximum length as the length attribute of the string. The actual length of the result is the length of the expression minus the number of characters removed. If all characters are removed, the result is an empty varying-length string.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

UPPER and TRANSLATE

IBM i

- **TRANSLATE(arg1, to-string, from-string)**

```
select translate(Lastname, 'S', 's') from members
```

Lastname = 'Humphries' TRANSLATE to 'HumphrieS'

- **UPPER(arg1)** converts the case of a column from lowercase or mixed case to uppercase

```
select upper(Lastname) from members
```

Lastname = 'Humphries' UPPER to 'HUMPHRIES'

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-15. UPPER and TRANSLATE OD477.0

Notes:

The **TRANSLATE** function translates one or more characters of the first argument based on the second argument.

The first argument must be a character string. The argument cannot be a graphic string. The data type and length attribute of the result are the same as the first argument.

The second argument is a character string constant of no more than 256 characters. Note that this is sometimes called the *output translation table*.

The **UCASE** and **UPPER** functions are identical to the **TRANSLATE** scalar function with only the first argument specified. These functions convert the character string to upper case.

The converse functions are **LOWER** and **LCASE**. These functions convert the argument to lower case. For the example:

```
select lower(lastname) where lastname='Humphries'
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Do not cover this function. The students may reference it themselves later.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Conversion functions

IBM i

- DECIMAL, FLOAT, INTEGER convert numeric data
- DIGITS returns character representation of numeric value
- HEX returns hex representation of a value

```
select caryear, digits(Caryear),hex(caryear) from
cars

CARYEAR  DIGITS ( CARYEAR )  HEX ( CARYEAR )
 1,967            1967              01967F
 1,965            1965              01965F
 1,985            1985              01985F
 1,965            1965              01965F
 1,984            1984              01984F
 1,977            1977              01977F
 1,971            1971              01971F
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-16. Conversion functions OD477.0

Notes:

These functions perform conversion of numeric data to other forms. **DIGITS**, for example, converts a numeric value to character representation of the absolute value of a number. The argument must be an integer or decimal value.

The **HEX** function returns a hexadecimal representation of a value. The argument can be any value. The result of the function is a character string.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — This function should not be covered in class.

Additional information —

Transition statement — This is one way to “edit” numeric columns. We will see the CAST function at the end of this topic. It is an effective way to convert data also. It has the capability of many conversion functions in one tool.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DECIMAL

IBM i

```
select empno, lastname, job, decimal(salary,6,0)
             from employee

EMPNO   LASTNAME          JOB       DECIMAL
000010  Haas               PRES        52,750
000020  Thompson           MANAGER     41,250
000030  Kwan               MANAGER     38,250
000050  Geyer              MANAGER     40,175
000060  Stern              MANAGER     32,250
000070  Pulaski            MANAGER     36,170
000090  Henderson          MANAGER     29,750
000100  Spenser            MANAGER     26,150
   :       :
```

- **Converts format of numeric column**

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-17. DECIMAL OD477.0

Notes:

DECIMAL converts (or maps) the format of a numeric column. In the example, we are changing the number of decimal places (scale) from two places to zero places.

The **DECIMAL** function returns a packed decimal representation of:

- A number
- A character string representation of a decimal number
- A character string representation of an integer
- A character string representation of a floating-point number

The result of the function is a decimal number with precision of *p* and scale of *s*, where *p* and *s* are the second and third arguments.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Here is the format of the EMPLOYEE table:

COLUMN	DATA TYPE	COLUMN LENGTH	SCALE
EMPNO	CHAR	6	
FIRSTNME	CHAR	12	
MIDINIT	CHAR	1	
LASTNAME	CHAR	15	
WORKDEPT	CHAR	3	
PHONENO	CHAR	4	
HIREDATE	DATE	10	
JOB	CHAR	8	
EDUCLVL	BINARY	4	0
SEX	CHAR	1	
BIRTHDATE	DATE	10	
SALARY	DEC	9	2
BONUS	DEC	9	2
COMM	DEC	9	2

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This function should not be covered in lecture.

Additional information — If the first argument of **DECIMAL** can be null, the result can be null; if the first argument is null, the result is the null value.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

CHAR

IBM i

- Converts:
  - Character to character
  - Integer to character
  - Decimal to character
  - Floating point to character
  - Date/time to character

- CHAR(BIRTHDATE, USA)
  - Original data 1985-04-03
  - becomes 04/03/1985

- CHAR(CARYEAR)
  - Original data 1,967
  - becomes 1967

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-18. CHAR OD477.0

Notes:

The **CHAR** function returns a string representation of:

- A date/time value if the first argument is a date, time, or timestamp.
- A character string if the first argument is any type of character string.
- An integer number if the first argument is a SMALLINT or INTEGER.
- A decimal number if the first argument is a packed or zoned decimal number.
- A double-precision floating-point number if the first argument is a DOUBLE or REAL.

CHAR can present your data in a column in an easier to view pattern. In the visual, CARYEAR has been presented as *1967* (character string) rather than *1,967* (decimal number).

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — A CAST expression can also be used to return a string expression. We will discuss CAST in the final topic of this unit.

Note that CHAR supports only ISO, USA, EUR, and JIS date formats.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

MOD

IBM i

- Determines remainder
  - Arguments must be numeric
  - Divides the first argument by the second
  - Checks for zero divisor first
    - If zero, result is +++++++++
- MOD(200,3)
  - = 2
- MOD(12,MONTH(CURDATE()))
  - = 2 if MONTH = 10

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-19. MOD OD477.0

Notes:

The **MOD** function divides the first argument by the second argument and returns the remainder.

The formula used to calculate the remainder is

MOD(x,y) = x - (x/y) * y

where x/y is the truncated integer result of the division.

The arguments must be numbers. The second argument cannot be zero. You should check the value of the divisor expression for zero before you perform this function. We will show you how to code conditional expressions that will handle this kind of situation.

If an argument can be null, the result can be null; if an argument is null, the result is the null value.

The attributes of the result are determined as follows:

- If both arguments are integers with zero scale, the data type of the result is large integer.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- If one argument is an integer with zero scale and the other is decimal, the result is decimal with the same precision and scale as the decimal argument.
- If both arguments are decimal or integer with scale numbers, the result is decimal.
- If either argument is floating point, the data type of the result is double-precision floating point.

This visual also shows another scalar function, **CURDATE()**. This function will retrieve today's date from the system. There are also other similar scalars that we will look at briefly.

References:

- *DB2 for i SQL Reference*, Built-in Functions

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

MIN and MAX

IBM i

- **Argument data types *must* be compatible**
  - Must pass two or more arguments
  - MIN returns the minimum value of a set of arguments
  - MAX returns the maximum value of a set of arguments

- **If the values of NUMx are 10, 14, 2, and 6**
  - **MIN(NUM1, NUM2, NUM3, NUM4)** = 2
  - **MAX(NUM1, NUM2, NUM3, NUM4)** = 14

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-20. MIN and MAX OD477.0

Notes:

The **MIN** scalar function returns the minimum value in a set of values.

The arguments must be compatible. Character-string arguments are compatible with date and time values, but are not compatible with graphic strings. All but the first argument may be parameter markers. There must be two or more arguments.

The result of the function is the smallest argument value. The result can be null if at least one argument can be null; the result is the null value if one of the arguments is null. The selected arguments are converted, if necessary, to the attributes of the result.

The **MAX** scalar function returns the maximum value in a set of values.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — There is no practical limit to the number of arguments for a MIN or MAX scalar. You would be limited by the SQL maximum statement limit and the internal object limit. Both are large enough that we need not worry.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IFNULL, VALUE, and COALESCE

IBM i

- Used to check for null value
- Data types must be compatible
- **COALESCE / VALUE**
  - Returns the first non-null value of two or more arguments
- **IFNULL**
  - Returns the first non-null value of two arguments

Example: If the value of column MANAGER is null,
return 'No Manager'

IFNULL(MANAGER, 'No Manager')

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-21. IFNULL, VALUE, and COALESCE OD477.0

Notes:

The **COALESCE** and **VALUE** functions return the first argument that is not null in a series of arguments.

The arguments must be compatible. Character-string arguments are compatible with date and time values.

The **IFNULL** function is identical to the **COALESCE** scalar function but can only have two arguments.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date and time

IBM i

- SQL deals with the external form of the date and time
- Sets the date and time format desired
  - Can convert with CHAR
- Can extract the day, month, and year from the date
- Can perform addition and subtraction (durations)

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-22. Date and time OD477.0

Notes:

SQL offers very comprehensive support for date and time. Components of values can be extracted using scalar functions.

Remember that we are “isolated” from the internal format of date and time values. What we see is determined by the external format that can be set in session defaults of STRSQL, for example, or by using the scalar function **CHAR**.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date/time scalar functions (1 of 3)

IBM i

- DAYS
  - Number of days since December 31, 0000
- DATE, TIME
  - Extract the date and time from the timestamp
- CURDATE, CURTIME
  - Retrieve the date or time of day
  - Other forms can also be used
- Calculate from date or timestamp
  - DAYOFWEEK (Sunday = 1, Saturday = 7)
  - DAYOFMONTH = DAY
  - DAYOFYEAR

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-23. Date/time scalar functions (1 of 3) OD477.0

Notes:

These scalar functions are used to extract the desired portion of a date, time or timestamp column. The result is an integer data type.

CURDATE and CURTIME return a date or time, respectively, when the SQL statement that uses them is executed. They can be arguments for other date and time functions such as DAYOFWEEK, DAYOFMONTH and DAYOFYEAR.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date/time scalar functions (2 of 3)

IBM i

- Extract portions of the date, time, or timestamp. The result is an integer:
  - DAY, MONTH, YEAR
  - HOUR, MINUTE, SECOND
  - MICROSECOND
  - EXTRACT
- DAYNAME
  - Returns the name of the day of the week
- MONTHNAME
  - Returns the name of the month

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-24. Date/time scalar functions (2 of 3) OD477.0

Notes:

These are other scalars that you can use in date and time calculations and presentation. One of the ones that is most interesting is **DAYOFWEEK** because in RPG IV, there is currently no equivalent built-in-function and you must code a routine to handle this.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — There are also some special registers that provide date information that will be discussed at the end of this topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date/time scalar functions (3 of 3)

IBM i

- LAST_DAY
  - Returns a date representing the last day of the month for an argument date
- NEXT_DAY
  - Returns a date that represents the first weekday named by an expression that is later than the argument date

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-25. Date/time scalar functions (3 of 3) OD477.0

Notes:

The LAST_DAY function returns a date that represents that last day of the month for an argument date value.

The NEXT_DAY function will return a date from an argument date value that is the first weekday named by a requested expression that is later than the original date value. We'll show you an example on the next visual.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — An example of the NEXT_DAY function follows on the next visual.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Date function samples

IBM i

- Original HIREDATE = 1965-01-01
  - MONTH(HIREDATE) = 1
  - DAY(HIREDATE) = 1
  - DAYS(HIREDATE) = 717,337
  - YEAR(HIREDATE) = 1,965
  - HIREDATE + 1 YEAR = 1966-01-01
  - HIREDATE + 1 MONTH = 1965-02-01
  - HIREDATE + 30 DAYS = 1965-01-31
  - NEXT_DAY(HIREDATE, 'TUESDAY') = 1965-01-05
  - DAYOFWEEK(HIREDATE) = 6
  - DAYNAME(HIREDATE) = Friday
  - NONTHNAME(HIREDATE) = January

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-26. Date function samples OD477.0

Notes:

The above examples help illustrate the different uses of labeled durations. MONTH, DAY and YEAR are all used to extract the appropriate unit from a date value (HIREDATE). The durations YEAR, MONTH and DAYS are also used in date calculations. Note how DAYS also has the other function of returning a numeric assignment to a date value.

The NEXT_DAY function returns a date or timestamp value that represents the first weekday, named by string-expression, that is later than the date expression.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Use the examples to guide the students through the use of labeled durations and their distinct differences.

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

EXTRACT and MONTHNAME

IBM i

```
select empno, extract(year from hiredate) as YEAR,
cast(monthname(hiredate) as char(20)) as month,
extract(day from hiredate) as day from Employee

 EMPNO          YEAR    MONTH                          DAY
000010          1.965   January                          1
000020          1.973   October                         10
000030          1.975   April                            5
000050          1.949   August                          17
000060          1.973   September                       14
000070          1.980   September                       30
000090          1.970   August                          15
000100          1.980   June                            19
000110          1.958   May                             16
000120          1.963   December                         5
000130          1.971   July                            28
000140          1.976   December                        15
000150          1.972   February                        12
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-27. EXTRACT and MONTHNAME OD477.0

Notes:

EXTRACT returns the specified portion of a datetime value. You can specify:

select empno, EXTRACT(YEAR from hiredate), EXTRACT(MONTH from hiredate), EXTRACT(DAY from hiredate) from Employee

to obtain year, month and day of a specified date.

MONTHNAME returns the month name from a datetime value.

DAYNAME returns the name of the day of the week (Sunday, Monday.....).

Next we will see an example of how the **YEAR** scalar can be used to assist in duration calculations between dates.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

YEAR

IBM i

```
select firstnme, lastname, birthdate,
       year(curdate() - birthdate) as Age
       from employee

FIRSTNME      LASTNAME         BIRTHDATE             AGE

Christine     Haas             1933-08-24             79
Michael       Thompson         1948-02-02             65
Sally         Kwan             1941-05-11             71
John          Geyer            1925-09-15             87
Irving        Stern            1945-07-07             67
Eva           Pulaski          1953-05-26             59
Eileen        Henderson        1941-05-15             71
Theodore      Spenser          1956-12-18             56
Vicenzo       Lucchesi         1929-11-05             83
Sean          O Connell        1942-10-18             70
```

When CURDATE = 2013-03-19

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-28. YEAR OD477.0

Notes:

YEAR extracts the year component of a date. In the example, we use YEAR to calculate the age of each employee. Note the use of **CURDATE()** to determine today's date.

DAYS converts a date to an integer value using December 31, 0000 as a basis. Once you have calculated **DAYS**, you can easily perform duration calculations.

Further, you must be careful to get the result you want. Consider this slight modification to the example in the visual:

```
select firstnme, lastname, birthdate,
   year(curdate()) - year(birthdate) as Age
   from employee
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

```
FIRSTNME     LASTNAME        BIRTHDATE            AGE

Christine    Haas            1933-08-24            80
Michael      Thompson        1948-02-02            65
Sally        Kwan            1941-05-11            72
John         Geyer           1925-09-15            88
Irving       Stern           1945-07-07            68
Eva          Pulaski         1953-05-26            60
Eileen       Henderson       1941-05-15            72
Theodore     Spenser         1956-12-18            57
Vicenzo      Lucchesi        1929-11-05            84
Sean         O Connell       1942-10-18            71
```

The result above is different because we are not considering partial years and we are always rounding before we calculate the age.

In the example below, we are using our legacy APINV_PF file that was defined using DDS before null support was available. Notice the use of an expression to determine the number of days between the date that payment was due and the date that payment was made. The year 9999 is used in this table to mean that the invoice has not been paid. No durations are calculated when the year is 9999.

```
select char(ponbr) as PO_NUM , apdatepaid, apduedate,
       days(apdatepaid)-days(apduedate) as days_outstanding
       from apinv_pf
       where not year(apdatepaid) = 9999
```

```
PO_NUM     Date        Due         DAYS_OUTSTANDING
           Paid        Date
           YYYYMMDD    YYYYMMDD
300071     1996-04-11  1996-04-11                 0
300073     1998-04-11  1998-04-11                 0
300077     1998-03-15  1998-03-01                14
300083     1998-09-30  1998-09-30                 0
300084     1998-03-15  1998-02-14                29
300097     1998-07-31  1998-07-26                 5
```

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Special registers

IBM i

- Contain system placed value
- Can be accessed and retrieved in SQL
  - Current Date or Current_Date = system date
  - Current Time or Current_Time = time of day
  - Current Timestamp or current_timestamp = timestamp
  - Current TimeZone or Current_timezone = timezone
  - SESSION_USER = current session user ID

```
 select firstnme, lastname, current timestamp
       from employee
FIRSTNME       LASTNAME          CURRENT TIMESTAMP
Christine      Haas              2001-02-14-15.06.17.305536
Michael        Thompson          2001-02-14-15.06.17.305536
Sally          Kwan              2001-02-14-15.06.17.305536
John           Geyer             2001-02-14-15.06.17.305536
Irving         Stern             2001-02-14-15.06.17.305536
Eva            Pulaski           2001-02-14-15.06.17.305536
Eileen         Henderson         2001-02-14-15.06.17.305536
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-29. Special registers OD477.0

Notes:

Special registers contain system information such as date, time, timestamp data that can be retrieved in SQL. The SESSION_USER special register specifies the run-time user ID at the current server. The data type for this special register is VARCHAR(128).There are others available in addition to these.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

4.3. CASE and CAST

Instructor topic introduction

What students will learn — Introduce the CASE structure. CASE can be used in most places that an expression is required. Highlight the searched versus simple WHEN specification and the fact that the ELSE clause is optional.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

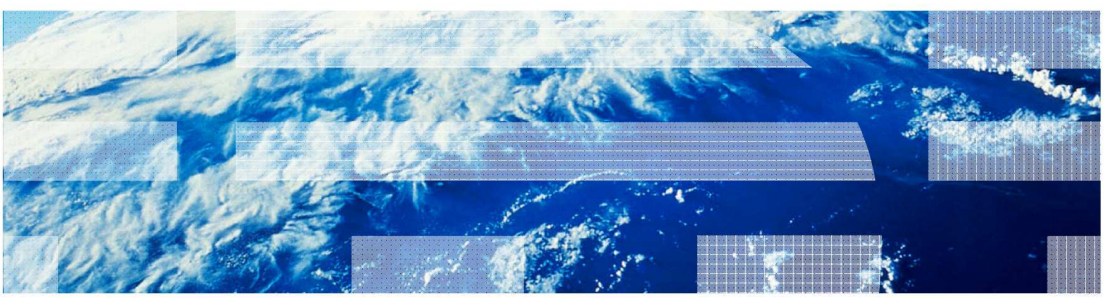


Figure 4-30. Topic 3: CASE and CAST OD477.0

Notes:

Let's learn how to code IF-THEN-ELSE logic in SQL with CASE expressions. We'll also look at how CAST can perform the same functions as many of the other conversion functions.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CASE expressions (1 of 3)

IBM i

```
SELECT empno, lastname,
     CASE
       when salary<25000 then 'Low'
            when salary<40000 then 'Average'
            when salary>=40000 then 'High'
     end as Salary_Classification
  from employee

EMPNO   LASTNAME          SALARY_CLASSIFICATION

000010  Haas              High
000020  Thompson          High
000030  Kwan              Average
000050  Geyer             High
000060  Stern             Average
000070  Pulaski           Average
000090  Henderson         Average
000100  Spenser           Average
000110  Lucchesi          High
000120  O Connell         Average

select empno, lastname,
     case
         when salary<25000 then 'Low'
         when salary<40000 then 'Average'
         else  'High'
     end as Salary_Classification
  from employee
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-31. CASE expressions (1 of 3) OD477.0

Notes:

CASE expressions allow you to code IF-THEN-ELSE logic in SQL to make logical decisions.

DB2 for IBM i evaluates the conditions in top to bottom order. The value returned from the CASE will be the result from the first true condition. If no CASE condition is satisfied, the ELSE value is used. If no ELSE is coded, the value returned is NULL by default.

CASE expressions can appear in various places in an SQL statement, including the SELECT and its WHERE, GROUP BY, and HAVING clauses. CASE can also be used in IN and VALUES clauses.

The examples illustrate the use of a CASE in the SELECT clause. The first CASE expression returns a salary classification for the employees rather than the salary itself. The salary classification is based on the range of the salary. If the salary is less than 25000, the salary classification is *LOW*. It is *AVERAGE* if the salary is equal to or higher than 25000, but is lower than 40000. Salaries of 40000 and higher are considered *High*.

In the example, we could code the last *when salary>=40000 then High* as *else High.*

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The appropriate column of the result table is called SALARY_CLASS. using the AS keyword.

This example, which has no expression between the CASE keyword and the first THEN keyword, is called a *searched WHEN clause*.

References:

DB2 for i SQL Reference, Language Elements

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CASE expressions (2 of 3)

IBM i

```
select empno, lastname,
       case
           when salary<25000 then 'Low'
           when salary<40000 then 'Average'
           else 'High'
       end as Salary_Classification,
       case substr(workdept,1,1)
           when 'A' then 'Administration'
           when 'C' then 'Customer Service'
           when 'D' then 'Product Development'
       end as AREA_TYPE
       from employee


EMPNO   LASTNAME         SALARY_CLASSIFICATION       AREA_TYPE

000010  Haas             High                        Administration
000020  Thompson         High                        -
000030  Kwan             Average                     Customer Service
000050  Geyer            High                        -
000060  Stern            Average                     Product Development
000070  Pulaski          Average                     Product Development
000090  Henderson        Average                     -
000100  Spenser          Average                     -
000110  Lucchesi         High                        Administration
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-32. CASE expressions (2 of 3) OD477.0

Notes:

This example, where we have added a second CASE expression has an expression between the CASE keyword and the first THEN keyword, is called a *simple WHEN clause*. The expression uses the **SUBSTR** scalar function to determine the first character of the WORKDEPT column and the WHEN clauses are evaluated for each expected value of that character.

If the value of the substring scalar is:

- *A*, *ADMINISTRATION* is displayed in the result.
- *C*, *CUSTOMER SERVICE* is displayed.
- *D*, *DEVELOPMENT* is displayed.

Any other value, null is displayed. Remember that null is displayed as a “-” character.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CASE expressions (3 of 3)

IBM i

```
1  SELECT book_title,
          CASE book_status
               WHEN 1 THEN 'Available'
               WHEN 2 THEN 'Checked Out'
               WHEN 3 THEN 'Reserved'
               ELSE 'Unknown'
          END
          FROM book_catalog

2  SELECT license,
          CASE type
               WHEN 'Car' THEN 0.05 * weight
               WHEN 'Truck' THEN 75 * nbr_wheels
               WHEN 'Motorcycle' THEN 35.00
           END AS License_fee FROM vehicles

3  UPDATE properties
          SET taxrate =
          CASE
              WHEN property_area < 10000 THEN 0.05
              WHEN property_area < 20000 THEN 0.07
              ELSE 0.10
          END
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-33. CASE expressions (3 of 3) OD477.0

Notes:

These examples illustrate several ways in which you can use CASE:

1. Based on the value of **book_status** CASE will return a value of Available, Checked Out, Reserved, or Unknown.
2. This expression uses the vehicle type in a CASE expression to determine the fee to be levied for the license plate.
3. This UPDATE statement will set a value of **taxrate** in each row of the table based on the area of the property to be taxed.

Other examples of applications using CAST:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Debit and credits are stored a positive numbers and the program multiplies by -1 for the debits:

```
CASE
WHEN TRNTYP = 'DN'
THEN AMOUNT *(-1)
WHEN TRNTYP = 'IN'
THEN AMOUNT
END
```

- YTD GL figures:

```
Case
WHEN MONTH (CURRENT DATE) = 1
JANFIGURES
WHEN MONTH (CURRENT DATE) = 2
JANFIGURES + FEBFIGURES
WHEN MONTH (CURRENT DATE) = 3
JANFIGURES + FEBFIGURES + MARFIGURES
and so forth.
```

- GL figures across the page:

```
Case
WHEN TRNMONTH  = 1
JANFIGURES
End,
Case
WHEN TRNMONTH  = 2
 FEBFIGURES
End,

and so forth.
```


© Copyright IBM Corp. 2010, 2013
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Emphasize the flexibility of CAST. CAST can perform the same function as many of the other conversion functions.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CAST expressions

IBM i

- Changes values from one data type to another

Source data type	Target data type
integer, decimal, numeric, float	integer, decimal, numeric, float, char, varchar
char, varchar	char, varchar, UCS-2 graphic, UCS-2 vargraphic, integer, decimal, numeric, float, date, time, timestamp
graphic, vargraphic	graphic, vargraphic char and varchar if source is USC-2
date, time	char, varchar, timestamp
timestamp	date, time, timestamp, char, varchar

- **CAST (expression AS data type)**
  - ...CAST(charcolumn AS INTEGER) > 100...
  - ...CAST(name||address AS CHAR(80))...

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-34. CAST expressions OD477.0

Notes:

The **CAST** specification returns the cast operand (the first operand) cast to the type specified by the data type.

For example, assume that CUSCRD is a column of data type DECIMAL(11,2) in a table, CUSTOMER. The following statement changes the column type of CUSCRD from DECIMAL to INTEGER in the result set:

```
SELECT CUSNBR, CUSNAM, CAST(CUSCRD AS INTEGER) Customer_Credit
       FROM CUSTOMER
```

Assume that an application is only interested in the integer portion of the SALARY column (defined as DECIMAL(9,2)) from the EMPLOYEE table. The following CAST specification will convert the SALARY column to INTEGER.

```
SELECT EMPNO, CAST(SALARY AS INTEGER) FROM EMPLOYEE
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

You might think that CAST is basically the same as the scalar functions. There are advantages to using CAST over the scalars that perform similar conversions:

1. You only have to remember a single scalar function name for conversion of column data types.
2. CAST allows you to convert between any of the data types. However, the scalars do not provide all of the conversions provided by CAST. You might have to nest CASTs in order to obtain the desired conversion, but, it can be done.
3. You can override the SQL result defaults. For example, AVG of Decimal column always returns a decimal result. If your application expects INT, you simply code:

   CAST(AVG(deccol1) AS INTEGER)

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — Assign the checkpoint but ask the students to perform the exercise first and complete the checkpoint after the lab. They may need to continue the lab for functions while you lecture the database maintenance unit at the end of the course.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

CAST examples

IBM i

```
select lastname as Last_name,
       cast(salary+comm+bonus as char(10))|| 'USD'
       from employee

LAST_NAME        String Expression

Haas               57970.00  USD
Thompson           45350.00  USD
Kwan               42110.00  USD
Geyer              44189.00  USD
Stern              35330.00  USD
Pulaski            39763.00  USD
Henderson          32740.00  USD
:
:
:
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-35. CAST examples OD477.0

Notes:

In this example, we are calculating the total income of each employee and then converting it to a character data type of 10 positions. We are also concatenating “USD” to the resulting value in the each row.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U03** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U03** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U03** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint (1 of 2)

IBM i

1. True or False: The MIN function is identical to the MINUTE function.

2. True or False: A COLUMN function can be an argument of a SCALAR function.

3. The EMPLOYEE table includes a column, EDUCLVL, that indicates the employee's level of education. The number of years of education is used in the EMPLOYEE table to give the education level. A CASE expression can be used to group these and to show the level of education. If the value of EDUCLVL is 22, what will be returned by the following CASE expression?

```
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
  CASE
  WHEN EDUCLVL < 15 THEN 'High School'
  WHEN EDUCLVL < 19 THEN 'University Graduate'
  WHEN EDUCLVL < 23 THEN 'Masters Degree'
  ELSE 'PhD'
  END
FROM EMPLOYEE
```

   a. High School
   b. University Graduate
   c. Masters Degree
   d. PhD
   e. All of the above

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-36. Checkpoint (1 of 2) OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions (1 of 2)

IBM i

1. True or False. The MIN function is identical to the MINUTE function.
   The answer is False.

2. True or False. A COLUMN function can be an argument of a SCALAR function.
   The answer is True.

3. The EMPLOYEE table includes a column, EDUCLVL, that indicates the employee's level of education. The number of years of education is used in the EMPLOYEE table to give the education level. A CASE expression can be used to group these and to show the level of education. If the value of EDUCLVL is 22, what will be returned by the following CASE expression?

```
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
  CASE
  WHEN EDUCLVL < 15 THEN 'High School'
  WHEN EDUCLVL < 19 THEN 'University Graduate'
  WHEN EDUCLVL < 23 THEN 'Masters Degree'
  ELSE 'PhD'
  END
FROM EMPLOYEE
```

   a. High School
   b. University Graduate
   c. Masters Degree
   d. PhD
   e. All of the above.

   The answer is Masters Degree.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint (2 of 2)

IBM i

4. True or False: The function NOW returns a timestamp value.

5. True or False: The CAST function can perform the same function as many of the other conversion functions.

6. True or False: CASE expressions are used in the SELECT statement WHERE, GROUP BY and HAVING clauses.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-37. Checkpoint (2 of 2) OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions (2 of 2)

IBM i

4. True or False: The function NOW returns a timestamp value.
   The answer is True.

5. True or False: The CAST function can perform the same function as many of the other conversion functions.
   The answer is True.

6. True or False: CASE expressions are used in the SELECT statement WHERE, GROUP BY and HAVING clauses.
   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Using SQL functions*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-38. Exercise OD477.0

Notes:

Perform the workstation exercise *Using SQL functions*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe the purpose of column and scalar functions
- Use column functions in SQL queries
- Use scalar functions in SQL queries
- Code CASE and CAST expressions

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 4-39. Unit summary OD477.0

Notes:

While we are not discussing them in this class, you may want to investigate how to use user defined functions (UDFs) in the *SQL Reference Manual*.

These functions allow users to extend the function of the database manager by adding their own or third party vendor function definitions.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 5. JOIN and UNION

Estimated time

01:35

What this unit is about

This unit describes the advanced features of SQL focusing on the use of the JOIN statement and the UNION statement.

What you should be able to do

After completing this unit, you should be able to:

- List five types of JOINs supported on the System i
- Use JOIN to retrieve data from multiple tables or views
- Use UNION to retrieve data from multiple tables or views
- Use EXCEPT or INTERSECT to retrieve data from multiple tables or views
- Use correlation names

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercise

  The students perform a workstation exercise to test their understanding of JOIN and UNION.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- List five types of JOINs supported on the System i
- Use JOIN to retrieve data from multiple tables or views
- Use UNION to retrieve data from multiple tables or views
- Use EXCEPT or INTERSECT to retrieve data from multiple tables or views
- Use correlation names

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-1. Unit objectives OD477.0

Notes:

This unit focuses on how to combine data from more than one table.

JOIN versus UNION

Data can be combined using a JOIN or a UNION. A JOIN combines tables based on column values. A UNION clause combines tables based on rows.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

5.1. Using JOIN

Instructor topic introduction

What students will learn — Students learn how to use JOIN and UNION in SQL queries.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Figure 5-2. Topic 1: Using JOIN OD477.0

Notes:

JOIN allow you to select data from multiple tables as long as the tables share a common key that relates them to one another.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Combining data using JOIN

IBM i

- Problem: List employees and department name
- Given
  - Corporate database
  - Data stored in EMPLOYEE and DEPARTMENT tables

EMPLOYEE

Lastname	Workdept
Haas	**A00**
Thompson	B01
Kwan	C01
Geyer	E01
Stern	D11
Pulaski	D21
Henderson	E11
Spenser	E21
Lucchesi	A00

DEPARTMENT

Deptno	**Deptname**
A00	SPIFFY Computer Service Div.
B01	**Planning**
C01	Information **Center**
D01	Development Center
D11	**Manufacturing Systems**
D21	Administration Systems
E01	Support **Services**
E11	Operations
E21	Software **Support**

Haas	A00	SPIFFY **Computer Service Div.**
Lucchesi	A00	SPIFFY **Computer Service Div.**
O Connell	A00	**SPIFFY Computer Service Div.**
Thompson	B01	Planning
Kwan	**C01**	Information Center
Quintana	C01	Information Center
Nicholls	C01	Information **Center**
Stern	D11	Manufacturing **Systems**
Adamson	D11	Manufacturing Systems
Pianka	D11	Manufacturing **Systems**
Yoshimura	D11	Manufacturing Systems
Scoutten	D11	Manufacturing Systems

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-3. Combining data using JOIN OD477.0

Notes:

The ability to select data from multiple tables is one of the most powerful features of SQL. A relational database is broken up into smaller tables for simplicity and manageability. These smaller tables are normalized to reduce the amount of redundant information stored in the databases. Each table should only contain the required information plus a common key which relates it to the other tables. The JOIN function makes use of this common key to select data from two or more tables.

We have two normalized tables, EMPLOYEE and DEPARTMENT. But, we want to produce the result shown in the visual, a listing of each employee showing his or her DEPARTMENT. What should we do? Using a JOIN and the column WORKDEPT to match to DEPTNO is the solution to creating the result set we desire.

The data shown in the result table can be obtained by coding a SELECT that uses an INNER JOIN.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Spend some time showing how the result table is based upon data values in the two base tables, EMPLOYEE and DEPARTMENT.

The column used to join the tables is WORKDEPT of EMPLOYEE and DEPTNO of DEPARTMENT.

Transition — Before we show the students how to code a JOIN on the IBM i, let's describe the various types of JOIN in relational database terms.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Types of JOINs

IBM i

- **INNER**
  - Table on left joined to all matching rows of table on right of JOIN
- **LEFT OUTER**
  - Rows of table on left that match rows of table on right
  - Rows of table on left that do not match rows of table on right
- **RIGHT OUTER**
  - Rows of table on right that match rows of table on left
  - Rows of table on right that do not match rows of table on left
- **EXCEPTION Left or Right**
  - Rows of table on left that do not match rows of table on right
  - Rows of table on right that do not match rows of table on left
- **CROSS**
  - Each row of table on left is concatenated with EVERY row of table on the right

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-4. Types of JOINs OD477.0

Notes:

A JOIN always involves two or more tables. Five forms of JOIN are supported on the IBM i. We briefly describe them below in terms of *the table on the left* and *the table on the right*. The table on the left is the first table in the join operation.

The following forms of JOIN are supported on the IBM i:

INNER JOIN or simply **JOIN**

Specifies that each row in the table to the left will be joined with one or more rows in the table to the right using the join condition. Any rows in the table to the left that do not have a corresponding row in the table to the right will not be included in the result table.

LEFT OUTER JOIN or simply **LEFT JOIN**

Specifies that each row in the table to the left will be joined with one or more rows in the table to the right using the join condition. Any row in the table to the left that does not have a corresponding row in the table to the right will return the null value for each column in the table to the right.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

In other words, the result set will include rows from the table on the left table that have no matching values to the table on the right based on the join condition. An example would be employees who are not active in any projects.

RIGHT OUTER JOIN or simply **RIGHT JOIN**

Specifies that each row in the table to the right will be joined with one or more rows in the table to the left using the join condition. Any row in the table to the right that does not have a corresponding row in the table to the left will return the null value for each column in the table to the left.

The result set will include rows from the table on the right that have no matching values to the table on the left based on the join condition.

An example might be all projects without any employees assigned.

LEFT OR RIGHT EXCEPTION JOIN

Specifies that only the rows in the table to the left that have no corresponding rows in the table to the right using the join condition will be returned. The null value will be returned for each column in every row for the table to the right. There is a right exception join, too, that works just like a left exception join but with the tables reversed.

CROSS JOIN

Specifies that the result table will contain a row for each row from the table to the left concatenated with each row from the table to the right. In other words, the first row of the table to the left is concatenated with each and every row of the table to the right. The second row of the table to the left is concatenated with each and every row of the table to the right.

This process is repeated for every row of the table to the left.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce the concept of JOIN. We show examples of various JOINs in the visuals that follow.

Details — We want the students to be aware that:

- There are five types of JOINs supported on the IBM i.
- A JOIN involves two (or more) tables or views.
- There must be a column that is common to each table or view.
- The common column might be named differently, but it must have the same data type, length, and so forth.

Transition — Next we show some examples of using an INNER JOIN. This type of JOIN serves as an illustration of the two ways that JOINs can be coded in SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

INNER JOIN

IBM i

Generate a result set of LASTNAME and WORKDEPT from EMPLOYEE plus DEPTNAME from DEPARTMENT

```
select lastname, workdept, deptname
       from employee inner join department
       on workdept=deptno

LASTNAME          WORKDEPT  DEPTNAME
Haas                A00     SPIFFY Computer Service Div.
Thompson            B01     Planning
Kwan                C01     Information Center
Geyer               E01     Support Services
Stern               D11     Manufacturing Systems
Pulaski             D21     Administration Systems
Henderson           E11     Operations
Spenser             E21     Software Support
:                                   :

select lastname, workdept, deptname
       from employee, department
       where workdept=deptno
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-5. INNER JOIN OD477.0

Notes:

Here is the solution to the problem we stated several visuals ago.

The join of these two tables is based on equal values of a common column, department number, which is a column stored in both of the tables.

In the WHERE clause or in the ON portion of the FROM clause, you must specify the columns whose values are to be compared.

When retrieving column values from two tables, you have to ensure that the values are related to each other. Note that in this example, the column names are different in the two tables.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Here is a subset of the data from the EMPLOYEE table, which is ordered by WORKDEPT:

```
LASTNAME          WORKDEPT

Haas                A00
Lucchesi            A00
O Connell           A00
Thompson            B01
Kwan                C01
Quintana            C01
Nicholls            C01
Stern               D11
Adamson             D11
Pianka              D11
Yoshimura           D11
Scoutten            D11
Walker              D11
Brown               D11
:              :
```

Following is a subset of the rows from the DEPARTMENT table:

```
DEPTNO  DEPTNAME

 A00    SPIFFY Computer Service Div.
 B01    Planning
 C01    Information Center
 D01    Development Center
 D11    Manufacturing Systems
 D21    Administration Systems
 E01    Support Services
 E11    Operations
 E21    Software Support
```

In our example, we want to use the DEPTNO column to place the department number column from the DEPARTMENT table in the result set. The column for department has a different name in each of the two tables (workdept=deptno). If they were named the same, they would have to be written using the form:

```
tablename.columnname
```

The column name would be fully qualified and not ambiguous.

Notice the two forms of JOIN shown in the visual. The two forms of the INNER JOIN or, simply JOIN, are identical.

ON is used to specify a *join condition* as part of the JOIN portion of the SELECT. In our join condition examples, we often use a simple equal (=) condition, but almost any valid expression can be used to create your join condition.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review the SQL Reference for additional information.

References:

- *DB2 for i SQL Reference*, Queries

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Make sure you point out that because the department number column is named differently in the two tables, we do not have to fully qualify the name. However, if the columns had the same name, we would have to fully qualify each name in the JOIN.

Run these select statements to demonstrate the INNER JOIN:

1. From the EMPLOYEE table:

```
select lastname, workdept
       from employee;
```

2. From the DEPARTMENT table:

```
select deptno, deptname
       from department;
```

3. Join the two tables to produce the result set:

```
select lastname, deptno, deptname
       from employee
       inner join department
       on workdept=deptno;
```

Notice that the result set is not sequenced in any order. What would we have to add in order to obtain the set in the Department Number sequence? We would have to add an Order by deptno clause.

Transition — Now lead a class exercise to code the SQL statement to produce the result table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

INNER JOIN with USING clause

IBM i

Generate a result set of LASTNAME, WORKDEPT from EMPLOYEE plus PROJNO and EMPTIME from EMP_ACT

```
select lastname, workdept, projno, emptime
       from employee inner join emp_act
       using(empno)

LASTNAME          WORKDEPT  PROJNO    EMPTIME
Haas                A00     AD3100        .50
Haas                A00     MA2100        .50
Haas                A00     MA2110       1.00
Thompson            B01     PL2100       1.00
Kwan                C01     IF1000        .50
Kwan                C01     IF2000        .50
Geyer               E01     OP1000        .25
Geyer               E01     OP2000        .75
Pulaski             D21     AD3110       1.00
:                                          :
select lastname, workdept, projno, emptime
       from employee, emp_act
       where employee.empno = emp_act.empno
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-6. INNER JOIN with USING clause OD477.0

Notes:

The join of these two tables is based on equal values of a common column, employee number, which is a column stored in both of the tables. The column even has the same name in both tables.

In the **USING** clause portion of the **FROM** clause, you can use shorthand notation as a way of defining the join conditions. The **USING** clause is equivalent to a join condition where each column specified from the left table is compared to a column with the same name in the right table. In this example, EMPNO is the name of the column in both the EMPLOYEE table and the EMP_ACT table.

Since they are named the same, they would have to be written using the form:

```
tablename.columnname
```

The column name would be fully qualified and not ambiguous.

Notice the alternate form of JOIN shown in the visual. It produces the identical result set. WHERE is used to specify a *join condition* as part of the JOIN portion of the SELECT.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

In it, our join condition is fully qualified and uses a simple equal (=) condition, to create the join.

Review the SQL Reference for additional information.

References:

DB2 for i SQL Reference, Queries

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — When joining with columns that have the same name in multiple tables, you can use shorthand notion of the **USING** clause as a way of defining the join conditions.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

JOIN exercise

IBM i

Problem: Code the SQL statement to list all cars owned by each car club member in the following format.

```
LASTNAME        FIRSTNAME   CARMAKE          CARMODEL

Armstrong       Doug        Chevrolet        Corvette
Brown           Joyce       Dodge            Superbee
Hall            David       Plymouth         Roadrunner
Humphries       Ernest      Ford             Mustang
Humphries       Ernest      Ford             Thunderbird
Jamison         Annette     Ferrari          308
Jamison         Annette     Alfa Romeo       Targa
Shrauger        Mary        Chevrolet        Corvette
Shrauger        Mary        Plymouth         Hemi Superbird
Simpson         Gregory     Chevrolet        Chevelle SS454
   :               :           :                  :
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-7. JOIN exercise OD477.0

Notes:

We use the CARCLUB tables for this exercise.

Given the following column descriptions for the MEMBERS and CARS tables, you should be able to code a solution.

MEMBERS:

COLUMN	DATA TYPE	FIELD LENGTH	SCALE
MEMBERNUM	DECIMAL	3	0
LASTNAME	CHAR	25	0
FIRSTNAME	CHAR	15	
DUESAMOUNT	DECIMAL	5	2
STREET	CHAR	25	
CITY	CHAR	15	
STATE	CHAR	2	
ZIP	DECIMAL	5	0

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CARS:

```
COLUMN      DATA            FIELD
            TYPE        LENGTH SCALE

CARNUM      DECIMAL          3  0
MEMBERNUM   DECIMAL          3  0
CARMAKE     CHAR            15
CARMODEL    CHAR            15
CARYEAR     DECIMAL          4  0
```

If you would like to explore the data values in either table, you can run `SELECT * FROM CARS` or `SELECT * FROM MEMBERS`.

In the space below, code the SQL statement that will produce the report above.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The solution follows:

```
select lastname, firstname, carmake, carmodel
       FROM members INNER JOIN cars
       ON cars.membernum = members.membernum
       order by lastname, firstname
```

Some thoughts on the process of coding this statement:

- The problem asked for all the cars owned by members so that means we want an INNER JOIN with the table members on the left and cars on the right of the join.
- The result shows the table sequenced by **firstname** within **lastname**.

Two alternate solutions follow:

```
select lastname, firstname, carmake, carmodel
       FROM members, cars
       where cars.membernum = members.membernum
       order by lastname, firstname
```

```
select lastname, firstname, carmake, carmodel
FROM members inner join cars
using(membernum)
order by lastname, firstname
```

While the three forms are functionally equivalent, by coding INNER JOIN, you are explicitly saying what you want to do. Any person who has been exposed to SQL can understand the purpose of the statement. Discuss this aspect with the class.

On the other hand, this flexibility exists only when you are coding an INNER JOIN of two tables. We see in a few minutes that the WHERE can be more easily coded when joining three or more tables.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Correlation names

IBM i

- Temporary rename of the table
  - members = m
  - cars = c

```
select lastname, firstname, carmake, carmodel
       FROM members m INNER JOIN cars c
       ON c.membernum = m.membernum
       order by lastname, firstname
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-8. Correlation names OD477.0

Notes:

A correlation name can be defined in the FROM clause of a query and in the first clause of an UPDATE or DELETE statement. For example, the clause shown below establishes **C** as a correlation name for CARCLUB/CARS:

```
FROM CARCLUB/CARS C
```

A correlation name is associated with a table or view only within the context in which it is defined. Hence, the same correlation name can be defined for different purposes in different statements, or in different clauses of the same statement.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used as a shorter name for a table or view. In the example shown above, **C** was used to avoid having to enter CARCLUB/CARS more than once.

If a correlation name is specified for a table name or view name, any qualified reference to a column of that instance of the table or view *must* use the correlation name rather than the table name or view name. For example, the reference to CARS.CARMODEL in the

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

following example is incorrect, because a correlation name has been specified for the table CARS.

```
FROM CARS C
WHERE CARS.CARMODEL='442'
      ****
```

The qualified reference to CARMODEL should instead use the correlation name, **C**, as shown below:

```
FROM CARS C
WHERE C.CARMODEL='442'
```

If the column name is unique and not ambiguous, it does not need to be qualified by the correlation name.



Note

A correlation name is in effect for the SQL statement in which it is coded.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — A discussion of correlation names makes sense at this point since JOINS can be cumbersome with all the qualified names. Note that the correlation name can be used for the qualified name of a table or view only. The correlation name is “active” only for the statement in which it is coded.

Details — Connect the two **m**s and the two **c**s so that the students can see the correlation.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Add tables to the car club

IBM i

We want to track members' technical skills.

CARTECH

MEMNUM	TECHCODE
1	01
1	03
1	06
3	06
3	02
9	02
9	04
11	01
11	07

TECHAREA

TECHCODE	TECHDESC
06	Engine - Mechanical
01	Body Repair
02	Body Detailing
03	Engine Detailing
04	Interior Detailing
05	Brakes - Mechanical
07	Suspension - Mechanical

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-9. Add tables to the car club OD477.0

Notes:

At a recent meeting, the Car Club decided to keep track of the technical skills that each member possesses. The purpose would be to provide the information to other members when they requested assistance in some area of repairing a car or improving the appearance of the car.

The requirement was implemented as seen in the visual which shows you the values stored in two tables, described below:

TECHAREA

```
COLUMN      DATA              COLUMN
            TYPE         LENGTH   SCALE

TECHCODE    CHAR              2
TECHDESC    CHAR             25
```

This table has a code and a description. The code (TECHCODE) can be used to join to the corresponding TECHCODE in the CARTECH table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CARTECH

COLUMN	DATA TYPE	COLUMN LENGTH	SCALE
MEMNUM	DECIMAL	3	0
TECHCODE	CHAR	2	

Notice that the CARTECH table includes no descriptive columns. It simply contains all the technical areas in which various members have competence.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — These new tables are used in conjunction with the CARS and MEMBERS tables to show a more complex join.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Multi-table JOINs

IBM i

Problem: Produce a result set that includes:

Member number,
FirstName,
LastName,
Description of technical competence

TECHAREA

COLUMN	DATA TYPE	COLUMN	
		LENGTH	SCALE
TECHCODE	CHAR	2	
TECHDESC	CHAR	25	

CARTECH

COLUMN	DATA TYPE	COLUMN	
		LENGTH	SCALE
MEMNUM	DECIMAL	2	0
TECHCODE	CHAR	2	

MEMBERS

COLUMN	DATA TYPE	COLUMN	
		LENGTH	SCALE
MEMBERNUM	DEC	2	0
LASTNAME	CHAR	25	
FIRSTNAME	CHAR	15	
DUESAMOUNT	DEC	5	2
STREET	CHAR	25	
CITY	CHAR	15	

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-10. Multi-table JOINs OD477.0

Notes:

The club wants to list members and their areas of technical expertise for the Car Club's website.

Can you suggest how this information can be produced?

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Develop the need for a more complex join across many tables.

Details — Let the students talk about this problem and let them even suggest solutions to the problem before you show them the answer on the next visual.

If they need a hint, point out that the column named member number is common to the MEMBERS table and the CARTECH table. The columns have the same characteristics, but the column names are different. Also, mention that the TECHAREA table can be joined to the CARTECH table using the TECHCODE column.

We can solve this problem using JOIN. Are you surprised?

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Join of three tables using ON

IBM i

```
select membernum, rtrim(firstname)||' '||rtrim(lastname) as Name,
        techdesc as Technical_area
        from members m inner join cartech c
                on m.membernum =c.memnum
       inner join techarea t
       on c.techcode=t.techcode

MEMBERNUM          NAME                          TECHNICAL AREA
    1              Bob Slaney                    Body Repair
    1              Bob Slaney                    Engine Detailing
    1              Bob Slaney                    Engine - Mechanical
    3              Mary Shrauger                 Engine - Mechanical
    3              Mary Shrauger                 Body Detailing
    9              Romaine Rockefeller           Body Detailing
    9              Romaine Rockefeller           Interior Detailing
   11              Joyce Brown                   Body Repair
   11              Joyce Brown                   Suspension - Mechanical
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-11. Join of three tables using ON OD477.0

Notes:

A number of the capabilities of SQL are shown in the solution to the problem:

- Headings for the result set are tailored using the AS clause.
- The || operator and the RTRIM scalar function are used to present the values in a more legible fashion.
- Correlation names are used to represent table names for this query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Here is a subset of the data from the MEMBERS table:

```
Member Number  First Name       Surname
       1       Bob              Slaney
       2       Ernest           Humphries
       3       Mary             Shrauger
       4       Jim              Moline
       5       Doug             Armstrong
       6       Annette          Jamison
       7       Diane            Williams
      10       Gregory          Simpson
      11       Joyce            Brown
       8       David            Hall
       9       Romaine          Rockefeller
      12       Margaret         Brown
```

The following is the data from the TECHAREA table. It contains a description of various areas of technical expertise plus an associated code:

```
TECHCODE  TECHDESC
   06     Engine - Mechanical
   01     Body Repair
   02     Body Detailing
   03     Engine Detailing
   04     Interior Detailing
   05     Brakes - Mechanical
   07     Suspension - Mechanical
```

The following is the data from the CARTECH table. It contains a technical code and the member number who is proficient in that area:

```
MEMNUM  TECHCODE
    1      01
    1      03
    1      06
    3      06
    3      02
    9      02
    9      04
   11      01
   11      07
```

By joining the MEMBERS table to the CARTECH table, we can pull all skill codes for each member. Then, by joining the CARTECH table to the TECHAREA table, we are able to find the character description we want in our result set.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — In the example in the visual, the MEMBERS table is first joined to the CARTECH table. The resulting table of this join is then joined to the TECHAREA table.

Transition statement — The next example performs the same query using WHERE rather than ON.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Join of three tables using WHERE

IBM i

```
select membernum, rtrim(firstname)||' '||rtrim(lastname) as Name,
        techdesc as Technical_area
        from members m, cartech c, techarea t
        where m.membernum=c.memnum and c.techcode=t.techcode

MEMBERNUM   NAME                      TECHNICAL_AREA
    1        Bob Slaney                Body Repair
    1        Bob Slaney                Engine Detailing
    1        Bob Slaney                Engine - Mechanical
    3        Mary Shrauger             Engine - Mechanical
    3        Mary Shrauger             Body Detailing
    9        Romaine Rockefeller        Body Detailing
    9        Romaine Rockefeller        Interior Detailing
   11        Joyce Brown               Body Repair
   11        Joyce Brown               Suspension - Mechanical
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-12. Join of three tables using WHERE OD477.0

Notes:

The result set in this example is row-for-row the same as the previous visual. Unless you were to add an ORDER BY, you cannot be sure that the result set will be presented in a certain sequence. The order in which the joins are performed is determined by the order in which the conditions are coded in the WHERE clause. The order in which the tables are coded in the FROM has no influence whatsoever. In comparison, the order in which the join is performed is determined by the order in which the INNER JOINs are coded.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The result set in this example is exactly the same as the previous visual. Unless you were to add an ORDER BY, you can not be sure that the result set will be presented in a certain sequence.

Point out that the order in which the joins are performed is determined by the order in which the conditions are coded in the WHERE clause. The order in which the tables are coded in the FROM has no influence whatsoever. In comparison, the order in which the join is performed is determined by the order in which the INNER JOINs are coded.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

LEFT OUTER JOIN

IBM i

```
select lastname, firstname, carmake, carmodel
       FROM members LEFT OUTER JOIN cars
       ON cars.membernum = members.membernum
       order by lastname

LASTNAME                  FIRSTNAME        CARMAKE          CARMODEL
Armstrong                 Doug             Chevrolet        Corvette
Brown                     Joyce            Dodge            Superbee
Brown                     Margaret         -                -
Hall                      David            Plymouth         Roadrunner
Humphries                 Ernest           Ford             Mustang
Humphries                 Ernest           Ford             Thunderbird
Jamison                   Annette          Ferrari          308
Jamison                   Annette          Alfa Romeo       Targa
Moline                    Jim              -                -
Rockefeller               Romaine          -                -
Shrauger                  Mary             Chevrolet        Corvette
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-13. LEFT OUTER JOIN OD477.0

Notes:

The LEFT OUTER JOIN includes rows from the table named to the left where the values in the joined columns are matched as well as not matched by values in the joined columns of the table named on the right.

In the example in the visual, we list car club members who do not own a car (carmake and carmodel are nulls, represented by a dash) as well as those who do own cars.

We could do the same thing using a RIGHT OUTER JOIN to provide exactly the same result as LEFT OUTER JOIN:

```
FROM CARS RIGHT OUTER JOIN MEMBERS
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Note that the LEFT OUTER JOIN shows rows with corresponding values in both tables and in rows from the left table named in the LEFT OUTER JOIN clause.

If necessary to illustrate this point, show the following query:

```
select lastname, firstname, carmake, carmodel
       from members inner join cars
       on members.membernum = cars.membernum
       order by lastname
```

Point out that everyone listed owns at least one car. Now run it and modify the INNER JOIN to be a LEFT OUTER JOIN. We get the same results as an INNER JOIN plus we also get those members who do not own a car.

If there is a question about the possibility to list rows, including those that do not match from both tables, this can be done by running a full outer join. This is not supported on the i.

Since V5R1, RIGHT OUTER JOINs are supported. A RIGHT OUTER JOIN is the converse of a LEFT OUTER JOIN. You can use one instead of the other. Does the following code perform a RIGHT OUTER JOIN? The tables are the reverse of the example:

```
select lastname, firstname, carmake, carmodel
       FROM cars LEFT OUTER JOIN members
       ON cars.membernum = members.membernum
       order by lastname
```

This would give us a list of all cars, including those that do not have owners in the club. Obviously, since we do not have cars without owners, this is not possible. But, if it were, this JOIN would work.

Transition — The LEFT OUTER JOIN lists the non-matching rows in the result set. What if we simply wanted to list those rows from the MEMBERS table that had no corresponding rows in the CARS file? Then we would have a list of all MEMBERS in the club who do not have cars in the club. That would be an EXCEPTION JOIN, which we show you next.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

EXCEPTION JOIN

IBM i

```
select members.membernum, firstname, lastname
       from members
       exception join cars on members.membernum = cars.membernum


 Member Number  First Name        Surname
        4       Jim               Moline
        9       Romaine           Rockefeller
       12       Margaret          Brown
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-14. EXCEPTION JOIN OD477.0

Notes:

The left EXCEPTION JOIN lists those rows in the MEMBERS table that have no corresponding rows in the CARS file. The result set in this example is a list of all MEMBERS in the club who do not have cars in the club. There is a right exception join, too, that works just like a left exception join but with the tables reversed.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — Also available, but the details are not covered in this course:

- CROSS JOIN

Mention it and refer the students to the reference manual.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

5.2. UNION

Instructor topic introduction

What students will learn — Students learn how to use the UNION clause.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

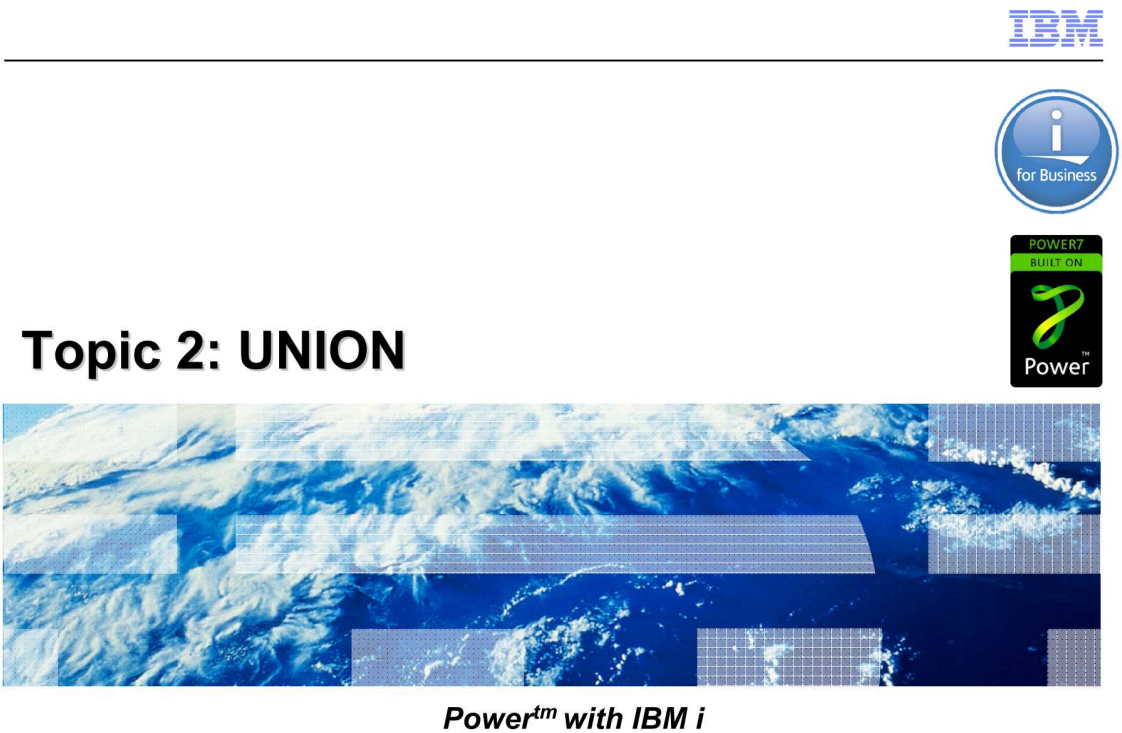


Figure 5-15. Topic 2: UNION OD477.0

Notes:

Unions are used to merge the results of two or more results tables to one overall result table. Let's explore some examples.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

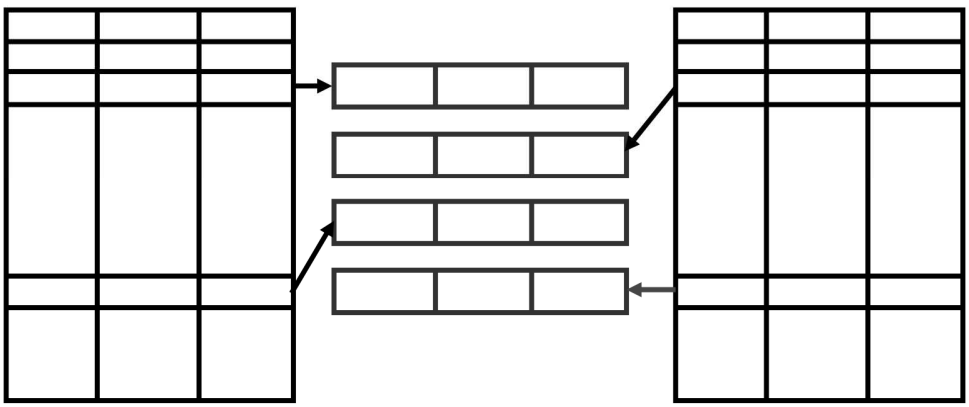
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

UNION of result tables

IBM i



- Like a merge
- Can be used to combine current and historical data

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-16. UNION of result tables OD477.0

Notes:

UNION or UNION ALL is used to merge the results of two or more results tables to one overall result table.

- If UNION ALL is specified, the result consists of all rows in each table.
- If UNION is specified without the ALL option, the result is the set of all rows in either table, with duplicate rows eliminated.

In either case, each row of the UNION table is either a row from the first table or a row from the second table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Tell the students that you want to list car club members and their cars below each other.

Let them execute the following example:

```
select lastname, firstname from members
union
select carmake, carmodel from cars
```

Ask what the students notice.

The query produces these results:

```
Surname                   First Name

Alfa Romeo                Targa
Armstrong                 Doug
Austin                    Healey
Brown                     Joyce
Brown                     Margaret
Chevrolet                 Chevelle SS454
Chevrolet                 Corvette
Chevrolet                 El Camino
Dodge                     Superbee
Ferrari                   308
```

Next, explain that UNION performs a sort of the result table in ascending sequence using the first resulting column.

Question

How can you list the member's last name first followed by their cars?

The students will offer several solutions, which you might discuss. If we were to include MEMBERNUM as the first column, it would force the sort on this column. One idea would be to use ORDER BY. It might be necessary to use a “dummy column.”

Mention that you can work with dummy columns in the SELECT clause. We discuss dummy columns in a subsequent visual. For example:

- A blank column **''** corresponds to a CHAR column;
- A zero column, **0**, corresponds to a column of a numeric data type

Additional information — Mention that the data types of the corresponding columns have to match, but not necessarily the lengths of these columns.

If the column name is the same in each SELECT, then you can use the column name and an ORDER BY statement.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

You could rename a column with an AS clause so that the column names would be the same. Then, you could use the column name in the ORDER BY statement. Otherwise, you could specify the column number.

Mention that the headings of the result table are derived from the headings of the table in the first SELECT clause.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

UNION and UNION ALL

IBM i

- **UNION**
  - Suppresses identical rows
  - Performs an ascending sort according to the values in all columns, taken as an entirety from left to right
  - ORDER BY can rearrange the result

```
select * from current.members
     union
select * from former.members
```

- **UNION ALL**
  - Does not suppress identical rows
  - Does not perform automatic sort

```
select * from us.members
     union all
select * from uk.members
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-17. UNION and UNION ALL OD477.0

Notes:

UNION performs a sort involving all columns and then suppresses duplicate rows. Hence we see a unique set of all current and former club members.

UNION ALL does not perform a sort and does not suppress duplicate rows. Hence we see a set of US and UK car club members. If it were possible that an individual was a member in both clubs, we would see the individual's name twice.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — ORDER BY performs an additional sort after the union sort is completed. Demonstrate this by running the following query:

```
select * from members
union all
select * from members
order by membernum
```

You should see two copies of each row, sequenced by MEMBERNUM and therefore each row is displayed one under another.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using dummy columns

IBM i

```
select membernum, lastname, ' ' from members union
select membernum, carmake, carmodel from cars
order by membernum

Member Number  Surname                    Constant value

=>     1       Slaney
       1       Oldsmobile                 Cutlass 442
       1       Chevrolet                  Corvette
       1       Maserati                   Diablo
=>     2       Humphries
       2       Ford                       Mustang
       2       Ford                       Thunderbird
=>     3       Shrauger
       3       Chevrolet                  Corvette
       3       Plymouth                   Hemi Superbird
=>     4       Moline
       :          :                           :
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-18. Using dummy columns OD477.0

Notes:

Dummy columns can be used in situations where you want to include more columns from one of the tables than from the other. In this example, we want to create a list of cars and owners, including the model of car, but not the first name of the owner. We added a dummy CHAR column, **' '**.

Notice that our labels are used as column headings and that the dummy column is assigned the label, *Constant value*. The -> points to the values from the row from the MEMBERS table that are grouped with those of the CARS table.

We used ORDER BY on the first column so that our rows from both the MEMBERS table and the CARS table would be sequenced together. This way we can see which MEMBERS own which CARS.

UNION does not have inherent features for arranging the data.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This example is coded as a UNION to show how a dummy column can be used. We could code a solution using a JOIN that would also answer the question of which members own which cars:

```
select members.membernum, lastname, carmake, carmodel
       from members join cars
       on members.membernum=cars.membernum
```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

UNION plus text column

IBM i

```
select carmake, carmodel, char(caryear), 'More than 30 years old'
       from cars
       where (year(curdate()) - caryear) > 30
union all
select carmake, carmodel, char(caryear), Not more than 30 years old'
       from cars
       where not(year(curdate()) - caryear) > 30

 Chevrolet        El Camino             1977         More than 30 years old
 Maserati         Diablo                1971         More than 30 years old
 Chevrolet        Chevelle SS454        1970         More than 30 years old
 Austin           Healey                1965         More than 30 years old
 Alfa Romeo       Targa                 1973         More than 30 years old
 Plymouth         Roadrunner            1969         More than 30 years old
 Dodge            Superbee              1969         More than 30 years old
 Plymouth         Hemi Superbird        1970         More than 30 years old
 Plymouth         GTX                   1966         More than 30 years old
 Ford             Thunderbird           1957         More than 30 years old
 Mercedes         450SL                 1974         More than 30 years old
 Chevrolet        Corvette              1985         Not more than 30 years old
                                                                      More...
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-19. UNION plus text column OD477.0

Notes:

This example uses UNION ALL to select certain rows of the table CARS and to add a text description.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — Point out that we could use a CASE expression rather than a UNION for this query.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

UNION with ORDER BY

IBM i

```
select carmake, char(caryear)
       from cars
union all
select '** Average', char(avg(caryear))
       from cars
order by 1

CARMAKE          CHAR ( CARYEAR )
** Average       1970.23529411764705882352941176 4
Alfa Romeo       1973
Austin           1965
Chevrolet        1967
Chevrolet        1985
Chevrolet        1965
Chevrolet        1977
Chevrolet        1970
Dodge            1969
Ferrari          1984
   :               :
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-20. UNION with ORDER BY OD477.0

Notes:

This example was coded to answer the request for a list of each car's model year compared to the average of the model years in the car club.

Notice the use of the ** prefix to force the AVERAGE to be sorted as the first row in the result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

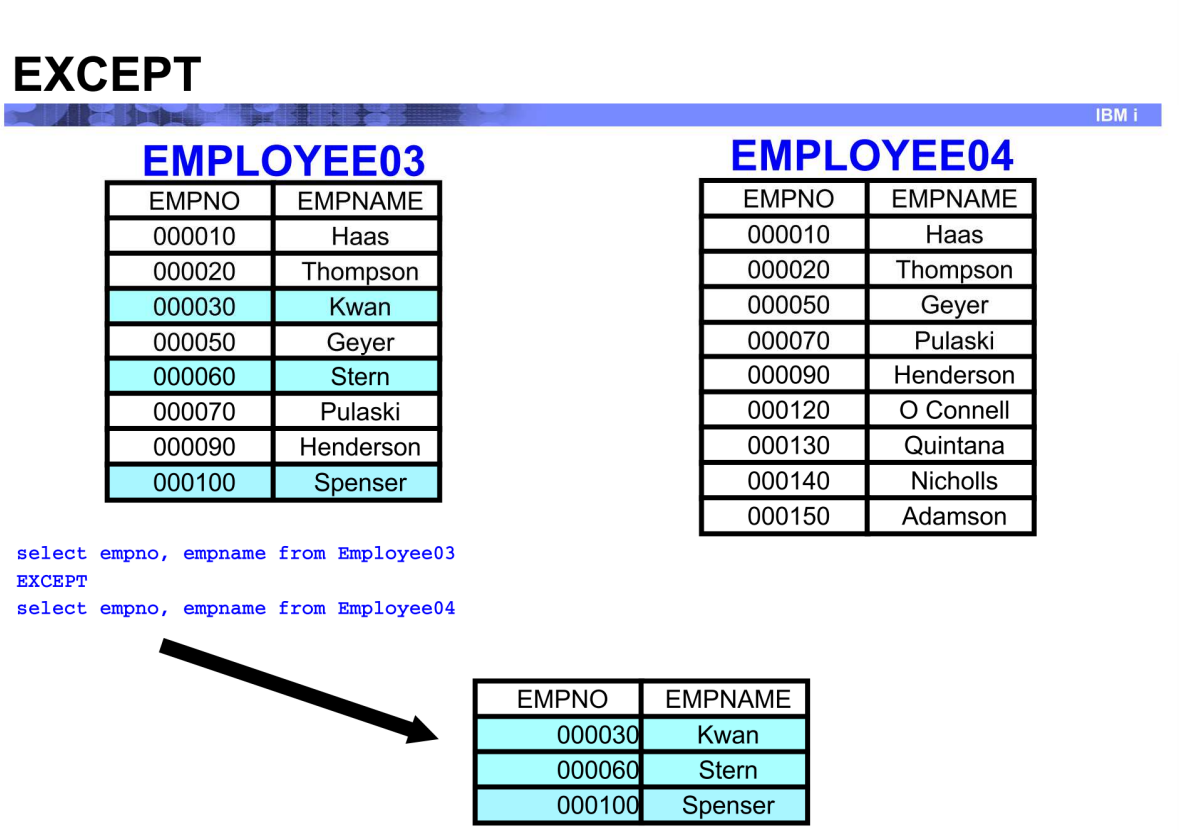
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-21. EXCEPT OD477.0

Notes:

The EXCEPT keyword returns the result set of the first subselect minus any matching rows from the second subselect.

```
select empno, empname from employee03
except
select empno, empname from employee04
```

To better understand the results from these SQL statements, imagine that SQL goes through the following process:

Step 1. SQL processes the first SELECT statement:

```
select empno, empname from employee03
```

Which results in an interim result table:

EMPNO	EMPNAME
000010	Haas
000020	Thompson
000030	Kwan

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

EMPNO	EMPNAME
000050	Geyer
000060	Stern
000070	Pulaski
000090	Henderson
000100	Spenser

Step 2. SQL processes the second SELECT statement:

```
select empno, empname from employee04
```

Which results in another interim result table:

EMPNO	EMPNAME
000010	Haas
000020	Thompson
000050	Geyer
000070	Pulaski
000090	Henderson
000120	O Connell
000130	Quintana
000140	Nicholls
000150	Adamson

Step 3. SQL takes the first interim result table, removes all of the rows that also appear in the second interim result table, and removes duplicate rows:

EMPNO	EMPNAME
000030	Kwan
000060	Stern
000100	Spenser

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

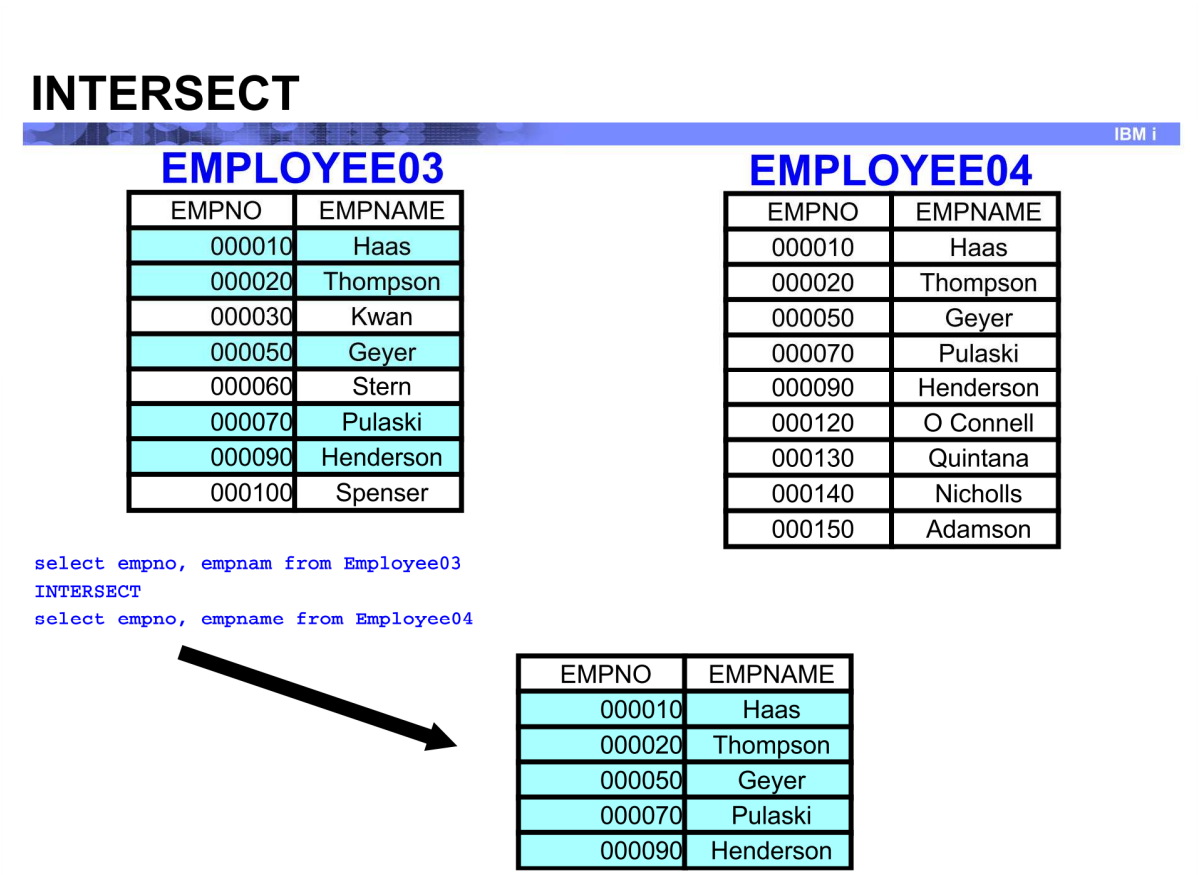
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-22. INTERSECT OD477.0

Notes:

The INTERSECT keyword returns a combined result set that consists of all of the rows that exist in both result sets.

```
select empno, empname from employee03
INTERSECT
select empno, empname from employee04
```

To better understand the results from these SQL statements, imagine that SQL goes through the following process:

Step 1. SQL processes the first SELECT statement:

```
select empno, empname from employee03
```

Which results in an interim result table:

EMPNO	EMPNAME
000010	Haas
000020	Thompson
000030	Kwan

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

EMPNO	EMPNAME
000050	Geyer
000060	Stern
000070	Pulaski
000090	Henderson
000100	Spenser

Step 2. SQL processes the second SELECT statement:

```
select empno, empname from employee04
```

Which results in another interim result table:

EMPNO	EMPNAME
000010	Haas
000020	Thompson
000050	Geyer
000070	Pulaski
000090	Henderson
000120	O Connell
000130	Quintana
000140	Nicholls
000150	Adamson

Step 3. SQL takes the first interim result table, compares it to the second interim result table, and returns the rows that exist in both tables minus any duplicate rows.

EMPNO	EMPNAME
000010	Haas
000020	Thompson
000050	Geyer
000070	Pulaski
000090	Henderson

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U04** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U04** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U04** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint

IBM i

1. Assume you want to list all members and their cars plus you want to list the members without cars. What are you doing?

2. True or False: When coding a join, coding a condition is optional. If you do not code a condition, all rows will be selected.

3. We want to make sure that we have been maintaining the CARS table correctly. We want to list all cars in the CARS table that are not associated with a member in the MEMBERS table. Is this possible with a JOIN?

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-23. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. Assume you want to list all members and their cars plus you want to list the members without cars. What are you doing?

   The answer is LEFT OUTER JOIN.

2. True or False: When coding a join, coding a condition is optional. If you do not code a condition, all rows will be selected.

   The answer is False. All rows from the first table will be combined with all rows from the second table.

3. We want to make sure that we have been maintaining the CARS table correctly. We want to list all cars in the CARS table that are not associated with a member in the MEMBERS table. Is this possible with a JOIN?

   The answer is Yes, an exception JOIN.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *JOIN and UNION*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-24. Exercise OD477.0

Notes:

Perform the workstation exercise *JOIN and UNION*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- List five types of JOINs supported on the System i
- Use JOIN to retrieve data from multiple tables or views
- Use UNION to retrieve data from multiple tables or views
- Use EXCEPT or INTERSECT to retrieve data from multiple tables or views
- Use correlation names

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 5-25. Unit summary OD477.0

Notes:

This unit focused on how to combine the data from more than one table.

JOIN versus UNION

A JOIN combines tables based on column values. A UNION clause combines tables based on rows.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 6. SubSELECTs

Estimated time

01:05

What this unit is about

This unit describes the use of subSELECT and subquery capabilities of SQL.

What you should be able to do

After completing this unit, you should be able to:

- Code subSELECTs
- Code subqueries in predicates

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercise

  The students perform a workstation exercise to test their understanding of subselect.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Code subSELECTs
- Code subqueries in predicates

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-1. Unit objectives OD477.0

Notes:

This unit focuses on how to write a select within a select. This is called a *subSELECT*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Cover this unit quickly as you will need time for the units that follow. Some subselects are labeled for reference only in the instructor notes.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

6.1. Using subSELECTs and subqueries

Instructor topic introduction

What students will learn — Students will learn how to use and code subqueries. They will also learn about SQL predicates as illustrated and used in subqueries.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

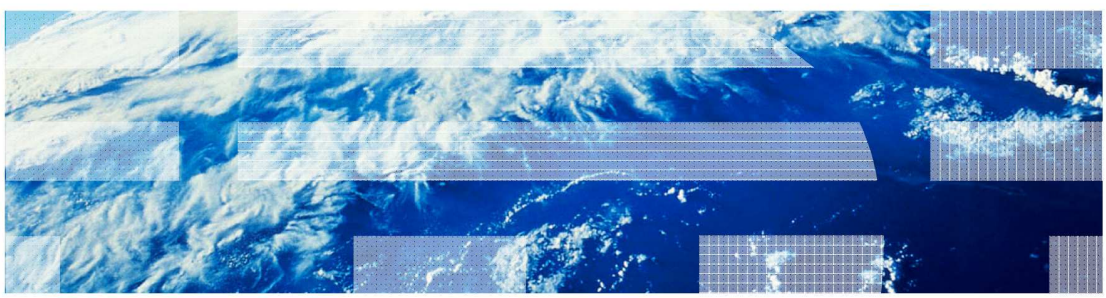


Figure 6-2. Topic 1: Using subSELECTs and subqueries OD477.0

Notes:

Subqueries allow you to use SELECT criteria within a predicate. Let's explore a few examples.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

What is a subSELECT?

IBM i

- **SELECT** within a **SELECT**
- Coded in:
  - **WHERE**
  - **HAVING**
  - **Predicate**
- **SELECT** in predicate is called a **SUBQUERY**

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-3. What is a subSELECT? OD477.0

Notes:

A subSELECT is used whenever the desired result cannot be obtained by a single pass against the table. A subSELECT presents a result set that can then be used in another select statement.

When a subSELECT is coded in a predicate, it is called a *subquery*. A predicate specifies a condition that is true, false, or unknown about a given row or group.

Generally, you will find that the terms subSELECT and subquery are used interchangeably.

While we do not discuss other subSELECTs in this class, you might want to read the documentation about other subSELECTs in the DELETE and UPDATE statements.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Note that a subSELECT can be coded in:

- fullselect
- CREATE VIEW
- INSERT
- Search condition: WHERE/HAVING clause of SELECT.

When a subSELECT is coded as part of search condition (predicate) it is known as a subquery. You can only use FROM, WHERE, GROUP BY, and HAVING (no ORDER BY) in subqueries. You are only allowed a maximum of 32 nested SELECTs (subselects).

In a SQL statement, a maximum of 32 tables are allowed. If a table is referenced more than once, then you must count it for each reference. The total maximum of subSELECTs, UNIONs, and JOINs is 32.

Transition — Before we discuss the details, let us show you a situation where you might use a subquery.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Why subSELECT?

IBM i

Query: Which members own cars that are older than the club average?

```
select lastname, carmake, carmodel, char(caryear) as Year
       from members inner join cars
       on cars.membernum = members.membernum
       where caryear < avg(caryear)
       order by lastname

Use of column function AVG not valid.
```

You *cannot* use a column function in a WHERE clause.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-4. Why subSELECT? OD477.0

Notes:

Suppose that we want to write a query to select all cars whose model year is older than the average model year of all cars in the club. We also want to display some other information.

The AVG function cannot be performed in a WHERE clause.

We must develop an alternate solution to the problem.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using subSELECT

IBM i

```
select lastname, carmake, carmodel, char(caryear) as year
       from members inner join cars
       on cars.membernum = members.membernum
       where caryear < (select avg (caryear) from cars)
       order by lastname

Surname       CARMAKE          CARMODEL         YEAR

Armstrong     Chevrolet        Corvette         1965
Brown         Dodge            Superbee         1969
Hall          Plymouth         Roadrunner       1969
Humphries     Ford             Mustang          1965
Humphries     Ford             Thunderbird      1957
Shrauger      Plymouth         Hemi Superbird   1970
Simpson       Chevrolet        Chevelle SS454   1970
Slaney        Oldsmobile       Cutlass 442      1967
Slaney        Chevrolet        Corvette         1967
Williams      Austin           Healey           1965
Williams      Plymouth         GTX              1966
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-5. Using subSELECT OD477.0

Notes:

The SELECT *must be enclosed in parentheses* to be evaluated correctly. It is incorrect to code:

```
select lastname, carmake, carmodel, char(caryear)
       from cars
       where caryear > select avg(caryear) from cars

Token AVG was not valid.
```

In this example, we are selecting cars with a model year newer than the average of all model years in the car club.

The sample subselect produces a *single value*. Thus, the comparison in the WHERE clause can be done with a simple arithmetic operator.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The exercise that we performed in the SELECT unit where we wanted all employees in the same department as “Adamson,” could have been solved using a subquery:

```
select lastname, firstnme, workdept
   from employee
   where workdept = (select workdept from employee
                                     where lastname='Adamson')
```

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition — Before we discuss subSELECTs further, let's review predicates.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Predicates and subqueries

IBM i

- Basic
  - Compares two values
- Quantified
  - Compares a value with a set of values
  - SOME, ANY, ALL
- EXISTS
- IN

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-6. Predicates and subqueries OD477.0

Notes:

Because you always code a subquery within a predicate, let us quickly review the SQL predicates:

- **Basic**
  - Compares two values
  - Uses the form:

```
expression = (subselect)
             <>
             ¬=
             <
             >
             <=
             ¬>
             >=
             ¬<
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

A subselect in a basic predicate must specify a single result column and must not return more than one value.

If the value of either operand is null or the result of the subselect is empty, the result of the predicate is unknown. Otherwise the result is either true or false.

- **Quantified**

  The quantified predicate compares a value with a set of values. It is similar in form to the basic predicate, but the result set is qualified with SOME, ANY, or ALL. The result of the subquery is expected to be a set of values.

  The condition

  ```
  caryear IN (1965, 1966, 1967)
  ```

  Is the same as

  ```
  caryear = ANY (1965, 1966, 1967)
  ```

- **EXISTS**

  The EXISTS predicate tests for the existence of certain rows. The subselect can specify any number of columns, and

  - The result is true only if the number of rows returned by the subselect is not zero.
  - The result is false only if the number of rows returned by the subselect is zero.

  A result of *unknown* can never be returned by an EXISTS.

- **IN**

  The IN predicate compares a value to a set of values.

  ```
  expression IN (subselect) = expression = ANY (subselect)
  expression NOT IN (subselect) = expression <> ALL (subselect)
  ```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition — Now we show you examples of queries that use the various forms of predicates.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with basic predicate

IBM i

Who is paid the highest annual salary?

1. What is the highest salary paid by the company?

```
select max(salary) from employee
MAX ( SALARY )
52,750.00
```

2. Who is paid that amount?

```
select rtrim(firstnme)||' '||rtrim(lastname) as name, salary
          from employee
          where salary = (select max(salary) from employee)

         NAME                                    SALARY

         Christine Haas                       52,750.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-7. Subquery with basic predicate OD477.0

Notes:

We want to find out who is paid the highest salary in the company. In order to find out, we need to break up the task into steps:

1. What is the highest salary paid in the company?

   We can find this out by coding this query:

   ```
   select max(salary) from employee
   ```

   We are returned a result set that contains the value 52,750.00.

2. Now we want to find out who earns this maximum salary. We include the first query as a subquery of this query:

   ```
   select rtrim(firstnme)||' '||rtrim(lastname) as name, salary
             from employee
             where salary = (select max(salary) from employee)
   ```

   The *subquery* is also called the *inner* query. The main query is called the *outer* query.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

A *basic* predicate is one that uses =, <, >, ¨, or any combination of these symbols. A subquery in a basic predicate specifies a single result column and returns a single value.

The result of the predicate can be **true**, **false**, or **unknown**.

- The predicate result is **unknown** if:
  - The value of the expression on the left side of the predicate is **null**.

    For example, if the predicate is `WHERE BONUS + COMM > 50000`, and COMM is null in the row being read by the outer query, the expression `BONUS + COMM` evaluates to null and the value of the entire predicate is unknown for this row of the outer query. In other words, it is not known whether or not the row currently being read by the outer query satisfies the predicate.
  - The subquery result set is empty or null.

    For example, if the predicate is:

    ```
    WHERE SALARY > (SELECT SALARY FROM EMPLOYEE WHERE EMPNO = '000405')
    ```

    Since there is no employee 000405, the result of the subquery will be empty and the entire predicate is **unknown** for any row of the outer query.
- The predicate result is **false** if the row in the outer query does not satisfy the operator of the subquery. The value returned by the subquery is 52750.00. If we are processing the second row of the table, the value of salary in the current row of the outer query is 38250.00. For the second row, the predicate value is false.
- In all other cases, the predicate value is **true**.

  The row from the outer query appears in the final result *only if* the predicate value is true for that row.

The subquery in the example determines the largest value of SALARY in the EMPLOYEE table. The outer query then determines employee number, last name, and salary of all employees whose salary is equal to the one returned by the subquery.

Note: Several people might earn the same salary. If the same maximum salary is earned by more than one person, the subquery would still return just the one value. The outer query would return all employees who made that salary.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — For this example and each example that follows, we will think through the problem presented in each question and break the problem down into small steps. For each example, often more than one visual is needed to develop a solution.

Details — Take the students through the process of arriving at a workable solution.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with the IN predicate (1 of 3)

IBM i

Which employees are assigned to at least one project?
List those employees who are working in departments that are responsible for projects.
1. Which departments are responsible for projects?

```
SELECT DEPTNO FROM PROJECT
```

2. What employees are assigned to these departments?

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM EMPLOYEE
  WHERE WORKDEPT IN
    (SELECT DEPTNO FROM PROJECT)

EMPNO  LASTNAME      WORKDEPT
000020 Thompson          B01
000030 Kwan              C01
000050 Geyer             E01
000060 Stern             D11
000070 Pulaski           D21
000100 Spenser           E21
000130 Quintana          C01
000140 Nicholls          C01
000150 Adamson           D11
000160 Pianka            D11
000170 Yoshimura         D11
:
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-8. Subquery with the IN predicate (1 of 3) OD477.0

Notes:

This sounds like an easy question, but before we can answer the question, we need to determine which departments are actively working on projects. Our database includes a PROJECTS table that can tell us the answer to the secondary question that we must answer first.

Once we have coded the query that gives us the department numbers, we include a subquery as part of the WHERE clause. The subquery will produce a result set and thus, we use the IN predicate to check whether a department number in the *outer* query matches one of the department numbers in the *inner* query (subquery) to produce the result set that answers our main question.

This question could also be answered by the following JOIN query:

```
SELECT EMPNO, LASTNAME, WORKDEPT
       FROM EMPLOYEE, PROJECT
       WHERE WORKDEPT = DEPTNO
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out the inner and outer query components. Emphasize that the process of breaking the problem into its components will assist the students in determining the subquery and finally the outer query.

Transition statement — Another example follows.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Subquery with the IN predicate (2 of 3)

IBM i

List all employees who work for department E01 or any of its reporting departments.
1. List department E01 and its reporting departments.

```
SELECT DEPTNO FROM DEPARTMENT
                  WHERE (DEPTNO = 'E01') OR (ADMRDEPT = 'E01')
```

2. List the employees in E01 or any of its reporting departments.

```
SELECT EMPNO, LASTNAME, WORKDEPT
       FROM EMPLOYEE
       WHERE WORKDEPT IN
             (SELECT DEPTNO FROM DEPARTMENT
                            WHERE (DEPTNO = 'E01') OR (ADMRDEPT = 'E01'))

EMPNO          LASTNAME                  WORKDEPT
000050         Geyer                     E01
000090         Henderson                 E11
000100         Spenser                   E21
000280         Schneider                 E11
000290         Parker                    E11
000300         Smith                     E11
000310         Setrignt                  E11
000320         Mehta                     E21
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-9. Subquery with the IN predicate (2 of 3) OD477.0

Notes:

Again, we break the problem up into its component parts to determine the subquery and the outer query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — The next example is based on NOT and is a little more challenging.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with the IN predicate (3 of 3)

IBM i

Which employees are not assigned to a project?

1. Which employees are assigned to projects?

```
select respemp from project group by respemp
```

2. Which employees are *not in* this result set?

```
select empno, rtrim(firstnme)||' '||lastname as name
           from employee
           where empno NOT IN(select respemp from project
                     group by respemp)

         EMPNO    NAME
         000110   Vicenzo Lucchesi
         000120   Sean O Connell
         000130   Delores Quintana
         000140   Heather Nicholls
         000170   Masatoshi Yoshimura
         000180   Marilyn Scoutten
         000190   James Walker
         000200   David Brown
            :          :
            :          :
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-10. Subquery with the IN predicate (3 of 3) OD477.0

Notes:

For this example, to determine which employees have not been assigned a project, we follow these steps:

1. Which employees are assigned to, that is, responsible for, a project?
   a. In the PROJECT table, there is a column designating the responsible employee, RESPEMP.
   b. We can formulate a query that tells us the employee number responsible for each project:

      ```
      select respemp from project
      ```

      Now we have an unordered list of employee numbers assigned to all projects including duplicates.
   c. We can refine the list so that we do not have duplicate employee numbers. We use a GROUP BY clause so that we get an ordered set of unique employee numbers:

      ```
      select respemp from project group by respemp
      ```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

OR

```
select distinct respemp from project
```

An ORDER BY could be used rather than a GROUP BY. If you use an ORDER BY, you will have a set with duplicates that will also work, but we decided that duplicates should not be in the set.

2. We search the employee table to find those employees *not in* the result set of employees who are assigned to projects.

The subquery in the example builds a list of the employee numbers of employees that are responsible for projects (RESPEMP in the PROJECT table). The outer query determines the employee number and the name of any employee whose employee number is *not* in the list returned by the subquery. In other words, the query is reporting on employees who are not assigned to projects.

Additional information — The IN predicate, when used with a subquery, enables the outer query to compare the values in a column of the outer table with a list of values provided by the subquery.

A subquery used with an IN predicate must identify a single result column and can return any number of values.

For an IN predicate, if any of the non-null values returned by the subquery match any of the non-null values of the column being searched by the outer query, the rows in the outer query that contain matching values will appear in the final result set. Nulls in the subquery result will never match nulls in the outer query result.

When IN is prefaced with NOT, as in the example, the subquery should be written to return only non-null values. Then, all of the rows of the outer query that do not match any of the values returned by the subquery will appear in the result set. If the NOT IN subquery returns a null value, the outer query will always produce an empty result set.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Define the predicates IN and NOT IN.

Details — In many cases, a subquery with IN can be replaced by a join. If we wanted information about employees that are responsible for projects (the example on the visual wants employees who are *not* responsible for projects), the following two queries would both provide that answer:

```
SELECT EMPNO, LASTNAME
       FROM EMPLOYEE
       WHERE EMPNO IN (select respemp distinct from project
                       group by respemp)

SELECT DISTINCT EMPNO, LASTNAME
       FROM EMPLOYEE, PROJECT
       WHERE EMPNO = RESPEMP
```

Transition statement — As mentioned in the student notes, you must be careful when using NOT IN with nullable columns. This is illustrated by the following example.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IN predicate used with nullable column (1 of 2)

IBM i

Which departments have no assigned employees?

1. Which departments have assigned employees?

```
select workdept from employee
    group by workdept ===>  A00, B01, C01, D11,D21,
                            E01, E11, E21,null
```

2. Which departments are *not in* this set?

```
select deptno, deptname from department
     where deptno NOT IN
     (select workdept from employee
      group by workdept)

DEPTNO  DEPTNAME
******  End of data  ********
```

EMPTY result set!!!

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-11. IN predicate used with nullable column (1 of 2) OD477.0

Notes:

In this example, we are looking for departments who have no assigned employees. Employees are assigned to a WORKDEPT in the EMPLOYEE table.

1. We select a set of department numbers who have employees assigned by scanning the employee table. Note that our set includes a NULL, which means that at least one employee has not been assigned to any department.
2. Satisfied with the results of our inner query, we code an outer query. But, we should be careful using NOT IN when referencing a column that is null capable. Assume that we have created the EMPLOYEE table with WORKDEPT declared as null capable.

   But, this query does *not* work. The result set is EMPTY. The subquery result set would contain the department numbers shown including a null value. As a result, the whole predicate in the outer query evaluates to UNKNOWN, rather than TRUE or FALSE, and no rows in the DEPARTMENT table satisfy the query.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Illustrate that NOT IN for nullable columns must be used carefully.

Details — **Note:** The remaining visuals from this point are for student reference only. They should be skipped if you are using a two day agenda.

Transition statement — The next visual illustrates the use of a subquery using the ALL predicate.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IN predicate used with nullable column (2 of 2)

IBM i

Solution:

```
select deptno, deptname from department
     where deptno NOT IN
     (select workdept from employee
      where workdept is NOT NULL
      group by workdept)


DEPTNO   DEPTNAME
 D01     Development Center
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-12. IN predicate used with nullable column (2 of 2) OD477.0

Notes:

To obtain a valid result, you would need to change the query to accommodate a possible null value.

As an alternative, you could use an **EXISTS** predicate:

```
SELECT DEPTNO, DEPTNAME
       FROM DEPARTMENT D
       WHERE NOT EXISTS
             (SELECT * FROM EMPLOYEE
                      WHERE D.DEPTNO = WORKDEPT)
```

This is known as a *correlated subquery*. We will see another example soon.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with ALL predicate (1 of 2)

IBM i

Which department has the highest salary cost?

1. For each department, what is the total salary?

```
          select workdept, sum(salary) from employee
             group by workdept

WORKDEPT                                SUM ( SALARY )
  A00                                       128,500.00
  B01                                        41,250.00
  C01                                        90,470.00
  D11                                       222,100.00
  D21                                       150,920.00
  E01                                        40,175.00
  E11                                       104,990.00
  E21                                        95,310.00
  -                                          38,000.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-13. Subquery with ALL predicate (1 of 2) OD477.0

Notes:

The purpose of the query is to determine the department with the highest salary costs.

We first determine the total salary paid in each department, showing each department and its total salary cost. Then we can determine the largest value of total salary in this result set.

Since SALARY is a column in the EMPLOYEE table, we use that table to determine the total salary.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The SELECT as shown is not directly used as the subquery in the next visual. We derive the subquery from it. In this visual, the result set contains the work department in order to provide a better understanding of the data.

Ask the class “Why do you think that there is an entry with a null work department and a salary of 38,000.00?”

The answer is that we have hired a new employee at a starting salary of 38,000.00. This employee has not yet been assigned to a work department.

Transition statement — As you might already have guessed, the solution requires the ALL predicate. The next visual illustrates the SQL statement solving the problem.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with ALL predicate (2 of 2)

IBM i

2. Of the totals, which one is the largest?

```
select workdept as dept_num,
       sum(salary) as total_salary
       from employee
       group by workdept
HAVING sum(salary) >= ALL(select SUM(salary)
                          from employee
                          group by workdept)

DEPTNUM                              TOTAL_SALARY
D11                                  222,100.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-14. Subquery with ALL predicate (2 of 2) OD477.0

Notes:

The next step is to find the department for which the total salary paid is the highest of *all* departments. There may be more than one department with the highest total salary.

To find the greatest value of a set of values, you must determine the value in the set that is *greater than or equal to* all values of the set including the value itself.

We revise the inner query and remove the WORKDEPT column since we only need the set of total salaries created using the SUM() function.

Since the outer query needs to select departments rather than employees, a HAVING clause is needed rather than a WHERE clause. The HAVING clause expresses the condition that, for a summary row of the outer query to be displayed, the total salary for each individual department must be greater than or equal to the sum of salaries for all departments.

A quantified predicate compares each value in the outer query with the set of values provided by the subquery. A *quantified predicate* is a predicate that uses one of the

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

following keywords: ALL, ANY, or SOME. As usual, the subselect must specify a single result column. It can return any number of values.

When the ALL keyword is specified, the result of the predicate is:

- *True* if the subquery returns no values or if the specified relationship is true for every value returned by the subquery.
- *False* if the specified relationship is false for at least one value returned by the subquery.
- *Unknown* if the specified relationship is not false for any values returned by the subquery and at least one comparison is unknown because of a null value. Unknown is treated as false.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Remind the students that we must use HAVING because WHERE does not allow column functions.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery using ANY or SOME (1 of 2)

IBM i

Which employees are paid a salary that is higher than the average salary of at least one department?

1. What is the average salary of each department?

```
select workdept, avg(salary) from employee
group by workdept

     WORKDEPT   AVG ( SALARY )
     A00        42,833.33333333333333333333333
     B01        41,250.00000000000000000000000
     C01        30,156.66666666666666666666666
     D11        24,677.77777777777777777777777
     D21        25,153.33333333333333333333333
     E01        40,175.00000000000000000000000
     E11        20,998.00000000000000000000000
     E21        23,827.50000000000000000000000
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-15. Subquery using ANY or SOME (1 of 2) OD477.0

Notes:

We want to determine all employees whose are paid a higher salary than the average salary of at least one department. For these employees, their employee number, last name, and salary should be listed.

We refine the problem and break it up into steps:

1. We need to determine the average salary of each department. We can do this by writing the query on the visual.

   This query provides a list of values. Each value is the average salary paid by one department. To obtain the average salaries of the departments, we must group by WORKDEPT in the subquery.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce the students to the problem and get them thinking about a possible solution.

Details — Once again, the SELECT as shown is not directly used as the subquery in the next visual. We derive the subquery from it. In this visual, the result set contains the work department in order to allow a better understanding of the data. We must drop the work department column in the subquery since we only want a set of values from one column to be tested by ANY.

Transition statement — The next visual illustrates the SQL statement needed to solve the problem. The solution uses the ANY predicate.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery using ANY or SOME (2 of 2)

IBM i

2. Compare the salary of each employee in each department to the set of department averages.

```
select empno, lastname, salary
       from employee
       where salary > ANY
        (select avg(salary) from employee
         group by workdept)

EMPNO     LASTNAME                SALARY
000010    Haas                    52,750.00
000020    Thompson                41,250.00
000030    Kwan                    38,250.00
000050    Geyer                   40,175.00
000060    Stern                   32,250.00
000070    Pulaski                 36,170.00
000090    Henderson               29,750.00
000100    Spenser                 26,150.00
  :                    :
  :                    :
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-16. Subquery using ANY or SOME (2 of 2) OD477.0

Notes:

1. Let us reword the problem. List all employees whose salary is greater than the average salary of *any* one of the departments.

   Another way to state the problem would be, “List all employees whose salary is greater than the average salary of *some* of the departments”.

   ANY means one or more, and therefore is the same as saying *SOME*.

2. Using the previous query as a basis for our inner query in the visual, we build our query to solve the problem. Since we only need the set of averages, we drop the department number (WORKDEPT) column from the query.

3. We build our outer query and compare each employee's salary to the set of department averages.

   If an employee's salary is greater than the average of any element of the result set, we display it.

Because employees have to be determined rather than departments, no grouping is required for the outer query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

In the example, the salary of employee HAAS is higher than all of the department averages. Employee KWAN qualifies because her salary is higher than the average salary of department C01 (her own), although her salary is not higher than the average salary of department A00.

The predicates ANY and SOME have the same meaning and effect. When ANY or SOME are used, the result of the predicate is:

- *True* if the specified relationship is true for at least one value returned by the subquery.
- *False* if the subquery returns no value or if the specified relationship is false for every value returned by the subquery.
- *Unknown* if the specified relationship is not true for any values returned by the subquery and at least one comparison is unknown because of a null value.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Discuss the solution with the students.

Details —

Transition statement — The next example illustrates the use of a correlated query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Correlated subquery (1 of 3)

IBM i

Which employees earn a salary that is higher than the average of their own department?

1. What is the average salary for each department?

```
select workdept, avg(salary) from employee
       group by workdept

WORKDEPT                        AVG ( SALARY )
 A00         42,833.33333333333333333333333
 B01         41,250.00000000000000000000000
 C01         30,156.66666666666666666666666
 D11         24,677.77777777777777777777777
 D21         25,153.33333333333333333333333
 E01         40,175.00000000000000000000000
 E11         20,998.00000000000000000000000
 E21         23,827.50000000000000000000000
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-17. Correlated subquery (1 of 3) OD477.0

Notes:

In this example, the task is to find all employees who have a salary greater than the average salary of their own department.

Again, let us break up the problem into smaller steps:

1. Reword the problem: “List the department, name, and salary of each employee whose salary is higher than the average salary of the department to which he or she is assigned.”
2. For each department, what is the average salary?
3. You might think ahead and notice that we need to know the average salary of each employee's department before we can make a comparison to that employee's salary.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce the problem to the students and lead them to the solution. The solution is shown on the next visual.

Transition statement — To solve the correlation between employees and their departments, we need a correlated subquery. How correlated subqueries are coded is illustrated in the third visual.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Correlated subquery (2 of 3)

IBM i

2. If we look at department A00, which employees are paid higher than the average salary?

```
          select  ' ' as Employee, avg(salary)
                  from employee where workdept='A00'
          union
          select empno, salary
                 from employee where workdept='A00'

EMPLOYEE                        AVG ( SALARY )
             42,833.33333333333333333333333
 000010      52,750.00000000000000000000000
 000110      46,500.00000000000000000000000
 000120      29,250.00000000000000000000000
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-18. Correlated subquery (2 of 3) OD477.0

Notes:

We can look at one department, A00, and find out the salary of each employee. A UNION with a dummy column (' ') is used so that we can see the average salary of A00. You can see that 000010 and 000110 earn more than the average salary of A00.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Next, I want to check the data. While this UNION tells me about the data for a given department, I must determine the average salary for a department in order to compare an employee in that department to that average.

Transition statement — Next, we code a correlated query to answer the question.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Correlated subquery (3 of 3)

IBM i

3. Which employees are paid higher than average salaries in their own department?

```
select workdept, rtrim(firstnme)||' '||rtrim(lastname) as
        name,
         salary from employee e
         where salary > (select avg(salary) from employee
        where workdept = e.workdept)
        order by workdept
```

WORKDEPT	NAME	SALARY
A00	Christine Haas	52,750.00
A00	Vicenzo Lucchesi	46,500.00
C01	Sally Kwan	38,250.00
D11	Irving Stern	32,250.00
D11	Bruce Adamson	25,280.00
D11	Masatoshi Yoshimura	24,680.00
D11	David Brown	27,740.00
D11	Jennifer Lutz	29,840.00
D21	Eva Pulaski	36,170.00
D21	Salvatore Marino	28,760.00
D21	Maria Perez	27,380.00
E11	Eileen Henderson	29,750.00

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-19. Correlated subquery (3 of 3) OD477.0

Notes:

Non-correlated subqueries execute the subquery once as an inner query and use the result to control the rows returned by the outer query.

A correlated query works differently:

1. A row is read by the outer query.
2. A value from that row is passed to the subquery and is used to control which rows are returned by the subquery.
3. The predicate in the outer query is used to determine if the outer table row will appear in the result set.

This process is repeated for each row of the outer table until each one has been examined and either written to the result set or omitted from it.

In the example:

1. The outer query reads a row in the EMPLOYEE table and gets the employee number, last name, salary, and department (WORKDEPT).

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

2. The (work) department for the employee is passed to the subquery. The subquery calculates the average salary for all employees in the work department that was passed from the outer row.
3. The result of the subquery is compared to the individual salary of the employee whose row was read by the outer query. If the individual salary exceeds the result of the subquery, the outer table row is written to the result set; otherwise, it is ignored.

This process is repeated for each row of the outer query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Discuss the solution with the students.

Details — Make sure that the students understand correlated subqueries. When a correlation variable in the FROM clause of the outer query (“e” in the example) is used in the WHERE clause of the subquery, the query is correlated.

In the development of this example, I used a GROUP BY in the first visual because I was using a column function in the base SELECT. I was only interested in the average for each department and only wanted one row per department. In the solution shown in this visual, I am viewing the data from the EMPLOYEE perspective and cannot use a GROUP BY. Using a GROUP BY would not work because the GROUP BY orders the rows based on your group expression and then presents one row or group.

Additional information — Explain what happens when the subquery executes for a given row of the outer query for the EMPLOYEE table. Before the subquery executes, E.WORKDEPT receives the value of the WORKDEPT column for that row. Suppose, for example, that the row is for employee KWAN. Her work department is C01. Therefore, the subquery executed for that row is:

```
(SELECT AVG(SALARY)
        FROM EMPLOYEE
        WHERE WORKDEPT = 'C01')
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with EXISTS predicate (1 of 2)

IBM i

At the beginning of each month, Christine wants to know whether she should schedule a 25th Service Anniversary luncheon.

1. Which employees are celebrating their 25th anniversary this month?

```
select * from employee where year(curdate() - hiredate) = 25
```

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000030	Sally	A	Kwan	C01	4738	1975-04-05
000260	Sybil	P	Johnson	D21	8953	1975-09-11
000330	Wing		Lee	E21	2103	1976-02-23

```
Where CURDATE = March 7th, 2001
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-20. Subquery with EXISTS predicate (1 of 2) OD477.0

Notes:

This example shows how an EXISTS might be used. Suppose that the company president, Christine, schedules a 25th Service Anniversary at the end of every year to recognize all employees who celebrated their 25th anniversary anytime during that year. She only holds the luncheon if there are employees who have celebrated their quarter century of service during that year.

First, we must check to see which employees have celebrated their 25th anniversary. In this case, we have three employees who are eligible.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subquery with EXISTS predicate (2 of 2)

IBM i

2. If there are any employees celebrating their 25th anniversary, produce a list of all current employees to be issued invitations to the luncheon.

```
select lastname, firstnme, empno, hiredate
from employee
where exists
  (select * from employee where year(curdate() - hiredate) = 25)
order by lastname

LASTNAME        FIRSTNME      EMPNO   HIREDATE
Adamson         Bruce         000150  1972-02-12
Brown           David         000200  1966-03-03
Geyer           John          000050  1949-08-17
Gounot          Jason         000340  1947-05-05
Haas            Christine     000010  1965-01-01
Henderson       Eileen        000090  1970-08-15
Jefferson       James         000230  1966-11-21
Johnson         Sybil         000260  1975-09-11
Jones           William       000210  1979-04-11
Kwan            Sally         000030  1975-04-05
Lee             Wing          000330  1976-02-23
:
:
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-21. Subquery with EXISTS predicate (2 of 2) OD477.0

Notes:

The EXISTS predicate contains a subquery and evaluates to true if the result of the subquery contains at least one row. Otherwise, it evaluates to false.

No rows or sets are returned. You will only get a true or false.

The subquery will return a true if there are any employees eligible for the luncheon. In our case, since there are three service anniversaries, a list of all employees to invite is produced.

If the EXISTS clause returns a false, no list of employees would be produced and hence no luncheon would be scheduled that year.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If Christine were to decide to schedule the luncheons on a monthly basis, we would have to change our query slightly. By adding another condition in the WHERE clause of the inner SELECT:

```
select lastname, firstnme, empno, hiredate
   from employee
   where exists
         (select * from employee
                    where year(curdate() - hiredate) = 25 and
                          month(hiredate) = month(curdate()))
   order by lastname
```

on a monthly basis rather than once a year.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Scalar subselect

IBM i

```
select deptno, (select max(salary) from employee
                       where workdept=d.deptno) as maxsal
       from department d where admrdept like 'E%'


DEPTNO         MAXSAL
 E11        29,750.00
 E21        26,150.00
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-22. Scalar subselect OD477.0

Notes:

Another feature is the ability to use a Scalar SubSELECT as shown in the visual. In this example, the result of the subselect is presented as a column of the result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U05** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U05** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U05** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint

IBM i

1. List three quantified predicates that can be used with subqueries.

2. You have the requirement to list all employees with the highest salaries in each of their respective work departments. What kind of subquery can you use?

3. True or False: A subselect may be presented as a column of the result set.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-23. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. List three quantified predicates that can be used with subqueries.

   The answers are ALL, ANY, and SOME.

2. You have the requirement to list all employees with the highest salaries in each of their respective work departments. What kind of subquery can you use?

   The answer is Correlated subquery.

3. True or False: A subselect may be presented as a column of the result set.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Coding subSELECTs*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-24. Exercise OD477.0

Notes:

Perform the workstation exercise *Coding subSELECTs*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Assign the exercise but tell the students they might not have enough time to finish all of the exercise. They can complete it later.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Code subSELECTs
- Code subqueries in predicates

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 6-25. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 7. Query tools

Estimated time

01:30

What this unit is about

This unit describes the use of SQL Query Manager to design and store SQL queries. While Query Manager is a tool that might be used more frequently by users, programmers should be aware of its capabilities and should be able to respond to support questions.

This unit explains how you can use the DB2 for IBM i Query Manager to:

- Create and maintain queries
- Define report forms
- Tailor Query Manager profiles
- Create and run QM procedures

What you should be able to do

After completing this unit, you should be able to:

- Use Query Manager to design queries
- Use Query Manager to design reports
- Use Query Manager to create QM procedures

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercises

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Use Query Manager to design queries
- Use Query Manager to design reports
- Use Query Manager to create QM procedures

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

7.1. Using Query Manager

Instructor topic introduction

In this topic, you can present the visuals or present QM using the IBM i system. If you use the system, make sure you use the sample that is presented in the notebook.

What students will do —

How students will do it —

What students will learn — The students will learn the basics of using DB2 for IBM i Query Manager. Query Manager is a fairly straightforward product. In this topic, we cover some of the common steps that you would follow to create a query and a simple form.

How this will help students on their job —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

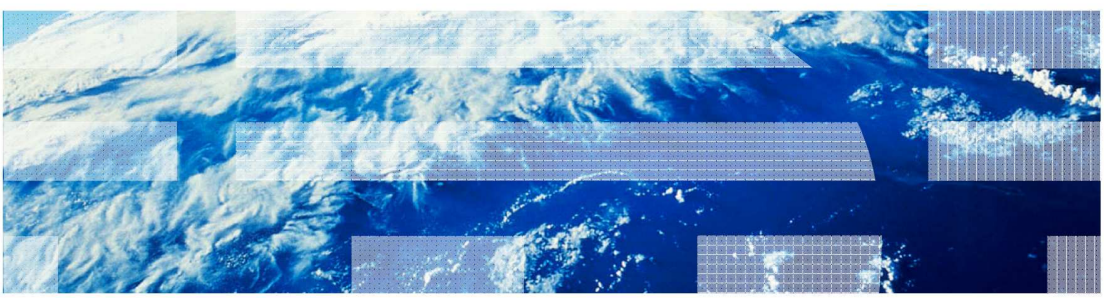


Figure 7-2. Topic 1: Using Query Manager OD477.0

Notes:

This topic presents the basics of using the DB2 for i Query Manager tool.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Features of Query Manager

IBM i

- Write queries
- Design reports
- View reports
- Write procedures

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-3. Features of Query Manager OD477.0

Notes:

Query Manager is part of the licensed program, DB2 for IBM i Query Manager and SQL Development Kit (5770-ST1).

Some SQL users prefer the Query Manager interface to SQL rather than the STRSQL interface. While users of Query Manager need not be familiar with SQL in order to use the tool, someone familiar with SQL will be able to generate more sophisticated queries. Any user of Query Manager must have an understanding of the basic concepts of the DB2 for IBM i database as well as report formatting. With Query Manager, you can write and store queries, design and display reports, and write SQL procedures that execute a series of SQL statements.

Query Manager queries are created using the SQL language. Even those who have limited experience using the SQL language can create and change queries using Query Manager's prompted query creation mode. A series of displays prompt you to enter all of the information that the database manager needs to create a query.

Query Manager queries can obtain information from a single table or from as many as 32 tables. You can select all the table columns, or only a few of the columns, and organize

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

them for the type of output chosen. You can have all table rows in the output, or you can select only a few to be included by using row selection tests. You can also use report breaks and arithmetic summary functions to include additional information in your report, and make it easier to read.

You determine what data the query is to retrieve, the format of the report, and whether the output should be displayed, printed, or sent to another database table.

When you send query output to a database table, you can create a new table. You can also create and maintain database tables using Query Manager's prompt-guided tables option.

In addition, you will find that Query Manager is easy to learn and use.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

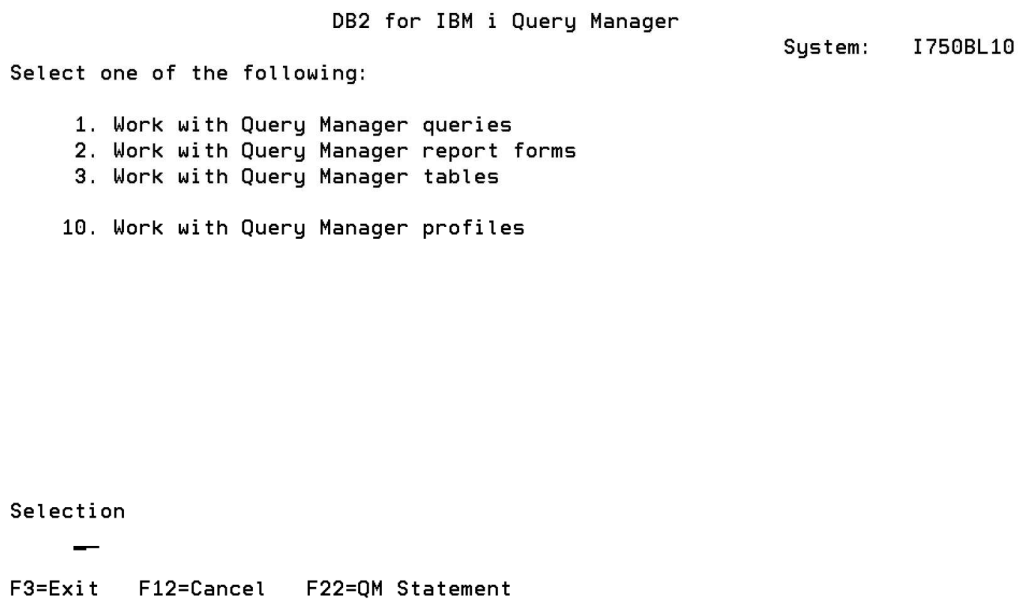
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Start Query Manager: STRQM

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-4. Start Query Manager: STRQM OD477.0

Notes:

The `STRQM` command can be issued from any command line. When Query Manager is started, the menu in the visual is displayed. The following options can be selected from this menu:

- **Work with Query Manager queries**

  Used to create, modify, copy, delete, display, print, rename queries, execute queries, and convert prompted queries into SQL queries.

- **Work with Query Manager report forms**

  Used to create, modify, copy, delete, display, print, rename form definitions, and obtain reports.

- **Work with Query Manager tables**

  Used to create, copy, delete, display, print, rename tables, and modify or add data.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- **Work with Query Manager profiles**

  Used to display, modify, and maintain your Query Manager profile and for setting the default values that Query Manager once it has been started.

  Also note F22 on the display. We can use this key to enter certain Query Manager statements. A complete list follows for your information. We will mention several statements that we will use in class as we encounter them:

```
 1. Connect to a relational database                    CONNECT
 2. Commit a unit of work                               COMMIT
 3. Disconnect a relational database connection         DISCONNECT
 4. Display the results of a query                      DISPLAY

 6. Erase an existing query management object           ERASE
 7. Create an externalized query management object      EXPORT
 8. Create an internal query management object          IMPORT
 9. Print a report or a query management object         PRINT
10. Release a connection to a relational database       RELEASE
11. Run an existing query                               RUN
12. Save query management data, query, or form          SAVE
13. Set the relational database connection              SET
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

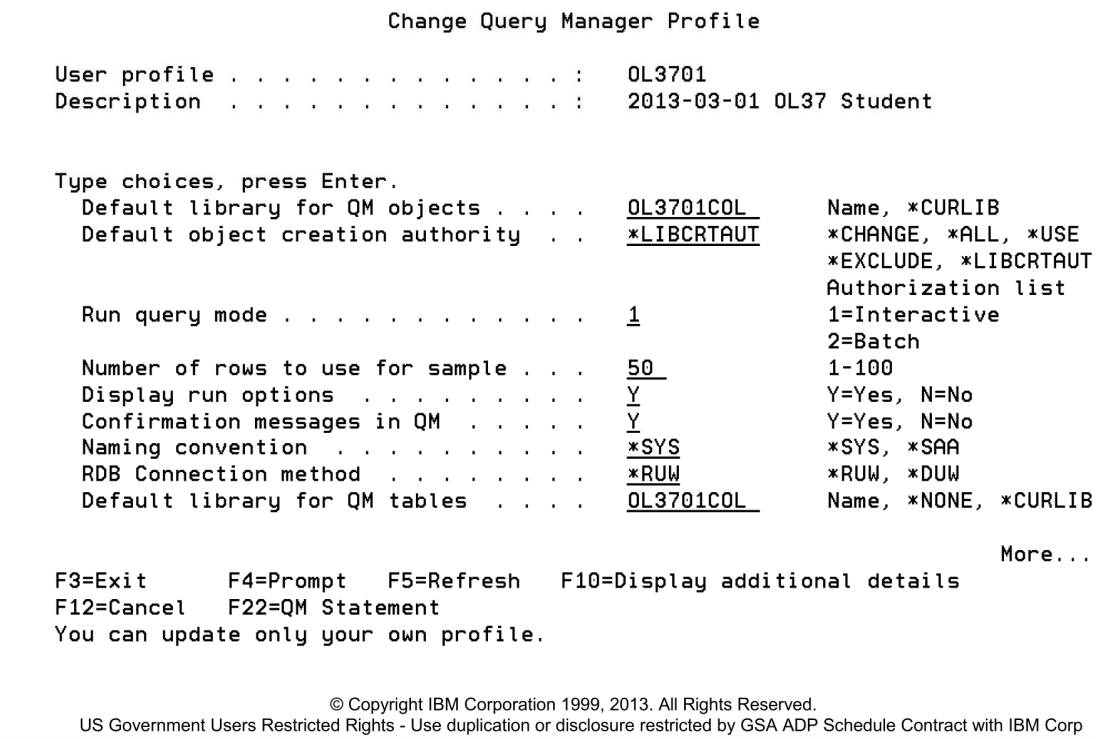
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Query Manger Profile

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-5. Query Manger Profile OD477.0

Notes:

By selecting option 10 from the Query Manager display, you can work with Query Manager profiles.

The Query Manager portion of a user's System i user profile is created the first time the user starts Query Manager, or the first time a security administrator accesses the profile through the Work with display. If you select to work with a System i user profile for which a Query Manager profile does not yet exist, Query Manager creates the Query Manager profile before performing the change, copy, or display.

When new profiles are created for new Query Manager users, the only SQL statement they have the authority to use when creating, changing, or running SQL queries is the SELECT statement.

Unless you have security administrator or higher authority, you will only see your personal QM profile. In your own profile, you can specify you want to run queries as interactive or batch jobs, set your default query creation mode to Prompted or SQL, and set your naming convention to *SAA or *SYS. Many other options are available that you will explore during the workstation exercise.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Note: A Query Manager profile cannot be deleted. If you want to deny complete access to Query Manager for certain users, a security officer can remove their authority to use the `STRQM` command.

The Query Manager profile contains information, such as:

- Query execution mode (interactive or batch)
- Output report routing
- Default library in which to store queries and reports
- Query writing mode (SQL or prompted)

The Query Manager portion of a System i user profile is created with all of the default values specified in the table 3 of the Query Manager reference. You will have an opportunity to review the complete profile in the machine exercise.

There are two types of user access levels available in Query Manager:

- **Basic**

  Basic access grants full access to the Tables option and restricted access to the Queries and Report Forms option. Users can display, print, and run queries and report forms, but cannot create or change them. Most of the values users can change in their Query Manager profiles control the way in which queries and reports are run.

  They cannot access any other user's profile. The Basic access level is suitable for users who want to run queries and reports, and create and use tables.

  Knowledge of the SQL language, databases, and report generation is helpful, but not necessary.

- **All**

  All access grants full access to the Queries, Report Forms, and Table options. Users can change most of the values in their own Query Manager profiles, but they cannot access any other user's profile. The All access level is the default, and is suitable for users who want to create, change, and run queries and report forms.

  Knowledge of the SQL language and report generation is beneficial.

  While All is the default, only a security administrator can change the value of user access. Security administrators (users with *SECADM authority) can maintain the Query Manager profiles for Query Manager users. As a security administrator, you can display the Query Manager profiles of all users for whom you have *READ rights, and can change or copy the profiles of all users for whom you have object management (*OBJMGMT) rights.

References:

- *DB2 for IBM i Query Manager Use*, Working with Query Manager Profiles.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — If possible, teach this topic using the system. That way you can show as much as you want of the QM profiles and you can show the differences in the displays presented to a QM user (sign on as OL3700/OL3700) rather than as a security officer.

Details — We have not covered all the details about the QM profile, but for this class, each of the students should set the following in their profiles:

- The default library for QM objects should be set to OL37nnCOL or *CURLIB.
- The default query creation mode can be set in the QM profile or from the Work with Query Manager Queries display using F19.

Additional information —

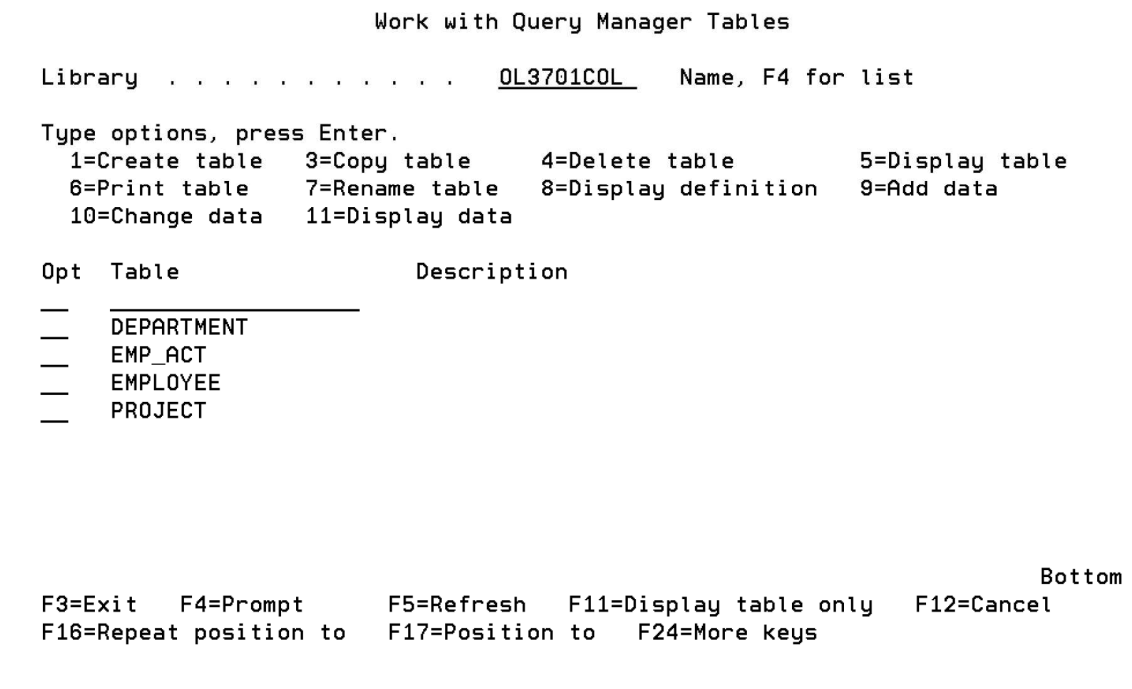
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Work with tables

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-6. Work with tables OD477.0

Notes:

This screen is displayed after selecting option **3** from the main **QM** menu. The collection used is the current library which can be overridden. All tables are displayed on the screen. Note that you cannot work with views on this screen.

Notice that many operations can be performed:

1=Create table

3=Copy table

4=Delete table

5=Display table

6=Print table

7=Rename table

8=Display definition

9=Add data

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

10=Change data

11=Display data

We will not discuss these in detail in this class. You can try some of these options during the workstation exercise.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — We illustrate the functions available in QM by using it to develop several SQL queries and to create and modify report forms. First, we work with QM Queries.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Work with Query Manager Queries

IBM i

```
                        Work with Query Manager Queries

Library  . . . . . . . . . . .   OL3701COL    Name, F4 for list
Query creation mode  . . . . :   SQL

Type options, press Enter.
  1=Create   2=Change   3=Copy   4=Delete   5=Display   6=Print   7=Rename
  9=Run      10=Convert to SQL

Opt  Query       Type      Description
1    HIGHSAL
__   QMLAB       SQL

                                                                    Bottom
F3=Exit   F4=Prompt      F5=Refresh   F11=Display query only   F12=Cancel
F16=Repeat position to   F17=Position to   F24=More keys
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-7. Work with Query Manager Queries OD477.0

Notes:

This panel is displayed after you select option **1** on the QM main panel.

Let's look at how to create a query in SQL or in Prompted mode. Then, we will see how to generate a report and change its format.

A new query can be created by selecting option **1** from the panel shown above. In this example:

- Notice that the Query Creation Mode is SQL based on the Query Profile. We could press **F19** to toggle the value to prompted.
- For SQL mode, you can write SQL instructions for which you have been given authority, as defined in your Query Manager User Profile. This mode of query definition is meant for users who already know SQL.
- Whether you use SQL or Prompted mode, work is significantly simplified by using the **F1** (Help) or **F4** (Prompt) keys as we did with interactive SQL mode.
- **F1** displays help information. By positioning the cursor on a prompt or a message, more detailed information is provided.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- **F4** displays a prompt window containing additional information on the current prompt. **F4** provides different results, depending on the cursor position. If pressed when the cursor is on a value input prompt for a parameter, it will display a list of possible values for that parameter. If it is pressed when the cursor is on the QM or system command line, then it displays a list of the supported QM or system commands.
- To create a new query, we would enter option **1** on the top line and enter the name for our query, `highsal`.
- Once we are finished this step, we press **Enter** to continue.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Edit Query

IBM i

```
                                  Edit Query
Columns . . . :   1 70                                  Query  . . . : HIGHSAL
QM . .  ____________________________________________________________________

Type SQL Statement
        ************************** Beginning of Data **************************
'''''''  select workdept, rtrim(firstnme)||' '||rtrim(lastname) as name,
'''''''       salary from employee e
'''''''       where salary > (select avg(salary) from employee
'''''''                       where workdept = e.workdept)
'''''''   order by workdept
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
        ****************************** End of Data ****************************

F2=Alternate keys   F3=Exit   F4=Prompt    F5=Run report   F6=Run sample
F9=Retrieve         F15=Check syntax       F24=More keys
(C) COPYRIGHT IBM CORP.
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-8. Edit Query OD477.0

Notes:

You can enter free-form SQL statements in this workspace. You can prompt any statement or clause of a statement using **F4**, as you did with interactive SQL. In the visual, we show you the completed SQL statement.

You can insert comments to describe your query. Identify comments by beginning them with a double hyphen (--). Everything from the double hyphen to the end of that line is treated as a comment and is stripped from the SQL statement during processing. A comment can start at any point in the line.

A helpful feature of Query Manager is the ability to syntax check the SQL statement using **F15**.

There are many other function keys available. Notice that you can run the query using F5 or F6. You can run the query that you are currently creating or changing from the Edit Query display and from most of the Define Prompted Query displays.

Press **F5** (Run report) if you want to run a full query, or **F6** (Run sample) to run the query using the sample restriction.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

When you run an SQL SELECT statement query or prompted query, you can choose to retrieve from the database either all of the rows specified by the query, or a limited number of rows (a sample). You set the sample number of rows you want to retrieve (between 1 and 100) through your Query Manager profile. The default value is set to 50 rows.

If you press **F5** or **F6** from the Edit Query panel to execute the query, a panel is displayed that will allow you to specify the form to be used in producing the report.

In this case, the default form (*SYSDFT) is used.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — The difference between using F5 or F6 is that F5 defaults to RUN SAMPLE ONLY *NO whereas F6 defaults to RUN SAMPLE ONLY *YES. RUN SAMPLE ONLY *YES allows you to run sample queries and reports only if Allow Interactive Sample Run is set to Yes in your Query Manager profile. The SQL SELECT statement query or prompted query is run with the sample option and the data is limited to the maximum number of rows specified in your profile for a sample. The query is run interactively, regardless of your Run Query Mode setting. The output can only be sent to “display.”

In the exercises, since we have small tables, you will see no difference between F5=Run sample and F6=Run report.

Additional information —

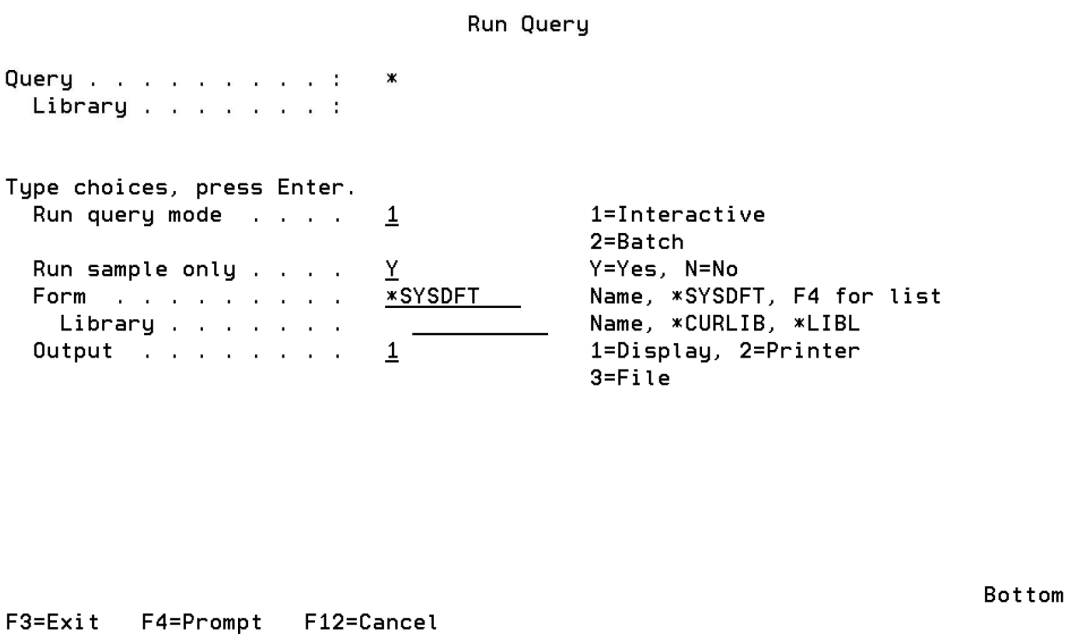
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Run Query

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-9. Run Query OD477.0

Notes:

After pressing **F6**=Run Sample, you will see this display. The * means that the system will use the current query and current report form associated with that query to produce your output. Note that the run query mode is interactive by default. You can change it to run any query in batch. Also, there is a parameter to permit you to send the output of any query to your display, your OUTQ, or a file.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

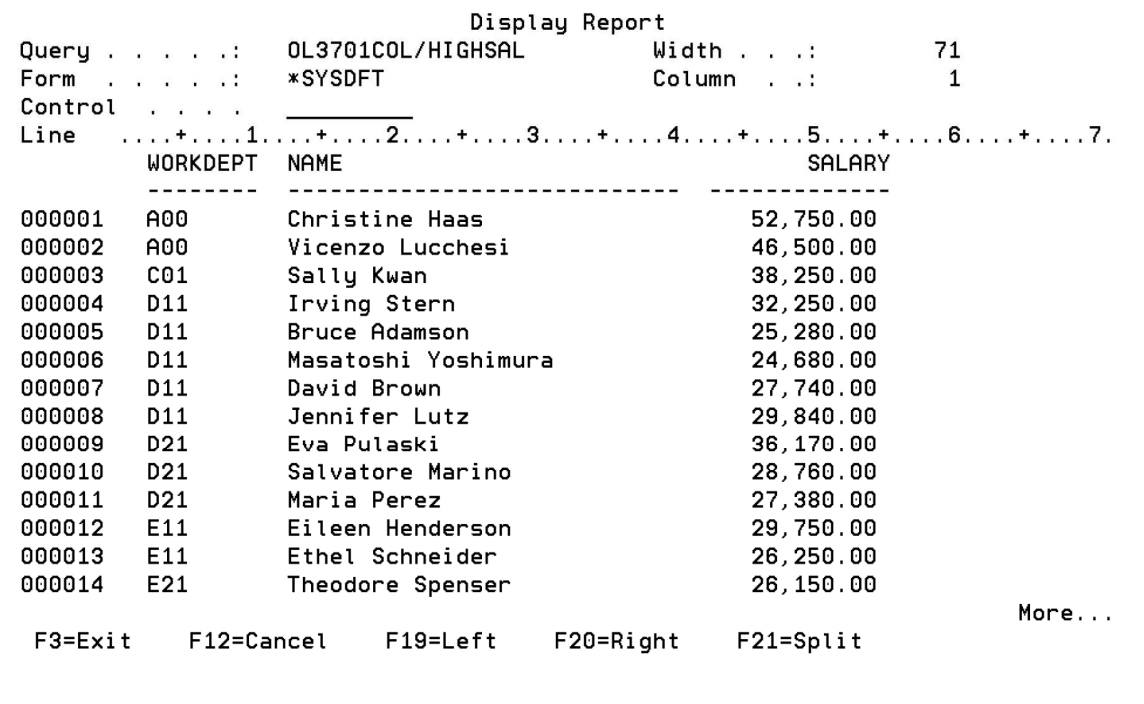
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Display Report

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-10. Display Report OD477.0

Notes:

The query results are displayed in report form using the column name or label as headings. Columns are edited based on their type, character, numeric, or date. The editing might not be what we want. Perhaps we want to change the order in which some columns are displayed.

Pressing **F3** or **Enter** will return you to the query definition after taking the prompt to save the active data. QM uses active data to display the report form. Therefore, you do not have to re-execute the query every time you modify the report form.

Now that we are satisfied with the data we have extracted using SQL, let us work with the format of the output.

After returning to the query, press **F13** to display the formatting options for the report. Then press **Enter** and the screen above is displayed.

As an example, we will format the columns of the report.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — We do not show each and every screen that is displayed. Our purpose is to give the students an appreciation of the capabilities of Query Manager so that they can investigate it further on their own systems.

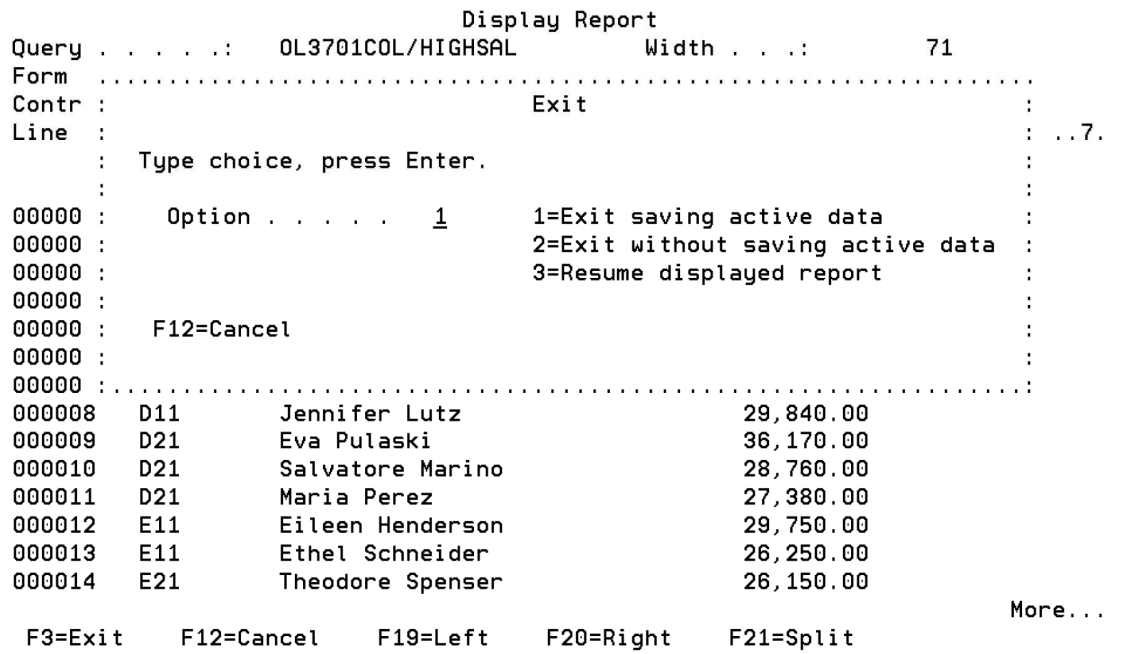
Additional information —

Transition statement — Next we make the output report easier to read.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exit by pressing either F3 or Enter

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-11. Exit by pressing either F3 or Enter OD477.0

Notes:

When you exit your query report using F3, you see a display that overlays your report. You have several options:

1. **Exit saving active data**

   Choose this option when you want to finish running the query and save the retrieved data as active data. You need to wait for the query to complete running before you return to the display from which you chose to run the query or report.

   Active data is created when an SQL SELECT statement query or prompted query is run interactively. Some queries take a long time to run; for example, queries that are complex or retrieve a large number of rows from the database. If you do not require full active data, you can save time by running these queries using the sample restriction, and create sample active data.

2. **Exit without saving active data**

   Choose this option to immediately return to the display from which you chose to run the query or report. The query does not complete its run and no active data is saved.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

3. **Resume displayed report**

   Choose this option to return to the Display Report display, where you can continue to view the data retrieved by the query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

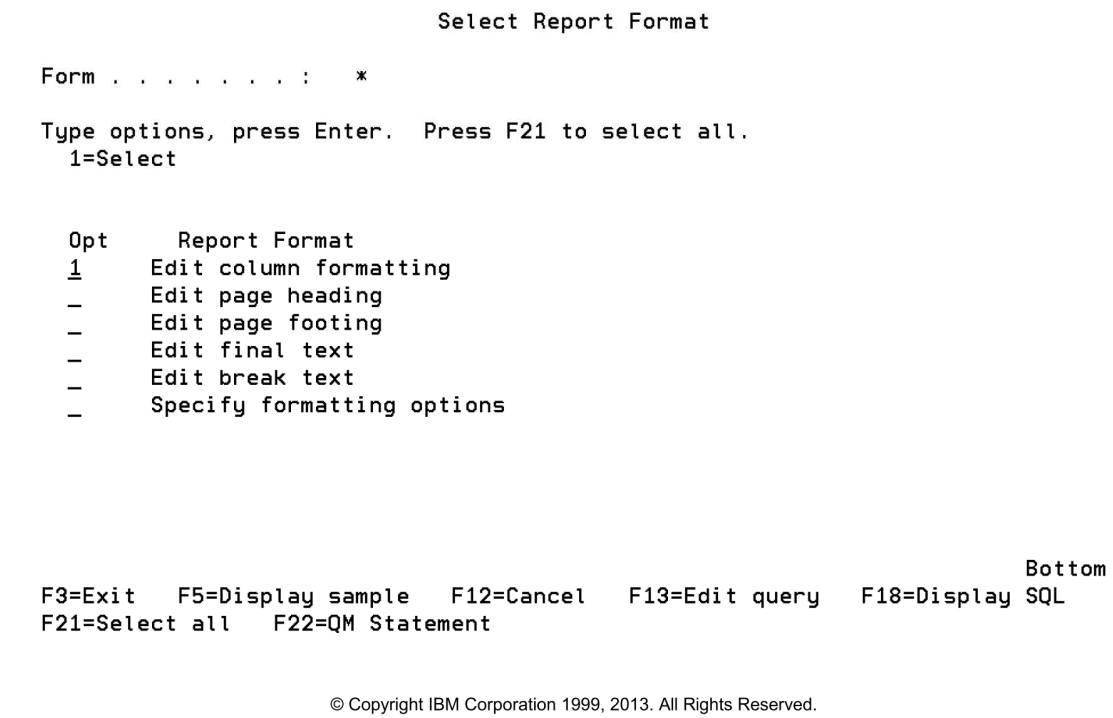
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

F13 Edit form > Edit column formatting

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-12. F13 Edit form > Edit column formatting OD477.0

Notes:

Press **F13 = Edit form** to bring up this display.

Using this display you can tailor the report output. You can change such things as column headings, add page headers and footers, and add text for subtotal breaks and final totals.

Press **Enter** to move to the visual on the next page.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

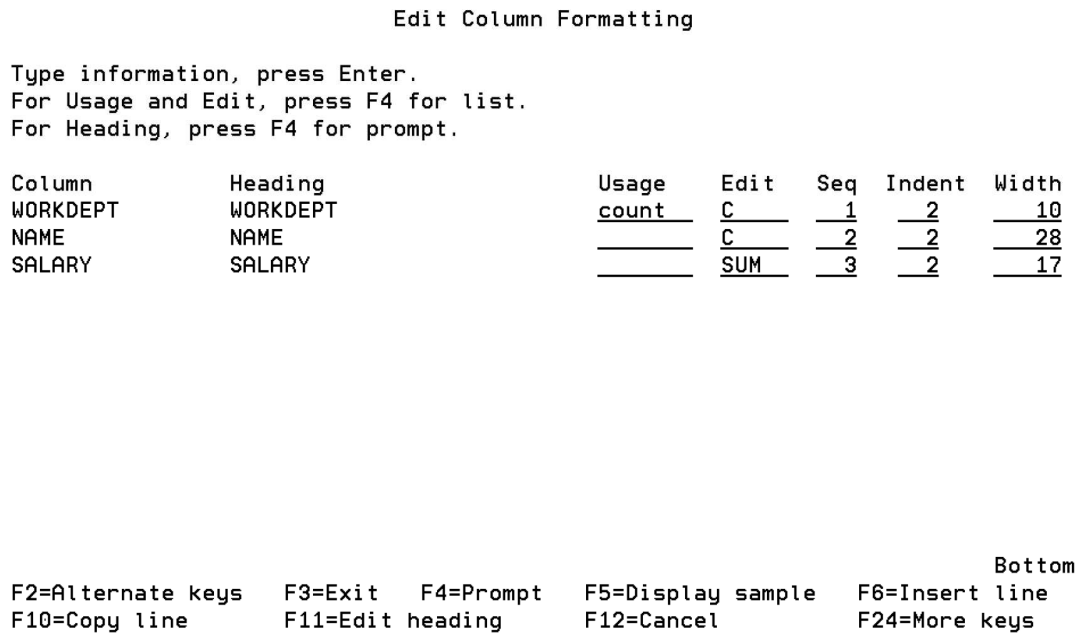
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Edit Column Formatting

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-13. Edit Column Formatting OD477.0

Notes:

You can specify the following information through this display:

- The heading text for each column.
- Any special usage to apply to a column, such as report breaks and summary operations. Place the cursor in the Usage column and press **F4** to view all available options.

  For example, we might want to calculate the total annual salary by specifying SUM for SALARY. Or, we could calculate averages, maximums, and minimums (AVG, MAX, MIN functions) for the salary. Breaks at various levels can be specified (BREAK1,...BREAK6). It is also possible to drop a column from the report - OMIT clause. It is possible to count the number of not-null values in a column (COUNT clause).
- The data type and format of the data in each column. For example, a format can be specified for the date and time or timestamp columns. You can edit numeric columns to

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

display commas (or periods as separators) and currency symbols. For these fields, **F4** provides help by listing and explaining all the valid edit codes.

- The order in which the columns are to appear (left to right).
- The spacing between columns.
- The width of each column.

Press **Enter** to move to the visual on the next page.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

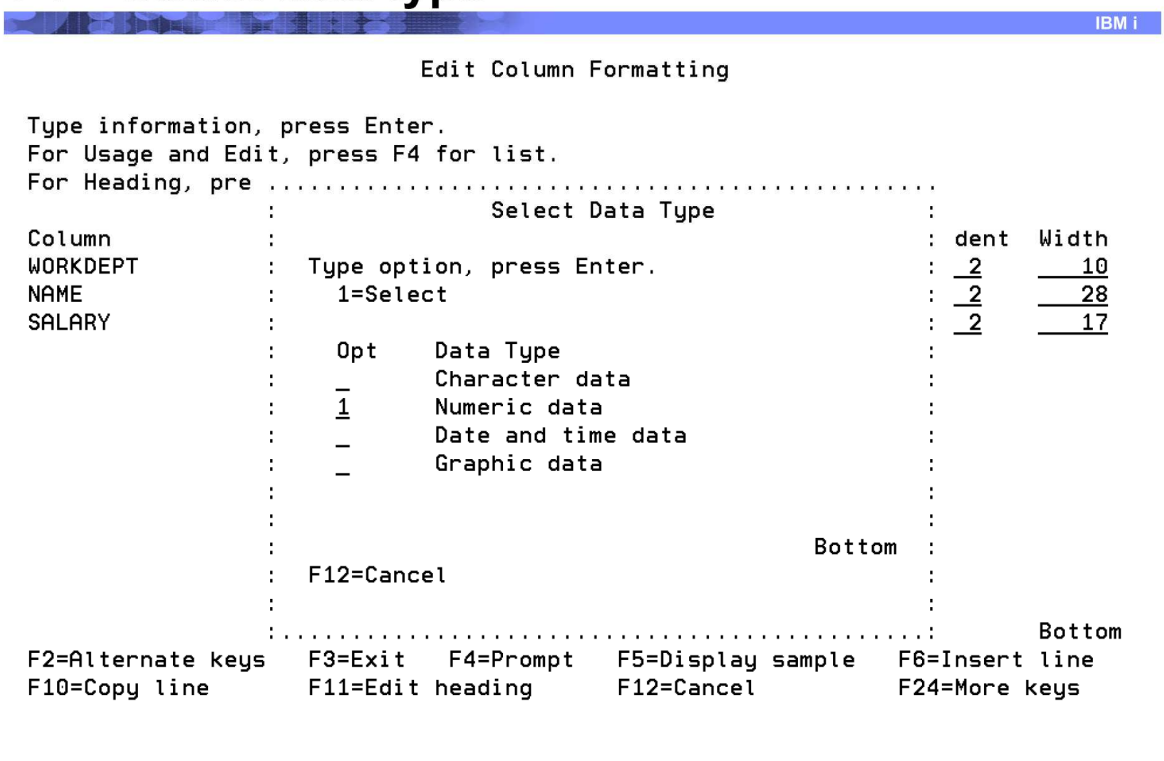
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

F4 > Select data type



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-14. F4 > Select data type OD477.0

Notes:

For the specific data field you want to change (in this example, SALARY), place the cursor under the Edit column then press **F4**.

While working within the Edit function, you can enter the data type. By default, the selected data type is based on the column's definition. All edit codes may be selected using F4.

Press **Enter** to see the display on the next page.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — I would not recommend changing the data type. If the column is selected from an existing table, the QM product will pick up the data type and display it. If you change it and specify an incompatible edit code, you will see the error, `Edit code for column n not valid`.

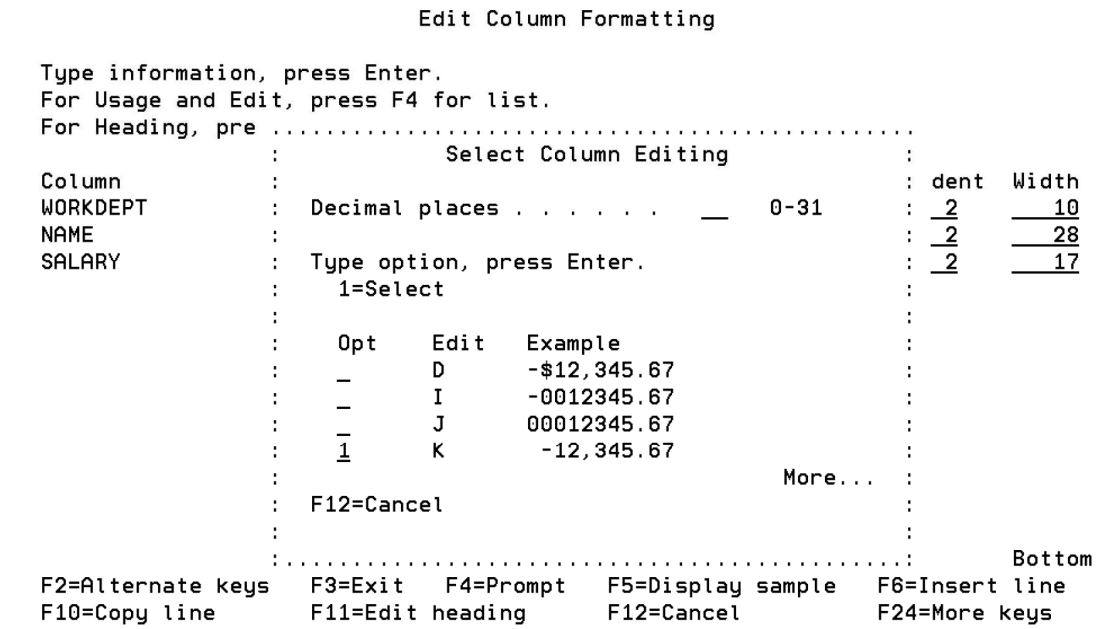
Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Select Column Editing

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-15. Select Column Editing OD477.0

Notes:

For the various data types, certain editing options are available, many of which look familiar to RPG programmers. Although, you will notice that some codes are different. You select one from the list so it is not important to know the codes precisely. Once you have designed a few forms, you will quickly get to know the codes. For example, on this display, you are editing the column format, setting it for numeric data type, and setting the number of decimal spaces used. The `K` sets the numeric format to `-12,345.nn` and the decimal places field determines how many decimal places are used in the column. `K3`, for example, could result in a numeric edited column value of the form, `-12,345.670`, being displayed.

You can use the prompting feature to specify the edit code, or once you are familiar with the edit codes, enter the one you want concatenated with the number of decimal places.

Press **Enter** to return to the Edit column formatting screen.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Point out where you fill in the edit code you want to use. Point out two things for a numeric column:

1. Select an edit code. Enter a `1` beside the `K`, for example.
2. Select the number of decimals. If the column has three decimals but you want to show two, you make the change where shown on the display. The decimal place is not shifted left or right. You are only changing how many places are displayed.

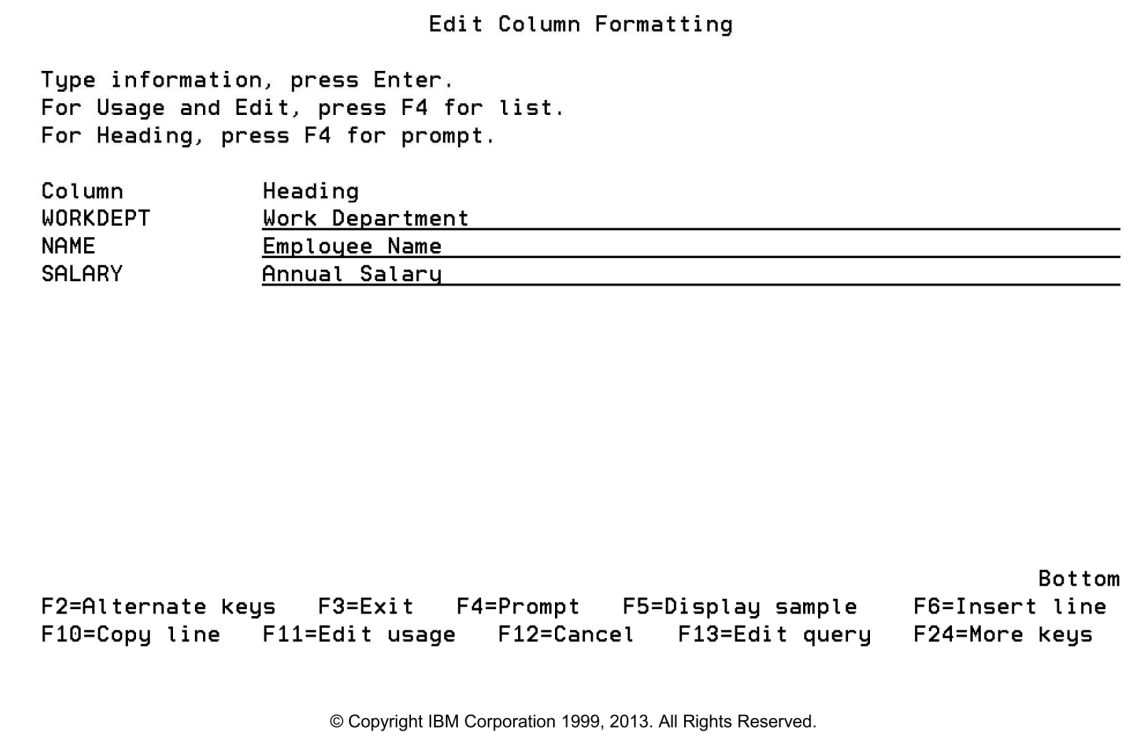
Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Edit Column Formatting

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-16. Edit Column Formatting OD477.0

Notes:

From the Edit Column Formatting screen, press **F11**=Edit heading to display this screen.

Various functions can be applied to columns. In our example, we want a COUNT of all the rows (based on the WORKDEPT column) and the SUM of the SALARY column (that is, total salary). We could specify subtotals at the WORKDEPT level if we wanted. If we did this, we would see both subtotals and grand totals.

Press **Enter** to return to the Select Report Format screen.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

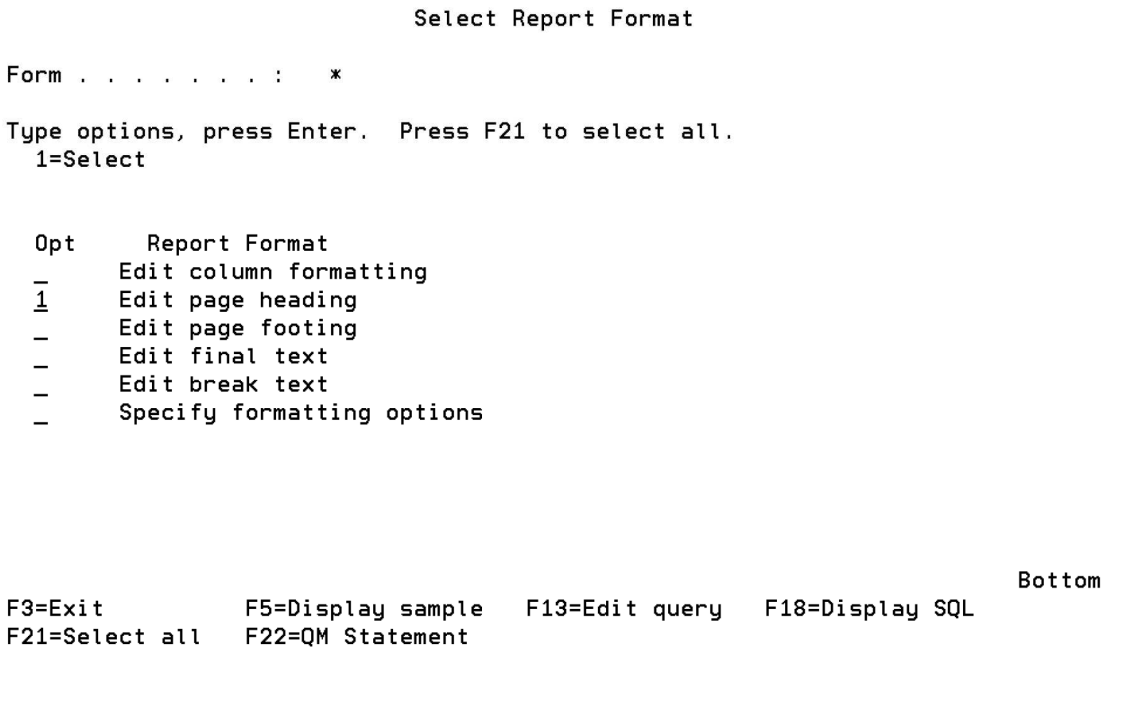
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Select Report Format > Edit page headings

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-17. Select Report Format > Edit page headings OD477.0

Notes:

On the Select Report Format screen, select the option for Edit Page Heading and press **Enter**.

For each column, you can modify the headings displayed. When you want to split the heading over two or more rows, use an underscore (_) to designate where to split the heading.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

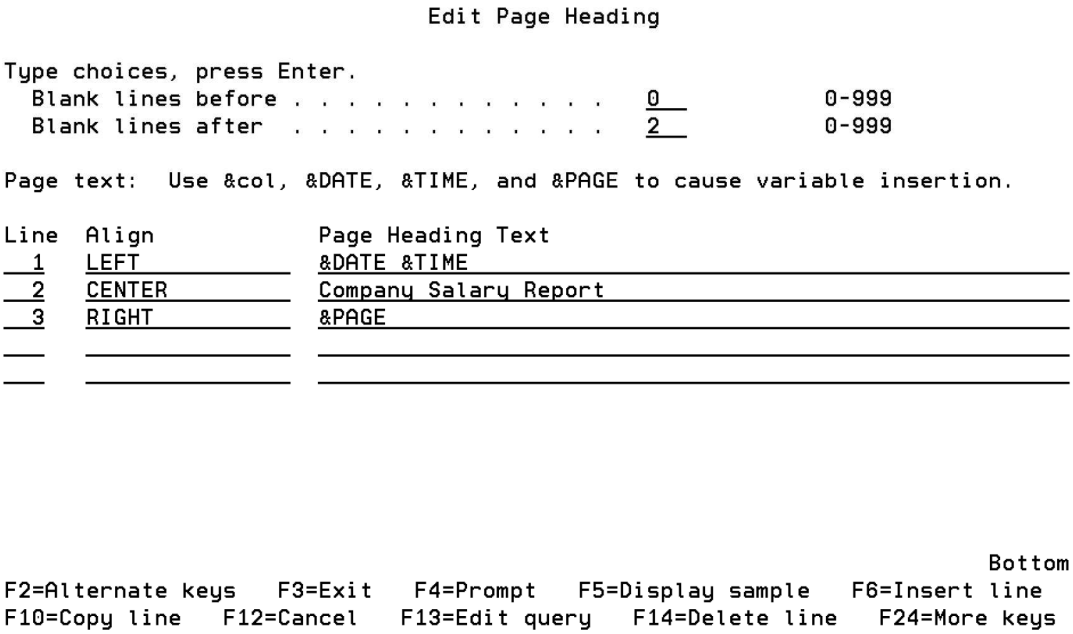
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Specify Page Heading

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-18. Specify Page Heading OD477.0

Notes:

You can also tailor your output so that the date and time appear by using the Query variables, *&DATE*, *&TIME*, and *&PAGE*.

Page headings text can be specified and, as you can see, you can specify where the text is to appear.

The headings (and footings) will appear only when you send your output to an OUTQ rather than display or file.

With little additional effort, it is possible to create a professional looking report with a title. You can also create date and page numbering and improved headings.

The headings at the beginning of the report are discussed, but similar conditions apply to the footings of the report with the Edit Page Footing option.

Lines in the heading can be added, modified and copied, or deleted by using F6, F10, F14 respectively.

The number of blank lines preceding and following the heading can be specified in Blank lines before and Blank lines after. A value of 2 is the default.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Headings can be centered or aligned right or left by using CENTER, RIGHT, and LEFT.

Headings can contain the following variables:

- *&n*

  Inserts the value of the nth column of the first row. This can be useful when you have subtotals and breaks. Note that the sequence of the columns in the report is the same as the sequence in which the columns are extracted by the query, not the sequence specified in the Edit Column Formatting.

- *&DATE*

  Inserts the current date in the report.

- *&TIME*

  Inserts the current time in the report.

- *&PAGE*

  Inserts the current page in the report.

Note: *&DATE*, *&PAGE*, and *&TIME* variables must be entered uppercase. Lowercase is treated as a literal.

A literal value can be included with these variables. To produce the page heading in the visual, we coded `&PAGE`. We could have coded `Page &PAGE`.

When you are done specifying your page heading information, press **Enter** then the F3+Exit.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information — The one thing you have to remember when designing these reports is that your printed output must fit within the width supported by the printer that will produce the output.

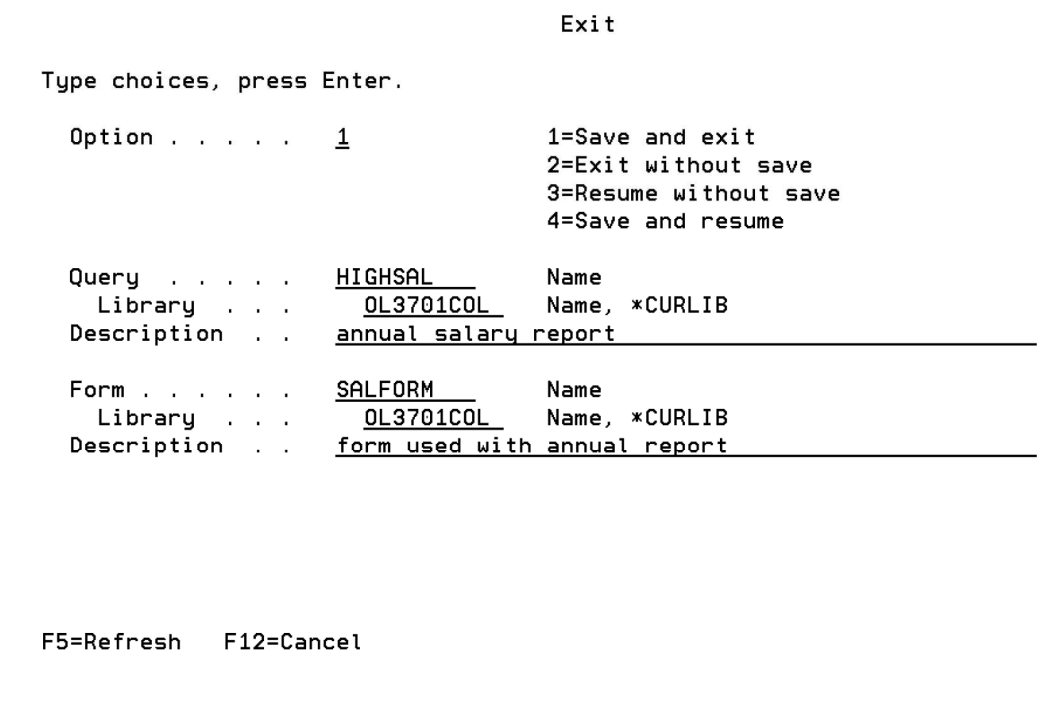
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

F3 > Save completed form

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-19. F3 > Save completed form OD477.0

Notes:

Now that you have designed your form, you need to save it.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

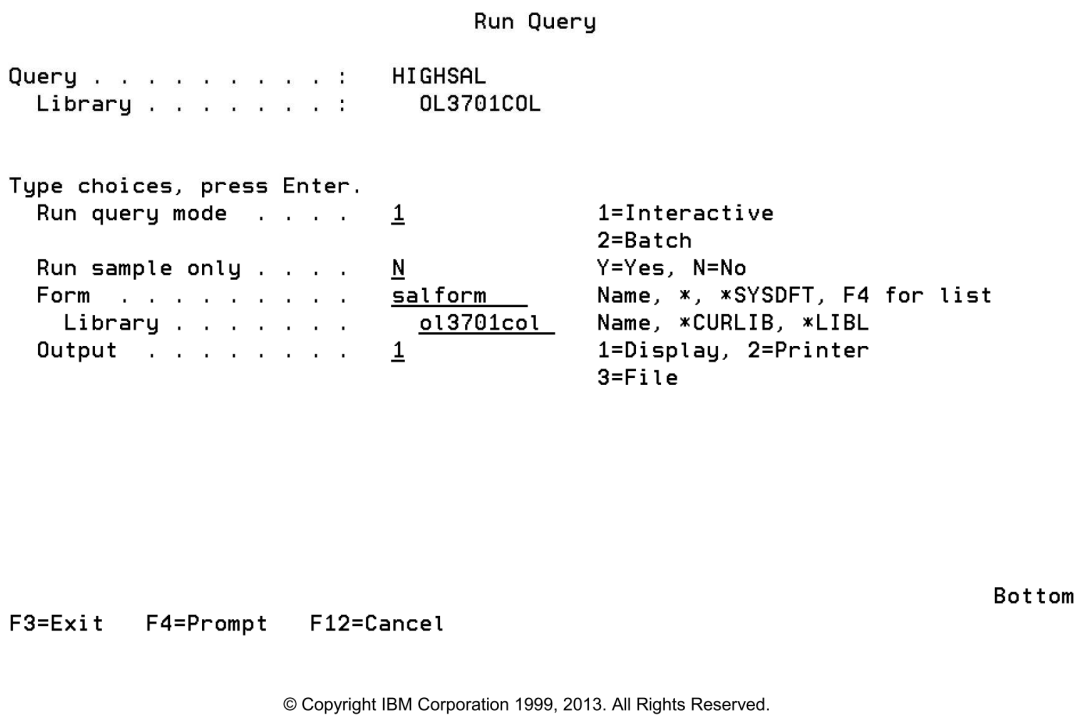
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Option 9 = Run Query

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-20. Option 9 = Run Query OD477.0

Notes:

You can now run your report and test your form. There are many ways to run the query. You can use a function key from many of the Query Manager displays or you can press **F22** and then specify Run query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Query final result

IBM i

```
                              Display Report
Query . . . . .:   OL3701COL/HIGHSAL         Width . . .:        71
Form  . . . . .:   OL3701COL/SALFORM         Column  . .:         1
Control . . . .    ________
Line    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
          Work Depar  Employee Name                    Annual Salary
          ----------  --------------------------  -----------------
000001    A00         Christine Haas                       52,750.00
000002    A00         Vicenzo Lucchesi                     46,500.00
000003    C01         Sally Kwan                           38,250.00
000004    D11         Irving Stern                         32,250.00
000005    D11         Bruce Adamson                        25,280.00
000006    D11         Masatoshi Yoshimura                  24,680.00
000007    D11         David Brown                          27,740.00
000008    D11         Jennifer Lutz                        29,840.00
000009    D21         Eva Pulaski                          36,170.00
000010    D21         Salvatore Marino                     28,760.00
000011    D21         Maria Perez                          27,380.00
000012    E11         Eileen Henderson                     29,750.00
000013    E11         Ethel Schneider                      26,250.00
000014    E21         Theodore Spenser                     26,150.00
                                                                  More...
 F3=Exit     F12=Cancel     F19=Left     F20=Right     F21=Split
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-21. Query final result OD477.0

Notes:

This is our improved report.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — Next, we cover another feature of Query Manager: Procedures.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

7.2. Using Query Manager procedures

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

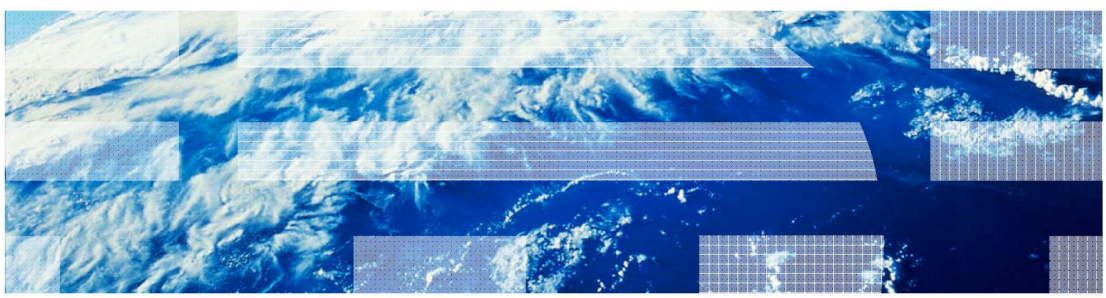


Figure 7-22. Topic 2: Using Query Manager procedures OD477.0

Notes:

Query Manager procedures are QM commands that are saved in and executed from source members. An example follows.

© Copyright IBM Corp. 2010, 2013
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Query Manager procedures

IBM i

- Set of QM commands enclosed in single or double quotes
- Create and save as member (type **TXT**) in any source PF
- Do not compile
- Execute from QM command line

```
 Columns . . . :    1  71            Edit                    OL3701COL/QPRCSRC
 SEU==> ____________________________________________________________    SALPROC
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
        *************** Beginning of data ************************************
0001.00 'RUN QUERY OL3701COL/HIGHSAL (FORM=SALFORM DISPLAY=NO'
0002.00 'SAVE DATA AS OL3701COL/TEMP
0003.00 'PRINT REPORT (WIDTH=80'
        ****************** End of data ***************************************
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-23. Query Manager procedures OD477.0

Notes:

Let us assume that accounts payable data is updated weekly and that a report on the latest update is required every Monday. To avoid having to write and execute a query and a form every time, we can create a *procedure* to invoke the query and use the appropriate form.

Procedures are created and changed outside Query Manager, often using the source entry utility (SEU) editor. The text must be entered in uppercase only. If there is any lowercase text, there will be errors when you run the query.

In the example, the procedure SALPROC contains the following commands:

- Execute query `HIGHSAL` using form `SALFORM`
- Save the data produced by the query
- Print the results

From this point in time forward, by simply issuing the command, `RUN SALPROC`, from inside Query Manager, all the commands contained in the query SALPROC will be executed automatically.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

A procedure allows you to run a set of query commands with a single `RUN` command. You can create a procedure for a common program section and then call that procedure with a single command rather than a series of commands.

The following rules apply when you are creating a procedure:

- Procedures are source file members.
- Procedures can contain Query Management commands and blank lines. Blank lines have no effect on the processing of the commands. They can optionally contain an H record and a comment (V) record. The comment record can be used as a text descriptor when importing a procedure.
- Each Query Management command must be in uppercase English letters.
- All commands must be surrounded by apostrophes or quotation marks. If the command contains a quotation mark, the internal quotation marks are represented by two successive apostrophes (') or quotation marks (“”).
- Procedures can contain a `RUN` command that runs another procedure or query.
- A single command is limited to 79 characters on a line.
- The width of a procedure line is limited to the source file record width.
- The width of a query command on a procedure line after procedure parsing is done is limited to 256 characters. Procedure parsing involves stripping leading and trailing blanks and reducing apostrophes and quotation marks.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Focus on the fact that the source must be entered in capital letters and that each individual statement must be enclosed in single or double quotes.

Point out the meaning of each statement. We have not spent any time on QM commands, but here are three:

- `'RUN QUERY OL3701COL/HIGHSAL (FORM=SALFORM DISPLAY=NO'`

  We run a specific query stored as a source member. When you run a QM query interactively, by default, it displays the result. We want to produce a report, so we specify `DISPLAY=NO`.

- `'SAVE DATA AS OL3701COL/TEMP`

  Normally, when you run interactively, you are prompted as to whether you want to save the results of the query. We select **yes** in this statement and specify the name of a table where the data is to be stored. Yes, the quote at the end of the above statement has been intentionally omitted.

- `'PRINT REPORT (WIDTH=80'`

  This command produces the spool file member.

Additional information —

Transition statement —

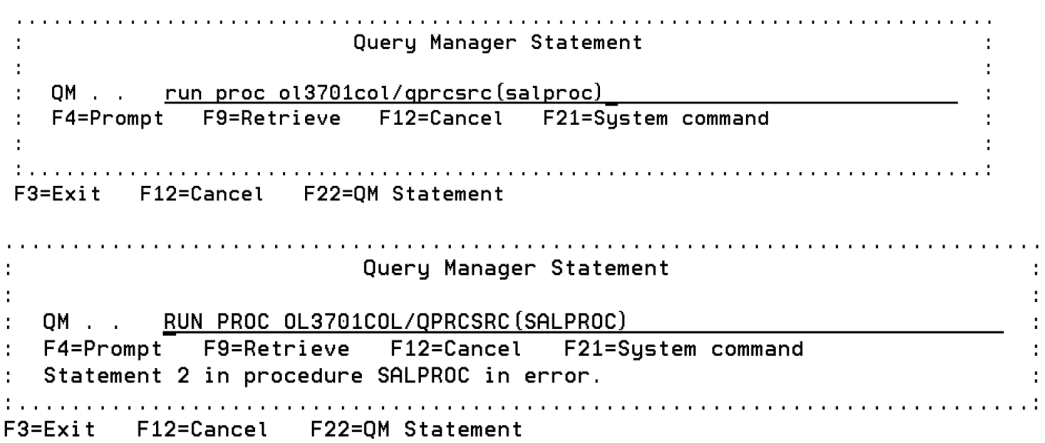
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Run Query Manager procedure

IBM i

- RUN PROC instead of RUN QUERY
- Can *only* execute from QM command line
- QM commands in source member are parsed at run time.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-24. Run Query Manager procedure OD477.0

Notes:

QM procedures can only be processed by Query Manager, so you must use STRQM to run them. This statement causes the procedure we created in the previous visual to be executed. Remember that these procedures are parsed, or interpreted, at run time.

If there are any errors detected, you will be notified in the QM command line window.

If you view the job log, you will find specific information about the error. For example:

```
Procedure statement contains unmatched apostrophes or quotes.
Statement 2 in procedure SALPROC in error.
RUN PROC command ended due to error.
```

To correct the problem, we would add the missing quote to statement 2.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

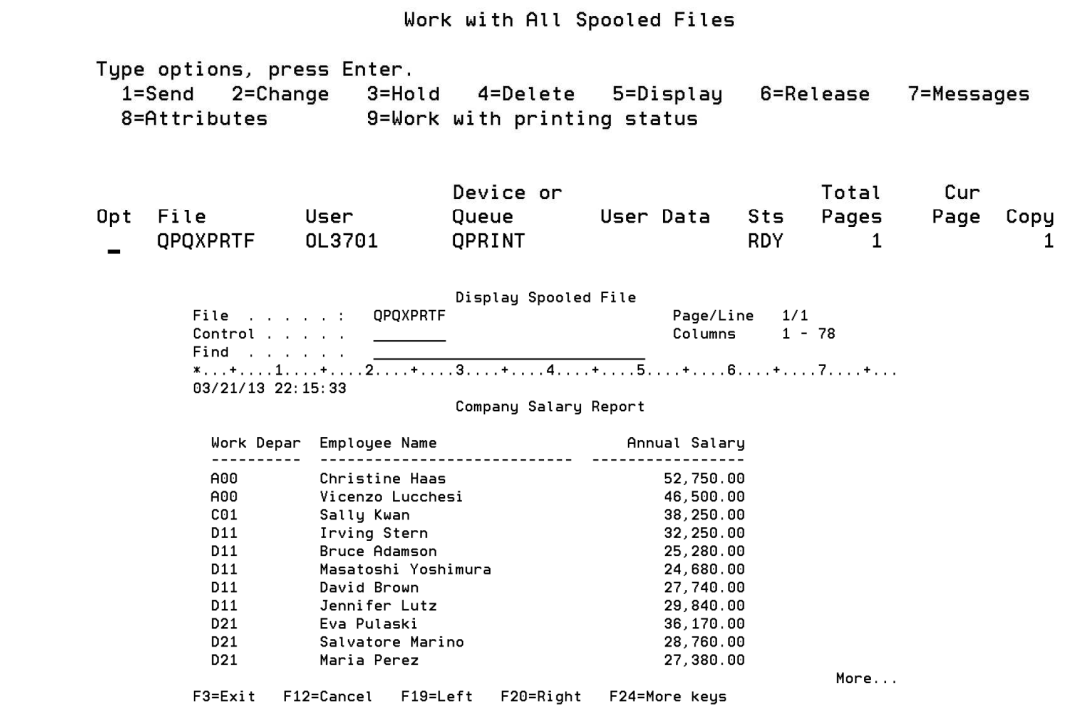
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Display your output

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-25. Display your output OD477.0

Notes:

Our example is complete as we illustrate in the visual the display of the spooled output from our query.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Other QM features

IBM i

- Commands
- Using CL commands with QM queries
- Variables
  - SELECT * from &TABLE

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-26. Other QM features OD477.0

Notes:

Query Manager offers much more than we have seen so far. There are many more commands available to you beyond **RUN**, **SAVE DATA AS**, and **PRINT**. A complete list follows. Additional information can be found in the reference document.

COMMIT ends a unit of recovery and commits database changes that the unit of recovery made.

- **CONNECT** changes the database that is associated with the query instance.
- **DISCONNECT** destroys one or more connections.
- **ERASE** removes a FORM, PROC, QUERY, or TABLE from the database.
- **EXIT** stops your application's session with Query Management and ends the associated instance of Query Management in your process.
- **EXPORT** creates a file containing the contents of certain Query Management objects.
- **GET** gets the value of a Query Management variable and provide it to a user program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- **IMPORT** copies a file containing an exported object into one of the following Query Management objects: FORM, PROC, or QUERY.
- **PRINT** prints a hard copy listing of an Query Management object.
- **RELEASE** places one or more connections in the released state.
- **RUN** processes a PROC or QUERY.
- **SAVE DATA AS** saves data in a table in the database.
- **SET CONNECTION** changes the state of a connection from dormant to current.
- **SET GLOBAL** sets the value of an Query Management variable from the user program or procedure.
- **START** provides an interface to start an instance of Query Management.

The following is a list of the CL commands that perform various QM functions:

- **ANZQRY** Analyze Query
- **CRTQMFORM** Create Query Manager Form
- **CRTQMQRY** Create Query Manager Query
- **DLTQMFORM** Delete Query Manager Form
- **DLTQMQRY** Delete Query Manager Query
- **RTVQMFORM** Retrieve Query Manager Form
- **RTVQMQRY** Retrieve Query Manager Query
- **STRQMPRC** Start Query Manager Procedure
- **STRQMQRY** Start Query Manager Query
- **WRKQMFORM** Work with Query Manager Form
- **WRKQMQRY** Work with Query Manager Query

Variables can be incorporated in QM queries. So far, you are aware of **&n**, **&DATE**, **&TIME**, and **&PAGE**.

Variables can be referred to by specifying the variable name within an SQL query or a user program using the **GET GLOBAL** command. When a variable name is referred to within an SQL query, the variable name must be prefixed with an ampersand (&) in order for Query Management to recognize it as a variable.

In the visual, the value **&TABLE** is considered Query Management variable.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: Query Manager is part of IBM i/OS.

2. True or False: Query Manager queries are SQL-based.

3. True or False: Query Manager output can be directed to the display, printer, or to another database table.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-27. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: Query Manager is part of IBM i/OS.

   The answer is False.

2. True or False: Query Manager queries are SQL-based.

   The answer is True.

3. True or False: Query Manager output can be directed to the display, printer, or to another database table.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Using SQL Query Manager*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-28. Exercise OD477.0

Notes:

Perform the workstation exercise *Using SQL Query Manager*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Use Query Manager to design queries
- Use Query Manager to design reports
- Use Query Manager to create QM procedures

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 7-29. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 8. Database maintenance

Estimated time

03:30

What this unit is about

This unit discusses how to create tables and views and how to describe the columns and rows within them.

We also discuss other DB2 for IBM i functions such as referential integrity and check constraints and how they apply to SQL.

What you should be able to do

After completing this unit, you should be able to:

- Use the CREATE statement to build a table, view or index
- Describe how referential integrity and check constraints are supported within SQL
- Use ALTER TABLE to add, drop, or alter a column
- Use the GRANT and REVOKE statements to limit access to SQL data
- Use the DROP verb to delete a table, view, or index
- Define the data elements in a table or view
- Use STRSQL and System i Navigator to create SQL objects
- Use INSERT, UPDATE, ALTER, and DELETE statements to maintain a database

How you will check your progress

Accountability:

- Checkpoint questions
- Workstation exercises

The students perform an exercise to create a schema, create a table, and create an index over that table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Use the CREATE statement to build a table, view, or index
- Describe how referential integrity and check constraints are supported within SQL
- Use ALTER TABLE to add, drop, or alter a column
- Use the GRANT and REVOKE statements to limit access to SQL data
- Use the DROP verb to delete a table, view, or index
- Define the data elements in a table or view
- Use STRSQL and iSeries Navigator to create SQL objects
- Use INSERT, UPDATE, ALTER, and DELETE statements to maintain a database

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This unit will appeal to programmers and will have limited value for end users. What I suggest you do is have those who are not interested in DB maintenance work on more functions and subqueries. While you are presenting this unit, ask the students to review the Functions and Subqueries units and complete any parts of the labs that have not been finished. You should ask the students to note any questions they have and save them until you have completed this unit.

We will cover how to manage SQL objects and how to create and maintain data in the database.

Then, we discuss how to control access to SQL data objects and how database features, such as triggers, referential integrity, and check constraints, can optionally be used by SQL.

Next, we provide an overview some of the SQL SYS objects that help you to manage your data.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

8.1. SQL data definition and manipulation

Instructor topic introduction

What students will learn — The students learn how to describe and alter tables and their attributes using Data Definition Language as well as manipulate the data using Data Manipulation Language.

Instructor notes — We focus our attention on those SQL Data Definition Language statements that define data to SQL: CREATE, ALTER, and DROP. We will also cover some Data Manipulation Language statements that manipulate data: INSERT, UPDATE, DELETE and DROP. Further explanation of all of the statements can be found in the SQL Reference Manual. I often repeat concepts and information in order to reinforce the students' understanding or to stimulate questions and class discussion. For example, we have already covered the CREATE statement, but it is briefly repeated here.

These topics are being covered in order to prepare the students for the final lab exercise environment.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

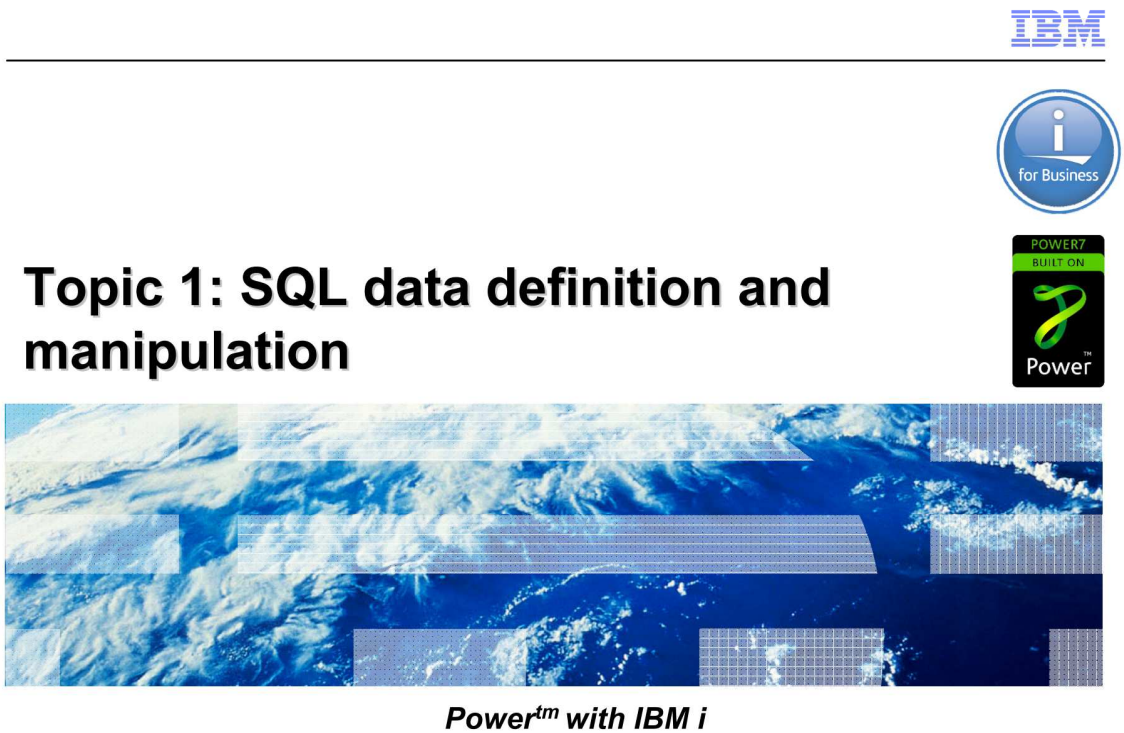


Figure 8-2. Topic 1: SQL data definition and manipulation OD477.0

Notes:

This topic explores the creation, alteration and maintenance of your tables and the data in those tables.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

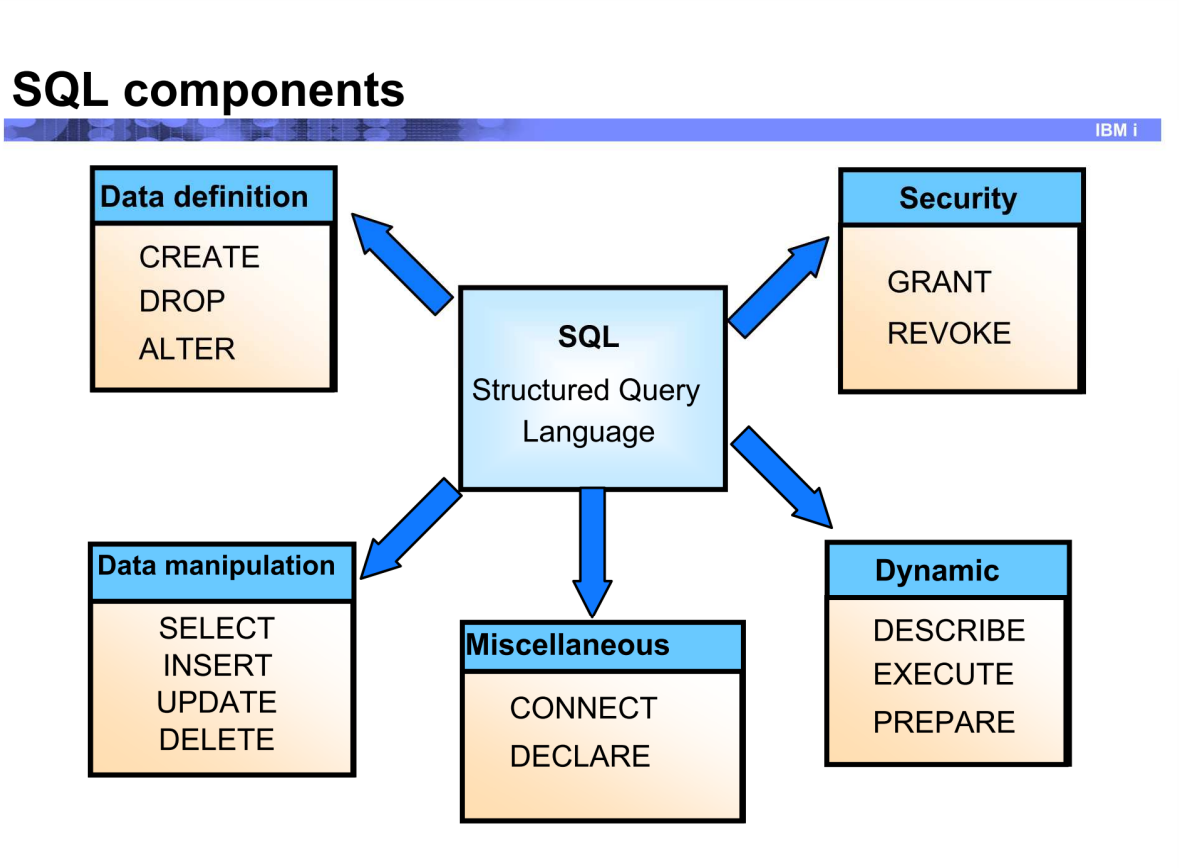
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-3. SQL components OD477.0

Notes:

In this topic, we discuss a few more details of Data Definition Language (DDL) and Data Manipulation Language (DML) statements that you can use to modify SQL objects:

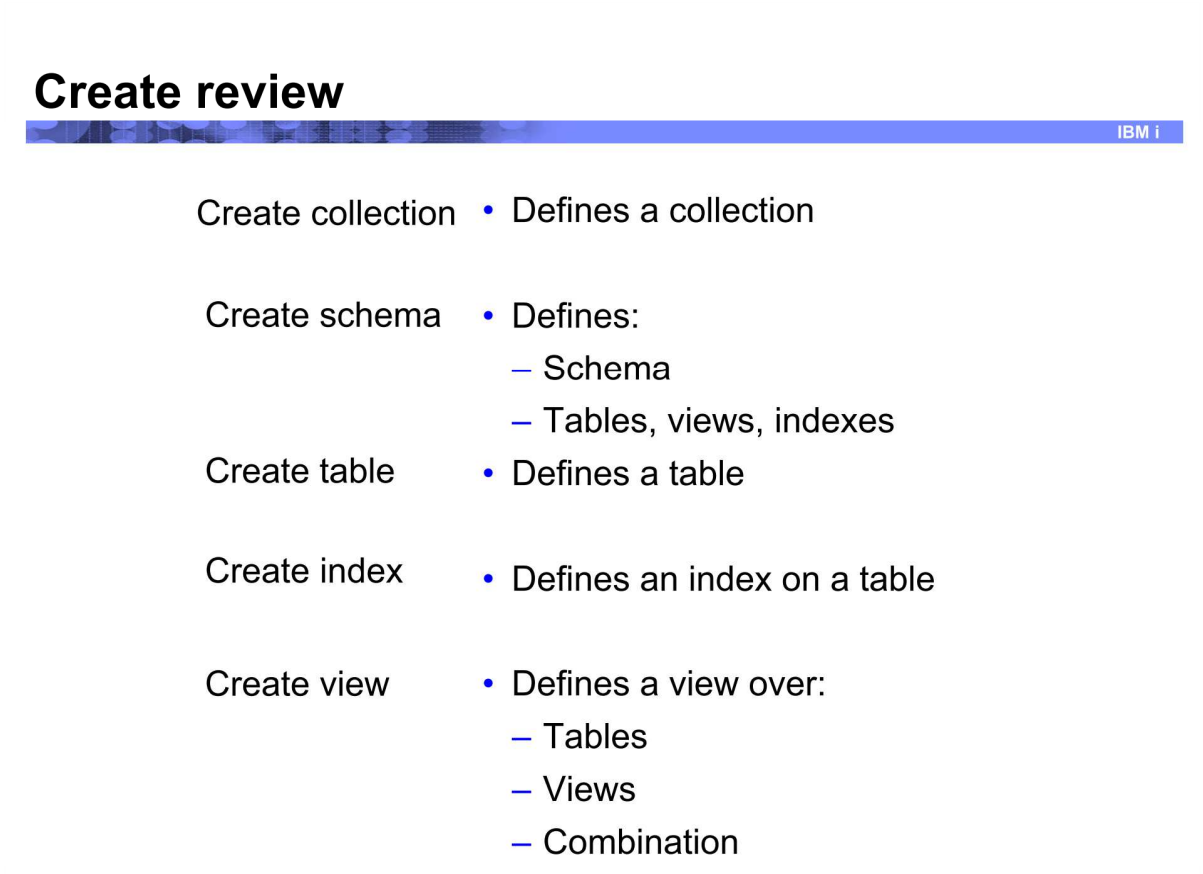
- Data Definition Language and Data Manipulation Language are integral parts of the SQL language.
- The SQL programmer can not only CREATE but also ALTER, or DROP SQL schemas, tables, and views.
- The SQL programmer can UPDATE existing rows, or DELETE existing rows from a table.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Focus on the DDL and DML statements that are highlighted. We discuss these statements in this topic. GRANT and REVOKE follow in the next topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-4. Create review OD477.0

Notes:

We have already seen CREATE used for several purposes. This visual summarizes and reviews them.

The CREATE COLLECTION statement creates a schema in which tables, views, indexes, and packages can be created.

The CREATE SCHEMA statement defines a schema and optionally creates tables, views, aliases, indexes, and distinct types. Comments and labels can be added in the catalog description of tables, views, aliases, indexes, columns, and distinct types. Table, view, and distinct type privileges can be granted to users.

CREATE TABLE is similar to CRTPF.

CREATE INDEX and CREATE VIEW are similar to CRTLF.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create table members

IBM i

```
CREATE TABLE CARCLUB/MEMBERS
         (MEMBERNUM DEC (3, 0) NOT NULL WITH DEFAULT,
          LASTNAME CHAR (25) NOT NULL WITH DEFAULT,
          FIRSTNAME CHAR (15) NOT NULL WITH DEFAULT,
          DUESAMOUNT DEC (5, 2) NOT NULL WITH DEFAULT,
          STREET CHAR (25) NOT NULL WITH DEFAULT,
          CITY CHAR (15) NOT NULL  WITH DEFAULT,
          STATE CHAR (2) NOT NULL WITH DEFAULT,
          ZIP DEC (5,0) NOT NULL WITH DEFAULT)
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-5. Create table members OD477.0

Notes:

Here is an example of the CREATE table statement for the CARS table.

For appropriate syntax, note that:

- Parameters for the statement are enclosed in brackets.
- Parameters are separated by commas.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This visual is the CREATE TABLE statement that we used in an earlier exercise where we discussed how to define various columns in the table. It is repeated here to remind students of the statement's syntax.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Column names

IBM i

- First character A through Z, #, @, $
- Subsequent characters A-Z, #, @, $, -, 0-9
- Maximum 30 characters
- Uppercase
- No reserved words
- No prefix beginning with:
  - SQ
  - RDI
  - DSN

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-6. Column names OD477.0

Notes:

SQL allows names that are fairly long which might cause problems in other interfaces. You can choose to limit the number of characters in your SQL names or use other SQL statements to rename objects.

Associated with each column name is a system name that must be less than or equal to ten characters. There is an optional FOR clause that you can use to assign a system name to the column. If you do *not* assign one, the system generates a name that is derived from your column name. These system column names must be unique in the schema.

Here is a subset of the rules for the system-column-name:

> If the column name does not contain special characters and is longer than ten characters, a ten character system-column-name will be generated as the first five characters of the name plus a five digit unique number. For example, the system-column-name for LONGCOLUMNNAME would be LONGC00001.

In the interest of simplicity, in this class we use SQL names of ten characters or less.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Names beginning with the characters SQ, SQL, sql, RDI, or DSN should not be used because precompilers generate host variables that begin with these characters.

While “SQ” is allowed in COBOL, it *must not* be used in RPG.

While system column names are generally not specified, they must be unique in the schema. It is generally easiest to let the system generate a unique system name for each column that will then be used in system catalogs (we discuss these in a later topic in this unit).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Labels and column headings

IBM i

```
For a Table or View:

  LABEL ON TABLE CARCLUB/MEMBERS is 'Car Club Members Table'

For Columns:

  LABEL ON COLUMN CARCLUB/MEMBERS
    (MEMBERNUM  IS  'Member Number',
     LASTNAME  IS 'Surname',
     FIRSTNAME  IS  'First Name',
     DUESAMOUNT  IS  'Annual Dues Amount',
     STREET  IS  'Street Address',
     CITY  IS  'City',
     STATE  IS  'State',
     ZIP  IS  'Postal Code')
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-7. Labels and column headings OD477.0

Notes:

The LABEL ON statement adds or replaces labels in the catalog description of tables, views, or columns:

- Labels are enclosed in quotes.
- It is not mandatory to label each column.

Note that labels can not be specified with the CREATE TABLE statement. They are specified separately in the LABEL ON.

Labels are also used to produce column headings that are used when displaying or printing query results.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — There is a way to make the labels break into tiers for reports (Query Manager) by starting each tier at 20 column intervals. In Query Manager, you can do this with underscores specifying the tier break.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Alter table

IBM i

```
ALTER TABLE CARCLUB/MEMBERS
            ADD COUNTRY CHAR(15) NOT NULL WITH DEFAULT
            ADD PHONENUMBER DEC(15) NOT NULL WITH DEFAULT
```

Can perform many operations in a single ALTER.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-8. Alter table OD477.0

Notes:

The ALTER TABLE statement adds, drops, or alters an individual column from an existing table. It is also used to add unique, referential, or check constraints and to add or drop a primary, unique, or foreign key. We discuss constraints in the next topic.

There are three clauses in the ALTER statement:

1. **ADD:** Used to add a column and its definition or to add a database constraint to the table.
2. **DROP**: Used to drop a column or a key from the table.
3. **ALTER**: Used to change a column in the table. You can change the data type or the NULL capability of a column, for example.

In this example we are adding two columns, COUNTRY and PHONENUMBER, to the MEMBERS table. We have decided to allow people from other countries to join the club. If you prompt this statement, only tables and physical files in the schema will be displayed.

You can perform multiple adds, drops, or alters in a single ALTER TABLE statement. In the example we are adding two columns to the table.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Any additional columns are added to the table after the existing columns.

Question

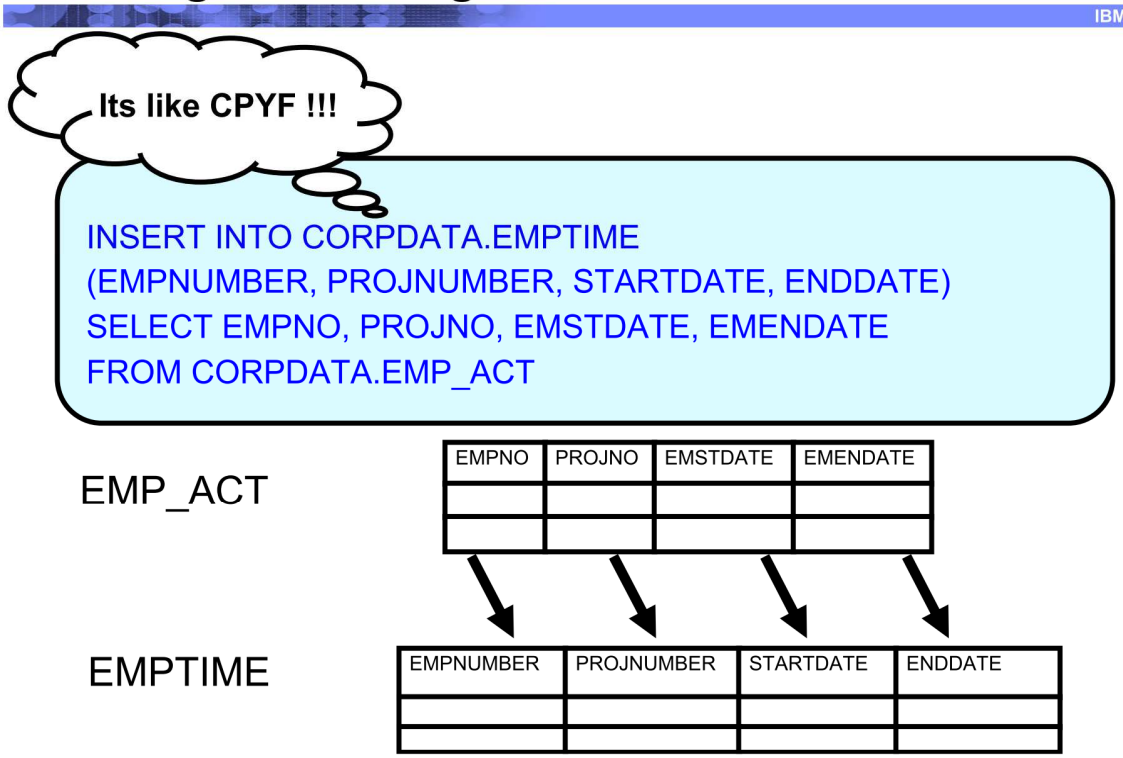
Can you guess what SQL statement would update the values of the new columns in the MEMBERS table?

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Inserting rows using a SELECT statement

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-9. Inserting rows using a SELECT statement OD477.0

Notes:

You can use a select-statement within an INSERT statement to insert zero, one, or more rows selected from the table or view you specify into another table. The table you select the rows from can be the same table you are inserting into. If they are the same table, SQL creates a temporary result table containing the selected rows and then inserts the rows from the temporary table into the target table.

One use for this kind of INSERT statement is to move data into a table you created for summary data. For example, suppose you want a table that shows each employee's time commitments to projects. You could create a table called EMPTIME with the columns EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE, and TTIME, and then use the following INSERT statement to fill the table:

```
INSERT INTO EMPTIME (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
FROM CORPDATA.EMP_ACT
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The select-statement embedded in the INSERT statement is no different from the select-statement you use to retrieve data. With the exception of FOR READ ONLY, FOR UPDATE OF, or the OPTIMIZE clause, you can use all the keywords, column functions, and techniques used to retrieve data. SQL inserts all the rows that meet the search conditions into the table you specify. Inserting rows from one table into another table does not affect any existing rows in either the source table or the target table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

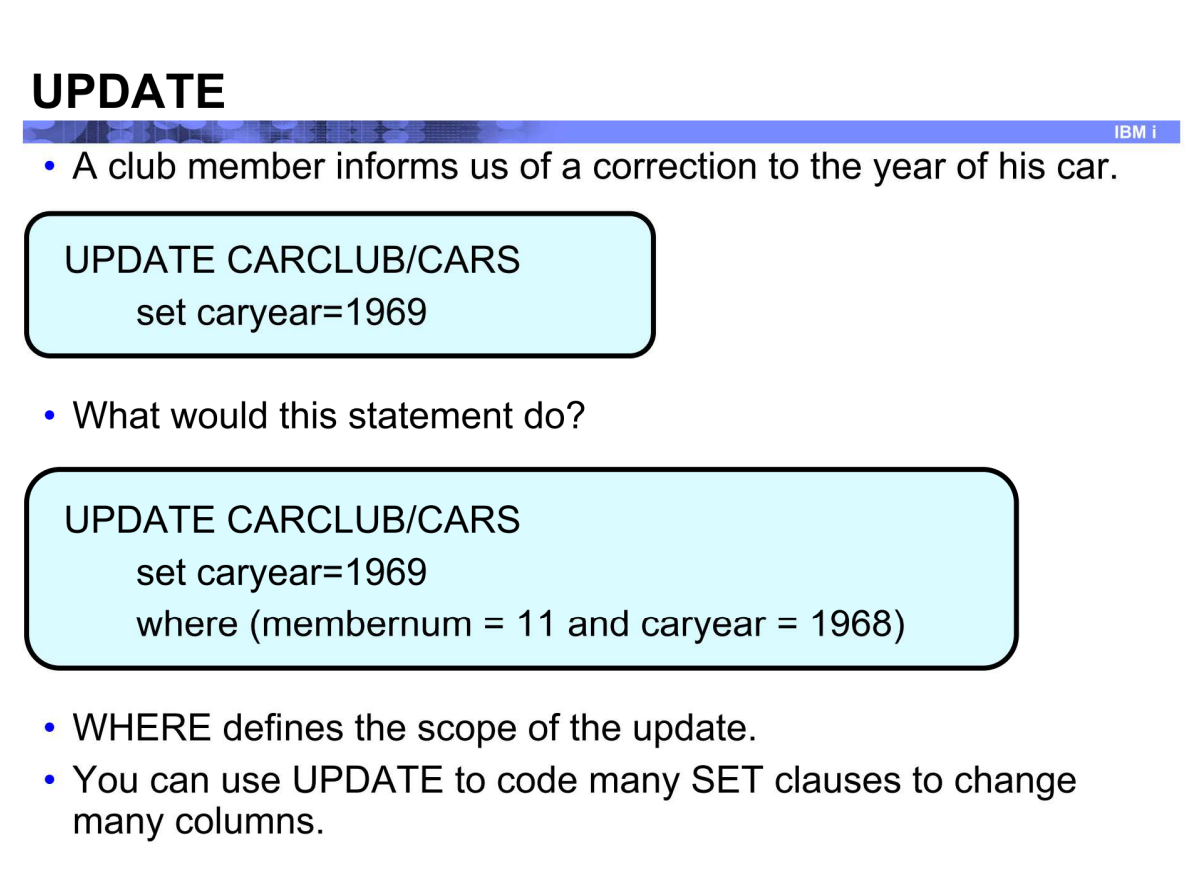
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-10. UPDATE OD477.0

Notes:

The UPDATE statement updates the values of specified columns in rows of a table or view. Updating a row of a view updates a row of its base table.

In this visual, we show several examples of UPDATE. Notice the searched form of UPDATE (WHERE clause), which updates one or more rows determined by a WHERE search condition. This example corrects the CARYEAR by changing it from 1968 to 1969.

Scope of update

Note that all rows that satisfy the search condition in the WHERE clause are updated. This makes UPDATE a very tool. If a group of employees were moved to a new manager in a new department, the change could be performed with a single SQL statement.

Many programmers appreciate the productivity they gain by using this capability. However, many will tell stories of the time they accidentally updated the complete table because they forgot to limit the scope of the UPDATE by coding a WHERE!

 © Copyright IBM Corp. 2010, 2013
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Actually, the second UPDATE could apply to more than one row if this particular member owns more than one 1968 model year vehicle. It would be better to be specific and code the WHERE like this:

```
UPDATE CARCLUB/CARS
       SET CARYEAR=1969
       WHERE CARNUM=15
```

Another feature of the UPDATE statement is that you can code many SETs in a single statement. With a single UPDATE statement, multiple changes can be made to the table. This feature alone makes embedded SQL (in application programs) a very attractive alternative to coding database I/O using native support.

Another form of UPDATE is known as the *positioned UPDATE*. This form is based on a cursor and can be only embedded in a program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — We use the UPDATE statement to assign values. Students might guess that ALTER would do this, but ALTER cannot be used to assign values. Let the students think for a moment.

What RPG opcode would change a record? UPDATE is the statement used in SQL. We spend more time on it later, but here is an example:

```
update carclub/members set country='United States'
       where membernum=00010
```

Emphasize the power of SQL statements such as UPDATE. Because many rows can be updated with a single and easily understood SQL statement, embedded SQL is an attractive alternative to writing native RPG I/O operations. This power requires care when you use it. Without a WHERE clause to limit the scope of an UPDATE, or an incorrectly coded WHERE, errors can be propagated throughout your table. So, testing is even more important because of the scope of SQL updates.

Transition — We discuss constraints in the next topic.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Advanced UPDATE techniques

IBM i

1)
```
UPDATE CARCLUB/CARS
    SET caryear=1969, carmake = 'Ford'
    WHERE  (membernum = 11 and caryear = 1968)
```

2)
```
UPDATE CARCLUB/CARS
    SET (caryear, carmake) = (1969, 'Ford')
    WHERE  (membernum = 11 and caryear = 1968)
```

3)
```
UPDATE CARCLUB/CARS
  SET caryear = (SELECT caryear FROM
          CARCLUB/CARS WHERE membernum = 10001)
                WHERE membernum = 10002
```

4)
```
UPDATE CARCLUB/CARS SET ROW =
(SELECT * FROM MYCOPY
WHERE CARS.CARNUM = MYCOPY.CARNUM)
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-11. Advanced UPDATE techniques OD477.0

Notes:

The SET clause of an UPDATE statement can be used in many ways to determine the actual values to be set in each row being updated. The following example lists each column with its corresponding value:

```
UPDATE CARCLUB/CARS
SET caryear=1969, carmake = 'Ford'
WHERE (membernum = 11 and caryear = 1968)
```

The previous update can also be written by specifying all of the columns and then all of the values:

```
UPDATE CARCLUB/CARS
SET (caryear, carmake) = (1969, 'Ford')
WHERE (membernum = 11 and caryear = 1968)
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Another way to select a value (or multiple values) for an update is to use a scalar-subselect. The scalar-subselect allows you to update one or more columns by setting them to one or more values selected from another table.

```
UPDATE CARCLUB/CARS
SET caryear = (SELECT caryear FROM CARCLUB/CARS
WHERE membernum = 10001) WHERE membernum = 10002
```

You can also update an entire row in one table with values from a row in another table.

```
UPDATE CARCLUB/CARS SET ROW =
(SELECT * FROM MYCOPY
WHERE CARS.CARNUM = MYCOPY.CARNUM)
```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

DELETE

IBM i

```
DELETE from CARCLUB/CARS

DELETE from CARCLUB/CARS where membernum=11 and
carmodel='Superbee'
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-12. DELETE OD477.0

Notes:

The DELETE statement deletes rows from a table or view. Deleting a row from a view deletes the row from the table on which the view is based. This form of the DELETE statement is used to delete one or more rows (optionally determined by a search condition).

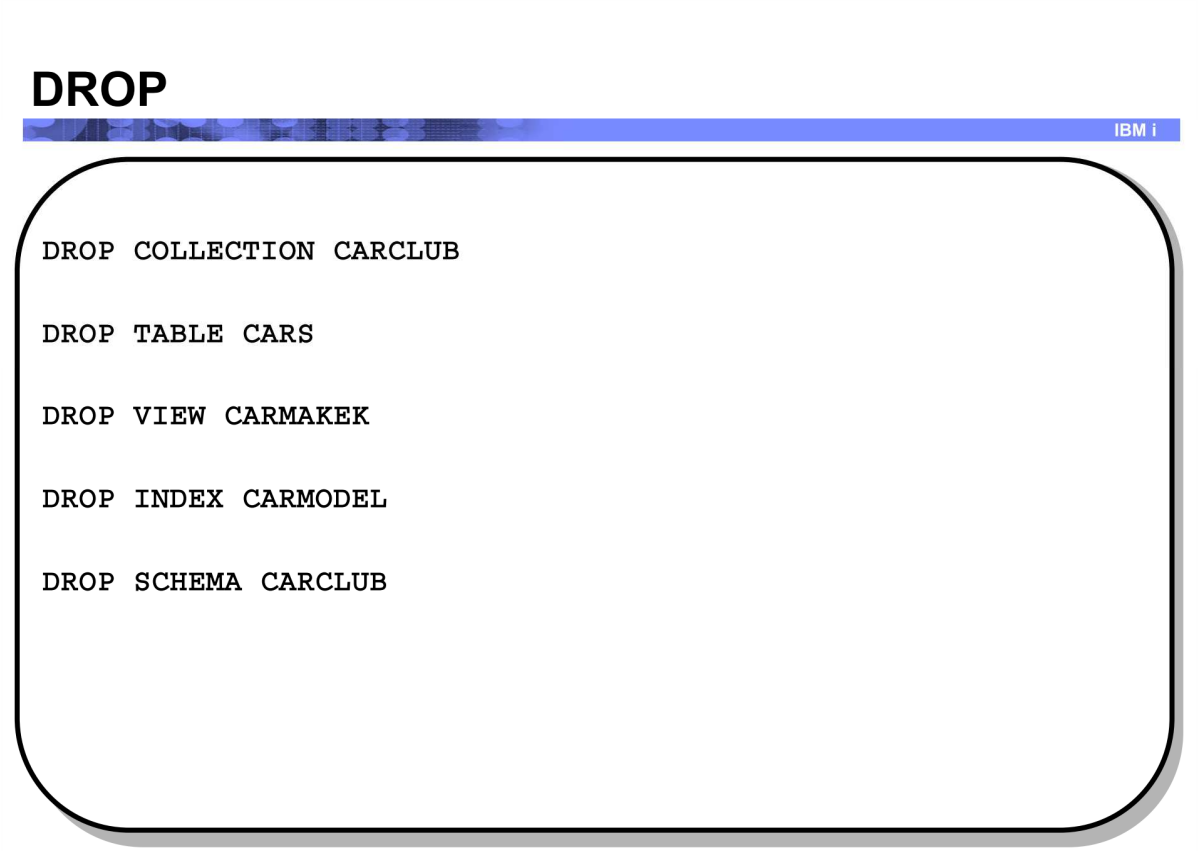
In the first statement, we clear the CARS table of data. In the second statement, we delete a row since this member has sold this car and no longer owns it.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Discuss a possible problem if this member of the car club owns more than one “Superbee.” In this situation all rows that satisfied the WHERE would be deleted.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-13. DROP OD477.0

Notes:

The DROP statement is the opposite of CREATE. DROP is used to delete an index, a view, a table, or an entire schema from the system.

The DROP statement deletes an object.

Note: Also, any objects that are directly or indirectly dependent on that object are also deleted.

This is different than DLTF. In the case of DLTF, all dependent objects must be deleted first. That is, logical files based on a physical file must be deleted before the physical file. With DROP, this is handled automatically.

Whenever an object is deleted, its description is deleted from the catalog.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Enhancing an SQL table*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-14. Exercise OD477.0

Notes:

Perform the brief exercise *Enhancing an SQL table*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

8.2. Data integrity

Instructor topic introduction

What students will learn — The students will learn that data integrity and business rules can be enforced through the use of check constraints, referential integrity, and triggers.

Note: We do not intend to teach RI, triggers, and check constraints to the students. We want them to be aware that they are supported within SQL and the database only.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Figure 8-15. Topic 2: Data integrity OD477.0

Notes:

SQL allows you to enforce data integrity and business rules through check constraints, referential integrity and triggers.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Data integrity

IBM i

- Access rights
- Table constraints
  - Unique
  - Check
  - Referential
- Triggers
  - Written in SQL
  - Written in RPG IV, COBOL, and so forth

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-16. Data integrity OD477.0

Notes:

1. **Access rights:** Secure access to data through the GRANT and REVOKE SQL statements, (similar to IBM i object and data rights).

   GRANT and REVOKE manage access to tables and views.

2. **Table constraints:** Validate the data entered or updated in a table through *rules* that are enforced at the table level. These rules are defined on the CREATE and ALTER SQL statements.

   A *unique constraint* defines a unique primary key composed of one or more columns of a table.

   A *check constraint* is a rule that specifies the values allowed in one or more columns of every row of a table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Referential integrity is that state of a database in which all values of all foreign keys are valid. A *foreign key* is a key that is part of the definition of a referential constraint. A *referential constraint* is the rule that the values of the foreign key are valid only if:

- They appear as values of a parent key
  or
- Some component of the foreign key is null

3. **Triggers:** Provide automated enforcement of application rules and functions by implicit program execution whenever rows are added, changed, or deleted. Triggers are implemented using SQL or a high level language, such as RPG IV.

References:

DB2 for i SQL Reference, Version 5, Chapter 5, Statements (Grant Table Privileges)

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce the facilities that SQL and DB2 for IBM i offer for data integrity. We do not get into a lot of detail on referential integrity and triggers. Because programmers can extend more control in the area of triggers, they may be interested in discussion of triggers outside of class time. Check constraints were new for V4R2 and this will be the first time that many i programmers have heard of them.

Additional information — The RENAME statement renames a table, view, or index. The name and the system object name of the table, view, or index can be changed. This statement can only be executed by a user with the appropriate level of object authority.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

GRANT and REVOKE

IBM i

```
GRANT SELECT, INSERT, UPDATE, DELETE ON CARCLUB TO OL37000

GRANT SELECT ON CARCLUB TO PUBLIC

GRANT ALL PRIVILEGES ON CARCLUB TO RJSLANEY


REVOKE SELECT ON CARCLUB FROM PUBLIC

REVOKE INSERT, UPDATE, DELETE on CARCLUB FROM OL37999



DROP CARCLUB
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-17. GRANT and REVOKE OD477.0

Notes:

GRANT and REVOKE are used to control access to tables and views. There are many different parameters that are associated with GRANT. A few are shown. REVOKE is used to remove privileges from a user profile.

In the visual, *PUBLIC is granted viewing privileges only to the CARS table. More privileges are granted to user OL37000. If user OL37999 leaves the company or moves to a different department his privileges would be revoked.

Note that dropping a table or view automatically revokes privileges to users.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Note that *PUBLIC authority is set by default on the CREATE statement:

- *EXCLUDE if you are using SQL naming
- *LIBCRTAUT if you are using SYS naming

when the table is created.

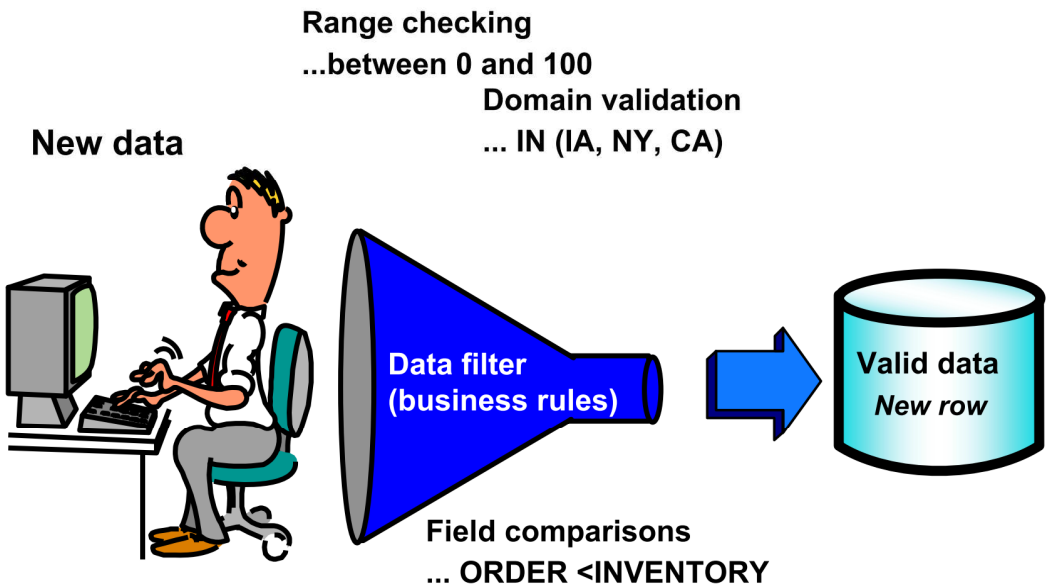
Additional information — Also note that GRANT allows you to restrict UPDATE to specific columns of a table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Check constraints

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-18. Check constraints OD477.0

Notes:

A *check constraint* is a rule that specifies the values allowed in one or more columns of every row of a table. Check constraints are optional and can be defined using the SQL statements CREATE TABLE and ALTER TABLE. The definition of a check constraint is a restricted form of a search condition. One of the restrictions is that a column name in a check constraint on a table must identify a column of that table.

A check constraint *assures the validity of data during inserts and updates* by checking the data against a search expression that you define.

For example, you can add a check constraint on a field such that values inserted into that field must be between 1 and 100. If a value does not fall within that range, the insert or update operation against your database is not processed.

A table can have an arbitrary number of check constraints. They are enforced by the database manager when:

- A row is inserted into the table.
- A row of the table is updated.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Each constraint for a table is given a unique name.

A check constraint is enforced by applying its search condition to each row that is inserted or updated. An error occurs if the result of the search condition is FALSE for any row.

We can implement the same constraint using the CL `ADDPFCST` command also.

Note an advantage of using CREATE TABLE over CRTPF with constraints. CREATE TABLE provides support for defining constraints at the same time that you create the table object. CRTPF offers no DDS support, so it requires an additional step when you need to add constraints to the table.

Check restraints compliment referential integrity. RI ensures that every foreign key value has a matching parent key value, but check constraints perform data validation in the values stored in columns.

For those with experience using DDS on the i, the RANGE, COMP, and VALUES keywords are also available. But, they only apply to display files. Data could be added or changed through various other routes circumventing these display file keywords. However, check constraints are watertight because they are implemented at table level, and offer more function.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — Use DSPFD to display the description of the CARS table with the check constraint implemented.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Defining check constraints

IBM i

```
ALTER TABLE CARCLUB/CARS
ADD CONSTRAINT CAR_IS_OLD
CHECK (CARYEAR < 1990 )
```

If INSERT with CARYEAR >= 1990:

Message: `INSERT or UPDATE not allowed by CHECK constraint.`

Cause: The value being inserted or updated does not meet the criteria of the CHECK constraint CAR_IS_OLD. The operation is not allowed.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-19. Defining check constraints OD477.0

Notes:

In this example, we want to enforce a club rule that only cars of model year 1989 or earlier are added to the table. We are applying the constraint to the column CARYEAR of the table, CARS.

Each constraint for a table is given a unique name. In the example, the constraint is named *CAR_IS_OLD*.

The *CAR_IS_OLD* check constraint is enforced by applying its search condition to each row that is inserted or updated. An error occurs if the result of the search condition is FALSE for any row. If the CARYEAR is greater than or equal to 1990, the constraint returns false, a message is issued, and it is not inserted or updated.

You cannot apply a new constraint to a table if there are currently values in that table that violate the constraint.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

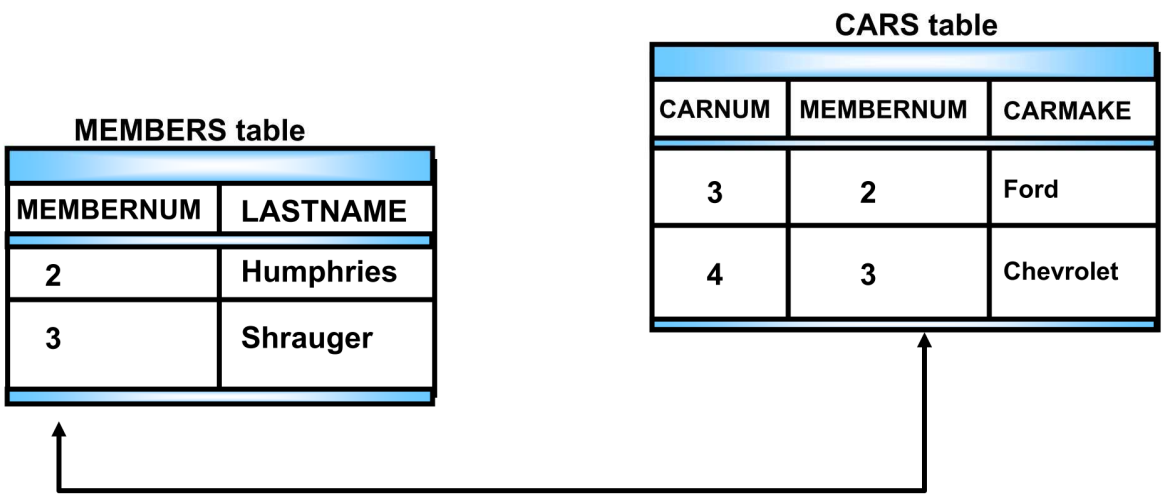
Instructor notes:

Additional information — Check constraints allow more sophisticated validation than has been previously available in DDS. More important, check constraints are enforced on all interfaces (RPG, DFU, Java, SQL, ODBC, Domino, and so forth).

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Referential integrity

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-20. Referential integrity OD477.0

Notes:

The table containing the parent key is called the parent table of the referential constraint, and the table containing the foreign key is a dependent of that table.

An example of such a dependent relationship is shown using our MEMBERS and CARS tables. While we can have a member with no cars in the club, the reverse is not true. So, we enforce a constraint that each row of the CARS table must have a valid value in the MEMBERNUM column. The column in the MEMBERNUM table is called a *foreign key* while the value of MEMBERNUM in the MEMBERS table is called the *primary key*. No row of the MEMBERS table can be deleted if it has a dependent row in the CARS table.

Referential constraints are optional and can be defined in CREATE TABLE statements and ALTER TABLE statements. Referential constraints are enforced by the database manager during the execution of INSERT, UPDATE, and DELETE statements. The enforcement is effectively performed at the completion of the statement except for delete and update rules of RESTRICT, which are enforced as rows are processed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The notes below will help you understand some of the background of RI if you are unfamiliar with the concepts. We do not intend to cover the background of RI. We want to relate it to you so that if you implement it or have implemented it, you can see how to use it in an SQL context.

Referential constraints with a delete or update rule of RESTRICT are always enforced before any other referential constraints. Other referential constraints are enforced in an order independent manner. That is, the order does not affect the result of the operation.

Within an SQL statement:

- A row can be marked for deletion by any number of referential constraints with a delete rule of CASCADE. CASCADE means that any table, view, index, file, or referential constraint that is dependent on the object being dropped will also be dropped.
- A row can only be updated by one referential constraint with a delete rule of SET NULL or SET DEFAULT.
- A row that was updated by a referential constraint cannot also be marked for deletion by another referential constraint with a delete rule of CASCADE.

The rules of referential integrity involve the following concepts and terminology:

- **Parent key:** A primary key or unique key of a referential constraint.
- **Parent row:** A row that has at least one dependent row.
- **Parent table:** A table that is a parent in at least one referential constraint. A table can be defined as a parent in an arbitrary number of referential constraints.
- **Dependent table:** A table that is a dependent in at least one referential constraint. A table can be defined as a dependent in an arbitrary number of referential constraints. A dependent table can also be a parent table.
- **Descendent table:** A table is a descendent of table T if it is a dependent of T or a descendent of a dependent of T.
- **Dependent row:** A row that has at least one parent row.
- **Descendent row:** A row is a descendent of row p if it is a dependent of p or a descendent of a dependent of p.
- **Referential cycle:** A set of referential constraints such that each table in the set is a descendent of itself.
- **Self-referencing row:** A row that is a parent of itself.
- **Self-referencing table:** A table that is a parent and a dependent in the same referential constraint. The constraint is called a *self-referencing constraint*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — While you should not cover the details of RI, it might be fun to work on the board with the students to write the SQL statement to add a referential constraint for the situation described in the visual.

```
ALTER TABLE CARS
      ADD CONSTRAINT MEMBER
      FOREIGN KEY (MEMBERNUM)
      REFERENCES MEMBERS
```

Before running this statement, you would have to alter the MEMBERS table by specifying MEMBERNUM as a PRIMARY or UNIQUE key.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

8.3. SQL SYS objects

Instructor topic introduction

What students will learn — The contents of SQL SYSxxx objects and how to view and use them.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

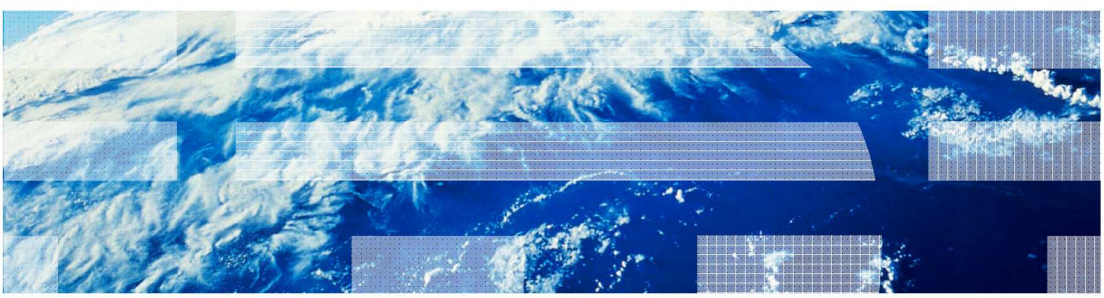


Figure 8-21. Topic 3: SQL SYS objects OD477.0

Notes:

A catalog is automatically created when you create a schema. This topic describes some of the tables and views and some uses for them.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL catalog

IBM i

- System maintained database of SQL objects and facilities used
- Information about tables, parameters, procedures, packages, views, indexes, and constraints
- Some are tables in QSYS: QADBxxxx
- Some are views in QSYS2: SYSLANGS, SYSPARMS and SYSPROCS
- SQL collections have their own set of catalog views.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-22. SQL catalog OD477.0

Notes:

The database manager maintains a set of tables containing information about the data in the database. These tables are collectively known as the *catalog*. The catalog tables contain information about tables, parameters, procedures, packages, views, indexes, and constraints on the i system.

This information is contained in a set of cross-reference tables in libraries QSYS and QSYS2. Library QSYS2 also contains a set of catalog views built over the QSYS catalog tables, which describe information about all the tables, views, indexes, packages, procedures, files, and constraints on the system. In each SQL schema there is a set of views built over the catalog tables, which contains information about the tables, views, indexes, packages, files, and constraints in the schema.

A catalog is automatically created when you create a schema. You cannot drop or explicitly change the catalog.

- *DB2 for i SQL Reference*, Appendix G. DB2 for i Catalog Views

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

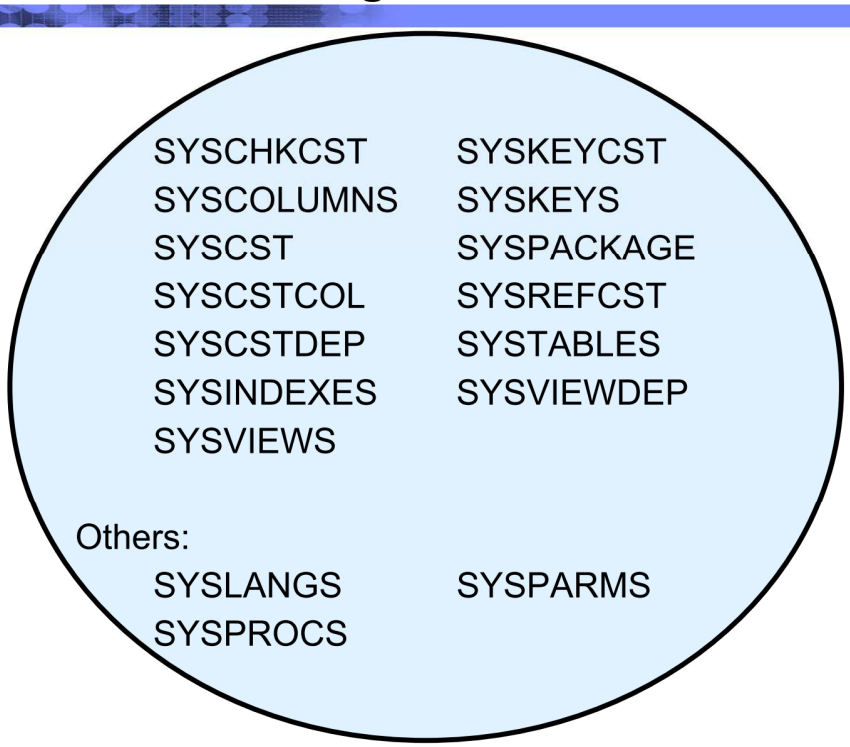
Details — There are sets of these views that are specific to a schema (the course schema, for example) and also system-wide (in QSYS or QSYS2).

You might want to demonstrate SYSCOLUMNS for the MEMBERS table. You can use RUNQRY or SELECT to show the contents of SYSCOLUMNS.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL collection catalog views

IBM i



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-23. SQL collection catalog views OD477.0

Notes:

SYSCHKCST	One row for each check constraint.
SYSCOLUMNS	One row for each column of each table and view.
SYSCST	One row for each constraint.
SYSCSTCOL	One row for every column in a unique or primary key constraint and the referencing columns of a referential constraint.
SYSCSTDEP	Records tables on which constraints are defined.
SYSKEYCST	One or more rows for each unique key, primary key, or foreign key.
SYSKEYS	One row for every column of an index.
SYSPACKAGE	One row for each SQL package.
SYSREFCST	One row for each foreign key.
SYSTABLES	One row every table, view, or alias.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SYSVIEWDEP	Dependencies of views on tables.
SYSVIEWS	One row for each view.
SYSINDEXES	One row for every index created by CREATE INDEX.

You can see a copy of the previous views in one of our class schemas.

There are several other views maintained by SQL. SYSLANGS, SYSPARMS, and SYSPROCS are stored only in QSYS2. They are monitored on a system-wide basis only.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using SQL collection catalog views

IBM i

- Can be used like any view
- Can produce database reports
- Alternative to DSPFFD, DSPFD, and DSPDBR
- System-wide catalog in QSYS and QSYS2

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-24. Using SQL collection catalog views OD477.0

Notes:

SQL schema catalog views can be used like any view. You can easily produce customized reports about the database, such as analyzing the impact of a column change.

You can use these views as an alternative to IBM i commands such as **DSPFFD**, **DSPFD**, DSPDBR, and so on. You can also interrogate system-wide catalogs in QSYS and QSYS2.

However, the DSPxxx commands are still useful because they provide information about all file types, such as **DSPF** and **PRTF** files (created using DDS).

There are also some APIs available that can provide similar information.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out that the SYSxxx views are over QADBxx tables in QSYS. The real data is in the QSYS. SYSxx catalog is cross-platform (standard SQL).

Also, note that these views might change across IBM i releases and refer to Appendix G in the SQL Reference.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Impact of changing a column

IBM i

```
SELECT COLUMN_NAME, TABLE_NAME, COLUMN_TEXT
       FROM CARCLUB/SYSCOLUMNS
       WHERE COLUMN_NAME='ZIP'
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-25. Impact of changing a column OD477.0

Notes:

This query tells you wherever the column name zip is used in the schema CARCLUB.

If you specified the SYSCOLUMNS view in QSYS2, you would discover where zip is used system-wide.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

8.4. SQL objects exercise environment

Instructor topic introduction

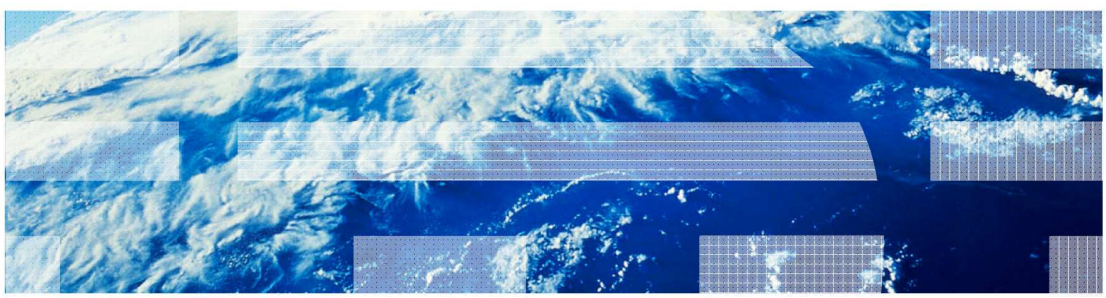
In this topic, we build the environment for the exercise where the students maintain a database that is a subset of what we have been using in other exercises in the class.

The visuals are ordered according to the tasks as they are performed in practice when establishing a test environment.

1. Show the structure and the contents of the TESTEMP table, which must be used for the workstation labs.
2. Show the CREATE TABLE statement for this table.
3. Create a view to provide a subset of this table.
4. Insert new rows into the table.
5. Insert other test data from the EMPLOYEE table (mass insert).
6. Update table TESTEMP (single row).
7. Update table TESTEMP (multiple rows).
8. Delete rows from table TESTEMP.
9. Drop table TESTEMP.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Topic 4: SQL objects exercise environment

Power^{tm} with IBM i

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

8.1

Figure 8-26. Topic 4: SQL objects exercise environment OD477.0

Notes:

This topic shows you how databases are setup and will set the environment for the exercise at the end of this unit.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

TESTEMP table

IBM i

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-04	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00
000111	SMITH	C01	1998-06-25	25000.00	- - - - -

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-27. TESTEMP table OD477.0

Notes:

Often, you will need to create test data or a test environment. This topic steps you through a methodology based on using SQL.

In our case, we show you how we created a sample table. The visual shows the contents of the TESTEMP table. This table is the basis for the exercise at the end of this unit.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Show the contents and layout of the TESTEMP table.

Details — The TESTEMP table will be created and used in the exercises.

Additional information — The next visual shows the CREATE TABLE statement for this table. Data types and the syntax of the CREATE TABLE statement have been discussed.

Transition statement — Now let's look how this table is created.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create table TESTEMP

IBM i

```
CREATE TABLE OL37nnCOL/TESTEMP
  (EMPNO        CHAR(6)        NOT NULL,
  LASTNAME      VARCHAR(15)    NOT NULL,
  WORKDEPT      CHAR(3)        ,
  HIREDATE      DATE           ,
  SALARY        DECIMAL(9,2)   ,
  BONUS         DECIMAL(9,2)    )
```

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-28. Create table TESTEMP OD477.0

Notes:

This CREATE TABLE statement creates the TESTEMP table.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

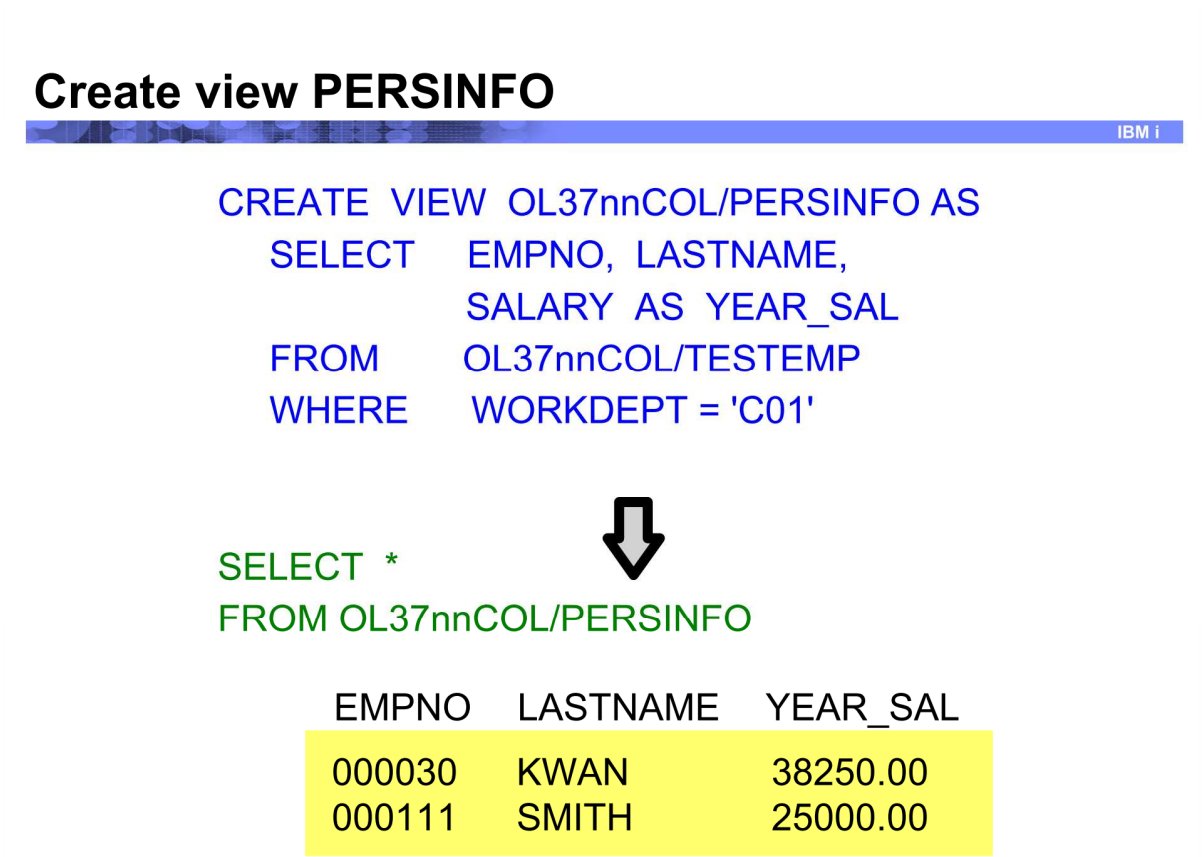


Figure 8-29. Create view PERSINFO OD477.0

Notes:

This CREATE VIEW statement describes a part of the base table, consisting of employee number, last name, and yearly salary of all employees in department C01.

The AS clause in the subselect enables you to specify column names in the view different from the column names in the referenced table. In the above example, SALARY AS YEAR_SAL is used to rename the SALARY column in the base table TESTEMP to YEAR_SAL in the view PERSINFO. Users referencing the view PERSINFO must use the new column name.

Another way to rename columns in a view is:

```
CREATE VIEW OL37nnCOL/PERSINFO (EMPNO, LASTNAME, YEAR_SAL)
       AS SELECT EMPNO, LASTNAME, SALARY
       FROM OL37nnCOL/EMPLOYEE WHERE WORKDEPT = 'C01'
```

When using this syntax, the number of column names listed in the parentheses must match the number of the selected columns.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Users who are authorized to use the view PERSINFO will get the listed columns for employees whose department number is C01. Other data is not visible.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To show a simple example of a CREATE VIEW statement.

Details — This example shows the creation of a simple view. Details are discussed in the advanced course.

Additional information — No special privileges are needed to create a view. You can create a view on any table for which you have SELECT authority.

Transition statement — After creating a table, the table is empty. There are many ways to enter data. For mass data insertion, we can use utilities, application programs, or other tools.

We use SQL to add data to our test table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Inserting rows into the table

IBM i

```
INSERT INTO ol37nncol/TESTEMP
    VALUES ('000111', 'SMITH', 'C01', '1998-06-25', 25000, NULL)

INSERT INTO ol37nncol/TESTEMP
    (EMPNO, LASTNAME, WORKDEPT, HIREDATE, SALARY, BONUS)
     VALUES ('000113', 'JONES', 'C01', '2001-06-25', 25000, NULL)

INSERT INTO ol37nncol/TESTEMP
    VALUES ('000114', 'THOMPSON', 'C01', '2001-06-25', 25000, NULL)
```



EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	25000.00	-
000113	JONES	C01	2001-06-25	25000.00	-
000114	THOMPSON	C01	2001-06-25	25000.00	-

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-30. Inserting rows into the table OD477.0

Notes:

Use an INSERT statement to insert rows into a table. Inserting a row using a view inserts the row into the table on which the view is based. Therefore, inserts using a view are only possible if the view contains all columns that are defined with NOT NULL in the base table.

In the first example, the column list is omitted. In this case, a value for each column in the table must be provided between each set of parentheses in the VALUES clause, in the order they are defined for the base table.

In the second example, a column list is specified. The number of values provided for a row using the VALUES clause must be equal to the number of column names in the column list. The first value is inserted in the first column of the list, the second value in the second column, and so on.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To illustrate the INSERT statement syntax for inserting one or more rows per INSERT statement.

Details — The students add rows to their test table during the lab.

Transition statement — The next visual presents another method for inserting rows into a table, namely, the copying of rows from another table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Insert multiple rows

IBM i

```
INSERT INTO OL37nnCOL/TESTEMP
SELECT EMPNO,LASTNAME,WORKDEPT,HIREDATE,SALARY,BONUS
FROM   OL37nnCOL/EMPLOYEE
WHERE  EMPNO <= '000050'
```

SELECT * FROM OL37nnCOL/TESTEMP



EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	25000.00	-
000113	JONES	C01	2001-06-25	25000.00	-
000114	THOMPSON	C01	2001-06-25	25000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-31. Insert multiple rows OD477.0

Notes:

Instead of the VALUES clause, include a SELECT statement in the INSERT statement to tell DB2 for IBM i that another table or view contains the data for the new rows.

In the example, the number and sequence of the selected columns of the EMPLOYEE table must match the number and sequence of the columns of the TESTEMP table.

If a columns list for the target table is used, all its NOT NULL columns must be included.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To illustrate how to insert multiple rows into the table TESTEMP.

Details — During the lab, these rows will be inserted into the table TESTEMP.

Additional information —

Transition statement — After the tables have been loaded, the data can be updated.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Update columns (1 of 2)

IBM i

Before:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	25000.00	-
000113	JONES	C01	2001-06-25	25000.00	-
000114	THOMPSON	C01	2001-06-25	25000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

```
UPDATE OL37nnCOL/TESTEMP
SET BONUS = 500,
      SALARY = 26000
WHERE EMPNO = '000111'
```



After:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	26000.00	500.00
000113	JONES	C01	2001-06-25	25000.00	-
000114	THOMPSON	C01	2001-06-25	25000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-32. Update columns (1 of 2) OD477.0

Notes:

The UPDATE statement updates the values of the specified columns for one or more rows of a table or view.

Our new colleague, Mr. Smith (employee number 000111) receives a first bonus and a salary raise. Therefore, we must set BONUS to 500 and SALARY to 26000.

If more than one column is changed, the columns must be separated by commas. The SET keyword must precede the first column and must not be repeated.

To set BONUS to NULL, the UPDATE statement must be coded as follows:

```
UPDATE TESTEMP
       SET BONUS = NULL
       WHERE EMPNO = '000111'
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To show how to change data in a table.

Details — If more than one column is changed, the columns must be separated by commas. The SET keyword must precede the first column and must not be repeated.

```
SET COL1 = VALUE ,
COL2 = VALUE ,
COL3 = VALUE
WHERE ...
```

Additional information —

Transition statement — The column values are changed for all rows for which the predicate in the WHERE clause is true. An example is shown on the next visual.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Update columns (2 of 2)

IBM i

Before:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	26000.00	500.00
000113	JONES	C01	2001-06-25	25000.00	-
000114	THOMPSON	C01	2001-06-25	25000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

```
UPDATE OL37nnCOL/TESTEMP
SET SALARY = SALARY + 1000
WHERE WORKDEPT = 'C01'
```



After:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	27000.00	500.00
000113	JONES	C01	2001-06-25	26000.00	-
000114	THOMPSON	C01	2001-06-25	26000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	39250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-33. Update columns (2 of 2) OD477.0

Notes:

Department C01 has finished a project early. The salary of all employees in the department should be increased by $1,000.

To update selected rows, use the WHERE clause to specify the rows to update.

If you omit the WHERE clause, all rows of the table or view are updated.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To demonstrate the importance of the WHERE clause in an UPDATE statement.

Details — Emphasize the importance of the WHERE clause.

Additional information —

Transition statement — Mr. Smith wanted a salary increase of $5,000. Because the company did not grant him the raise, he left the company. Let's see how his data is deleted from the TESTEMP table.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Delete rows

IBM i

Before:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000111	SMITH	C01	1998-06-25	27000.00	500.00
000113	JONES	C01	2001-06-25	26000.00	-
000114	THOMPSON	C01	2001-06-25	26000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	39250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

```
DELETE FROM OL37nnCOL/TESTEMP
WHERE EMPNO = '000111'
```



After:

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
000113	JONES	C01	2001-06-25	26000.00	-
000114	THOMPSON	C01	2001-06-25	26000.00	-
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	39250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-34. Delete rows OD477.0

Notes:

Mr. Smith left the company. Therefore, we delete the row with employee number 000111.

The DELETE statement removes entire rows from a table. Depending on the search condition of the WHERE clause, the DELETE statement removes zero, one, or multiple rows of a table.

To delete specific rows, use the WHERE clause to specify the rows to be deleted.

If you omit the WHERE clause, all rows of the table or view are deleted.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Explain the syntax of the DELETE statement and emphasize the importance of the WHERE clause when data are maintained.

Details — Highlight the remarks concerning the WHERE clause.

Additional information —

Transition statement — When we no longer need our test table TESTEMP, we can drop it. Let's look how to remove the table from the database.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Drop table TESTEMP

IBM i



DROP TABLE OL37nnCOL/TESTEMP

View PERSINFO is
automatically dropped

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-35. Drop table TESTEMP OD477.0

Notes:

When executing the statement DROP TESTEMP, DB2 for IBM i deletes the object, that is, the table. As a consequence, all rows and the table definition are deleted.

Any objects (for example, views, or indexes) that are dependent on the table are deleted as well.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Reminders

IBM i

- You must be authorized to execute CREATE and DROP table statements.
- You must be authorized to execute statements that change table content.
- Dropping a table drops dependent objects.
- The search condition in the WHERE clause qualifies rows to be updated or deleted.
- Inserting a row into a view actually inserts the row into the table on which the view is based.
- Updating a row of a view updates the row of the table on which the view is based.
- Omitting the WHERE clause of an UPDATE or DELETE statement causes all rows of the table or view to be updated or deleted.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-36. Reminders OD477.0

Notes:

Remember that omitting the WHERE clause of an UPDATE or DELETE statement causes all rows of the table or view to be updated or deleted.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL377U07** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL377U07** and click play.

c. Ask the students to indicate using a green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL377U07** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: Data Definition Language includes the CREATE, ALTER, and DROP statements of SQL.

2. A schema:
   a. Is an IBM i library
   b. Includes an SQL catalog and views of that catalog
   c. Includes journals and receivers for tables
   d. All of the above

3. True or False: Column names can be no more than 10 characters in length.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-37. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: Data Definition Language includes the CREATE, ALTER, and DROP statements of SQL.

   The answer is True. DDL includes CREATE, ALTER, and DROP.

2. A schema:
   a. Is an IBM i library
   b. Includes an SQL catalog and views of that catalog
   c. Includes journals and receivers for tables
   d. All of the above

   The answer is All of the above.

3. True or False: Column names can be no more than 10 characters in length.

   The answer is False. Column names can be up to 30 characters in length.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Maintaining database objects*



© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-38. Exercise OD477.0

Notes:

Perform the workstation exercise *Maintaining database objects*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completing this unit, you should be able to:

- Use the CREATE statement to build a table, view, or index
- Describe how referential integrity and check constraints are supported within SQL
- Use ALTER TABLE to add, drop, or alter a column
- Use the GRANT and REVOKE statements to limit access to SQL data
- Use the DROP verb to delete a table, view, or index
- Define the data elements in a table or view
- Use STRSQL and iSeries Navigator to create SQL objects
- Use INSERT, UPDATE, ALTER, and DELETE statements to maintain a database

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 8-39. Unit summary OD477.0

Notes:

There are other data types available in SQL that we are not covering in this course. They are discussed in the *SQL Reference Manual*:

- Large OBjects (LOBs)
- User Defined Data Types (UDTs)

These data types and other topics are covered in the course OL39/OL390 *DB2 for IBM i SQL Advanced Programming*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 9. Using embedded SQL

Estimated time

08:10

What this unit is about

This unit describes how to code and compile SQL statements in an IBM i program.

This unit will discuss:

- Static hardcoded SQL statements in programs
- Dynamic SQL and the ability to incorporate variables that are not resolved until program runtime
- The ability to generate SQL runtime statements

As we progress through the unit, we will identify the benefits of SQL over traditional methods of file processing, such as the ability of SQL to update a set of records in one statement.

What you should be able to do

After completing this unit, you should be able to:

- Describe how to perform I/O operations in an RPG IV program using embedded SQL
- Describe how to modify an existing RPG IV program to use embedded SQL
- Code embedded SQL statements in an RPG IV program
- List the differences between coding native and SQL-based I/O
- Create an executable program from an embedded SQL RPG IV program
- Use SQL error handling facilities in an RPG IV program
- Explain the purpose of the SQL Communications Area
- Explain the purpose of the SQL Descriptor Area
- List the differences between Static SQL and Dynamic SQL

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

How you will check your progress

Accountability:

- Checkpoint questions
- Exercises

The students will modify an existing RPG IV program and will add SQL statements to access and modify data in an SQL table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- Describe how to perform I/O operations in an RPG IV program using embedded SQL
- Describe how to modify an existing RPG IV program to use embedded SQL
- Code embedded SQL statements in an RPG IV program
- List the differences between coding native and SQL-based I/O
- Create an executable program from an embedded SQL RPG IV program
- Use SQL error handling facilities in an RPG IV program
- Explain the purpose of the SQL Communications Area
- Explain the purpose of the SQL Descriptor Area
- List the differences between Static SQL and Dynamic SQL

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

9.1. Introducing embedded SQL

Instructor topic introduction

The purpose of this topic is to provide an overview of what you can do with embedded SQL and to show an example.

Do not spend time explaining details. All the things that students need to know are in subsequent topics. This topic markets embedded SQL to the students and, will, I hope, motivate the class to experiment. They will perform four exercises of increasing complexity. At the end of this topic, students modify an RPG program and change a CHAIN to a SELECT. This exercise leads them through the process, and a lab review allows you time to explain the things they encountered while in lab.

What students will learn — In lecture, the students learn the purpose of embedded SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

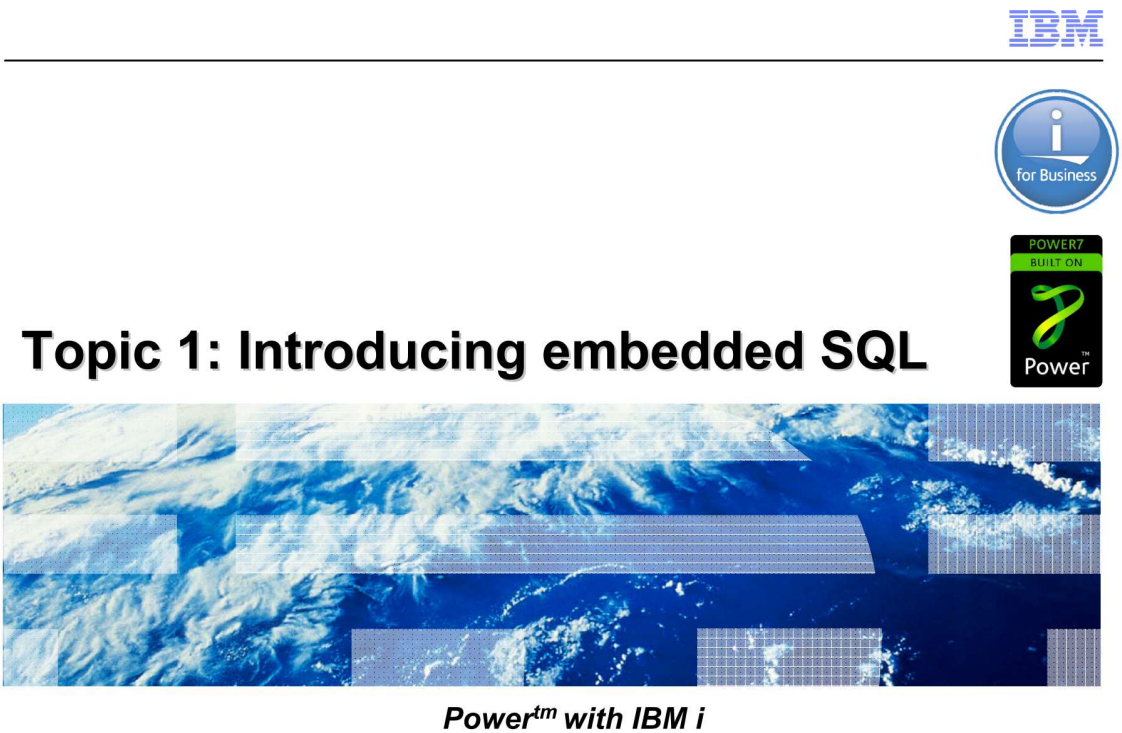


Figure 9-2. Topic 1: Introducing embedded SQL OD477.0

Notes:

Let's examine an example of embedded SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

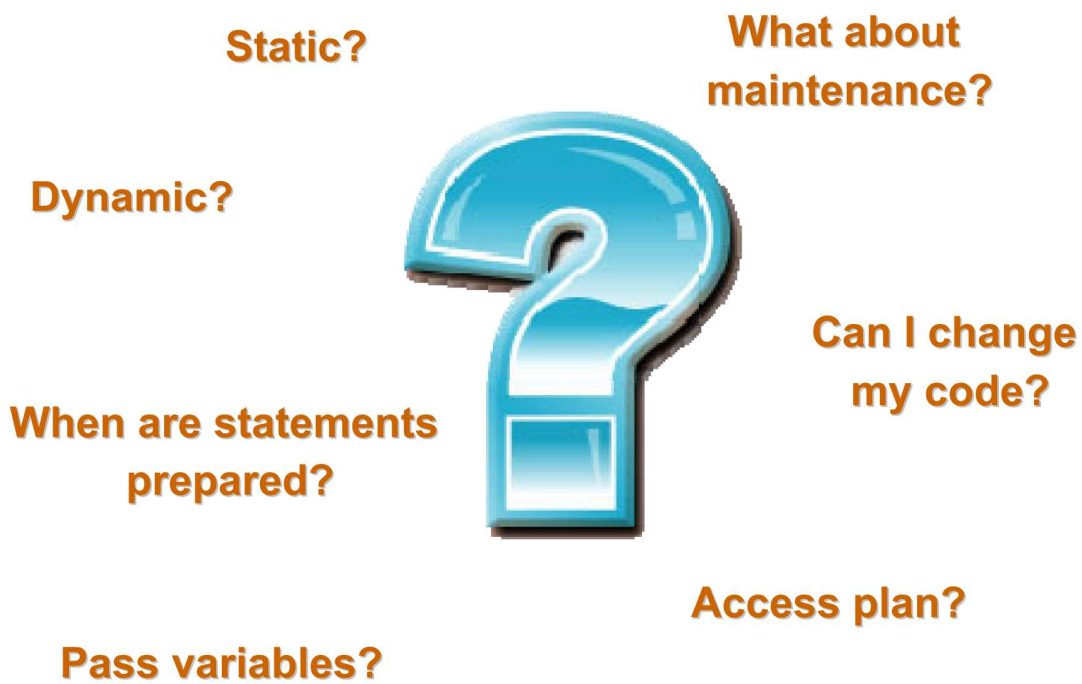
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Embedded SQL: Some questions

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-3. Embedded SQL: Some questions OD477.0

Notes:

There are different forms of embedded SQL. We discuss these forms in this unit and perform several machine exercises that enable your understanding of how to use and when to use embedded SQL in your programs.

We use RPG IV as our basis for embedded SQL. If you have not used RPG IV a great deal, be sure to ask questions. Some of the features of Embedded SQL are not supported by RPG III (also known as RPG/400). Since RPG III does not support pointers, it cannot support the Dynamic Varying List SELECT.

In this topic, we introduce the need for embedded SQL. We show you several examples, and then you perform a simple exercise.

Next, we describe the components of embedded SQL in more detail and explain how the SQL program is created in conjunction with the SQL precompiler. We then discuss some error handling techniques that can be applied to embedded SQL programs and follow that with a lecture on Dynamic SQL. Students perform machine exercises throughout this unit that will allow you to reinforce your understanding of using embedded SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — If any students have been exposed to applications that use embedded SQL, they probably were not aware of the various options and forms of embedded SQL.

Transition statement — Now we will describe the forms of embedded SQL that you may code, **dynamic** and **static**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Coding choices

IBM i

High-level language
- READ/WRITE/CHAIN/UPDATE/DELETE
- Embedded SQL in HLL programs

- OPNQRYF
  - Can be used to define set of records to be processed
  - Program processes subset

- Embedded SQL
  - Can be used to process SQL result sets
  - Can perform all I/O
  - Eliminates need for OPNQRYF

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-4. Coding choices OD477.0

Notes:

You have choices that you can make when you want to process one or more database tables and views in an HLL program. Usually, you are looking for flexibility and efficiency.

If you want to limit the rows processed or join several tables dynamically and do not want to keep the results permanently, you could choose to use OPNQRYF or embedded SQL.

A review of `OPNQRYF` follows:

The Open Query File (`OPNQRYF`) command enables a subset of records from a file to be selected for use during a particular execution of a program. It acts as a filter between the program and the database, so that the program only retrieves records that meet the criteria specified in the `OPNQRYF` command.

The command must be used in conjunction with an HLL program. The program may be run several times, without change, by simply changing the scope of `OPNQRYF` command on the database file.

`OPNQRYF` allows a dynamic definition of a view without using DDS. No permanent object is created, but a temporary file may be created.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

This command is used to do any combination of the following database functions:

- Join records from more than one file, member, and record format.
- The join operation that is performed may be equal or non-equal in nature.
- Calculate new field values by using numeric and character operations on field values and constants.
- Group records by like values of one or more fields, and calculate aggregate functions, such as minimum field value and average field value, for each group.
- Select a subset of the available records. Selection can be done both before and after grouping the records.
- Arrange result records by the value of one or more key fields.

SQL can perform the same functions as `OPNQRYF` but from *within your program* rather than outside your program in CL. When you code embedded SQL, you are using the SQL language as an alternate means of performing I/O. But, SQL can be more powerful than native IBM i HLL I/O operations. Embedded SQL can provide the same power and efficiency as `OPNQRYF`, plus it offers you several unique features:

1. The result set can be composed using a SELECT within your HLL, and the data is processed only once. With `OPNQRYF`, the file is processed once by the `OPNQRYF` command and the subset is then processed by your program. This results in more overhead than using embedded SQL.

   CAVEAT: This result set will require additional processing and may need code outside of RPG IV to be used. We will discuss the option of using Stored Procedures in the next unit.

2. A group of rows can be processed by, for example, a single DELETE or UPDATE SQL statement.
3. An SQL statement can be constructed dynamically in your application program. You pay a performance cost for this feature, but the cost can be offset by the flexibility that you offer in the program. And, the cost to develop and maintain the program may be lower than using only static SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Not only do we need to compare native I/O, OPNQRYF, and embedded SQL, we also want to build a need for embedded SQL.

Details — Emphasize that embedded SQL may produce a result set. RPG programs (at least) cannot deal with the result set without some special coding. As a consequence, ODBC, JDBC, System i Navigator, CLI, and so forth, are often used to process a result set.

Transition statement — Next, let us look at why you might use embedded SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Why use embedded SQL?

IBM i

- SQL can be easier to read and maintain.
- SQL can process multiple records in I/O operations.
- Query Optimizer tunes SQL-based I/O.
- SQL can make code more portable.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-5. Why use embedded SQL? OD477.0

Notes:

Instead of using IBM i native database file operations, such as READ, CHAIN, UPDATE, and DELETE, you can embed SQL statements in your RPG IV programs that process records in IBM i database tables.

SQL is the industry standard for database access and control, and is used by programmers on many different platforms. Some reasons to consider using embedded SQL in RPG IV programs on IBM i systems are:

- SQL can be easier to understand and maintain than the comparable RPG logic.
- SQL I/O behaves more *logically* than RPG. Whereas in RPG, you must be careful not to issue an update before a read or a chain, SQL does not require a SELECT to be performed before an UPDATE.
- In SQL, you always perform an operation on a table or a view. In RPG, you READ the filename but UPDATE the record format.
- SQL statements can perform an operation on a set of rows. A SELECT can produce a set of many rows.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- SQL can simplify the program logic when multiple records are processed in an I/O operation, using UPDATE or DELETE.
- SQL operations are monitored and tuned by the query optimizer, which is enhanced regularly. Thus, SQL I/O automatically takes advantages of the latest database enhancements that you have installed.
- Application migration *may* be simpler when using an industry standard language like SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Transition statement — Next, we show some simple examples of embedded SQL. The students may observe that it is simple and that the same could be done with native I/O. Then, we show students some of the advantages of using SQL compared to native I/O.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

What is embedded SQL?

IBM i

- Code SQL in HLLs
- Can perform same I/O operations as HLL
- Special source member type (SQLRPGLE)
- Preprocessor handles SQL
- Two forms:

1. Static
   - SQL statement(s) known at compile time

2. Dynamic
   - SQL statement(s) created on-the-fly during execution

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-6. What is embedded SQL? OD477.0

Notes:

Embedded SQL offers you many choices in developing your HLL programs. You have already used SQL as a query tool by executing individual statements using System i Navigator (Run SQL Scripts) and the interactive 5250 interface (STRSQL).

You already know how powerful the SQL language is and now you want to know why you should and how you can use SQL in your programs rather than the I/O operations that are native to the HLL.

Let us review the nature of two forms of embedded SQL, static and dynamic.

- **Static SQL**

  The source form of a static SQL statement is embedded within an application program written in a host language such as COBOL or RPG. The statement is **prepared** before the program is executed and the operational form of the statement persists beyond the execution of the program.

  A source program containing static SQL statements must be processed by an SQL precompiler before it is compiled. The precompiler checks the syntax of the SQL

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

statements, turns them into host language comments, and generates host language statements to call the database manager.

The preparation of an SQL application program includes precompilation, the preparation of all static SQL statements, and compilation of the modified source program.

- **Dynamic SQL**

  A dynamic SQL statement is prepared during the execution of an SQL application. The operational form of the statement persists until the last SQL statement leaves the call stack. The source form of the statement is a character string that is passed to the database manager by the program using the static SQL statement PREPARE or EXECUTE IMMEDIATE.

In summary, **Static SQL** is any SQL statement that is fixed in form when the program is compiled. Static SQL can be and is **prepared** by the SQL precompiler for the HLL **before the program is compiled and executed**.

Dynamic SQL is any SQL statement that is not known and is created during execution of the program. These statements must still be prepared but are prepared during execution, **after preprocessing and compilation**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Next, we compare using embedded SQL to using native I/O in an RPG IV program. Before we do this, we need to explain what embedded SQL in its static and dynamic forms are.

The rest of this topic gives the students an overview of what embedded SQL looks like and a basic understanding of the mechanics of coding embedded SQL. At the end of this topic, the students perform a simple exercise to replace an RPG IV CHAIN opcode with an SQL SELECT statement. Do not spend a lot of time explaining details in this topic. We discuss the details starting in the second topic. Simply introduce Embedded SQL and explain how easy it is to replace existing I/O with SQL statements.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Inquiry program: Native I/O

IBM i

```
FApInv_PF  IF   E           K Disk
FEmbed01D  CF   E             Workstn InDDS(Indicators)

D Indicators      DS
D   Exit                         1N   Overlay(Indicators:03)
D   Cancel                       1N   Overlay(Indicators:12)
D   Error                        1N   Overlay(Indicators:90)

 /free

    Exfmt  Prompt;
    Dow  Not Exit;

      Chain PoNbr ApInv_PF;
      Error = Not %Found(ApInv_PF);

      Dow %Found(ApInv_PF);
         Exfmt   Detail;
           Select;
           When  Exit;
           *InLR = *ON;
           Return;

           When Cancel;
           Leave;
           Endsl;
       Enddo;

       Exfmt Prompt;
     Enddo;

     *InLR = *ON;
     Return;

 /end-free
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-7. Inquiry program: Native I/O OD477.0

Notes:

If you have written much RPG IV, you will recognize that this program gets a specific record from the APINV_PF physical file and displays it. In the event of an error (not found condition), the user is prompted to re-enter a valid purchase order number.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

For your reference, here is a display of a subset of the records in the APINV_PF file:

```
04/24/01                          VENDOR LIST                          OL3800
                                                                       DIRTCITY

PO       Vendor  Vendor    PO Total
Number   Number  Invoice     Amount  Net Paid      Date Due   Date Paid

300071   10005   5000       1000.00       .00    04/11/1996  04/11/1996
300072   10005   1005       2000.00       .00    06/30/1996  12/31/9999
300073   10015   1580       3000.00       .00    04/11/1998  04/11/1998
300074   10020   2007       4000.00       .00    05/25/1998  06/01/9999
300075   10025   25697       500.00       .00    03/31/1998  12/31/9999
300076   10030   301         600.00       .00    08/15/1998  12/31/9999
300077   10035   350001      700.00       .00    03/01/1998  03/15/1998
300078   10040   40079       800.00       .00    09/24/1998  12/31/9999
300079   10045   45090       900.00       .00    06/25/1998  12/31/9999
300080   10050   500777      100.00       .00    08/15/1998  12/31/9999
                                                                    More...
```

Here are the field definitions for the file:

```
            Data        Field  Buffer    Buffer        Field
Field       Type       Length  Length  Position        Usage

PONBR       PACKED        6  0       4         1        Both
VNDNBR      PACKED        5  0       3         5        Both
APINVNBR    CHAR             8       8         8        Both
APDATE      DATE            10      10        16        Both
POTOTAMT    PACKED        7  2       4        26        Both
APDISCOUNT  PACKED        5  2       3        30        Both
APNETPAID   PACKED        7  2       4        33        Both
APSTATUS    CHAR             1       1        37        Both
APDATEPAID  DATE            10      10        38        Both
APCHECK#    PACKED        6  0       4        48        Both
APDUEDATE   DATE            10      10        52        Both
```


© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

And, here is the Display file used in the program:

```
A                                       REF(*LIBL/APINV_PF)
A                                       INDARA
A                                       CA03(03)

A          R PROMPT
A                                   3 30'Purchase Order Inquiry'
A                                       DSPATR(HI)
A                                   3 70DATE
A                                       EDTCDE(Y)
A                                   8 20'Enter Purchase Order Number:'
A            PONBR     R     D  I  8 49
A  90                                   ERRMSG('No such Purchase Order foun-
A                                       d - rekey' 90)
A                                  21  8'Press Enter to Continue'
A                                  22  8'F3=Exit'
A                                       COLOR(BLU)

A          R DETAIL
A                                       CA12(12)
A                                   3 30'Purchase Order Inquiry'
A                                       DSPATR(HI)
A                                   3 70DATE
A                                       EDTCDE(Y)
A                                   8 20'Purchase Order:'
A            PONBR     R        O  8 47
A                                  10 20'Vendor Number:'
A            VNDNBR    R        O 10 47
A                                  11 20'Vendor Invoice Reference:'
A            APINVNBR  R        O 11 47
A                                  13 20'Cheque Number:'
A            APCHECK#  R        O 13 47
A                                  21  8'Press Enter to Continue'
A                                  22  8'F3=Exit  F12=Cancel'
A                                       COLOR(BLU)
```

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Allow the students to compare the difference in using native opcodes versus embedded SQL.

Details — To assist you, there are some programs in the course schema that you can demonstrate. They match the visuals we display in this topic, as well as show the students the code. This is the better method to use to show the students embedded SQL versus showing the visuals.

The following members are found in QRPGLESRC. They should be shown to the students in the following order to match the flow of the notebook:

EMBED01RF	Demonstrate inquiry against the APINV_PF using native RPG IV file I/O operations. This is the program that is shown in the visual.
EMBED02RF	Identical program to EMBED01RF but using SQL SELECT statement instead of RPG CHAIN operation.

Do not spend a lot of time explaining the code in the program. Focus the students' attention on the CHAIN opcode followed by the error handling. If you have a class that has a lot of RPG III students, they may not be familiar with the RPG IV syntax and all the new function. They should be able to understand what the program is doing. I suggest that you not attempt to teach them RPG IV if some students are not familiar with it.

Each program has a corresponding compiled object that you can run to demonstrate the behavior and results.

Free format support: Good news for free form RPG IV coders. V5R4 properly supports embedded free form SQL statements. This example was added to illustrate free form coding of embedded SQL. You can still use EMBED01R and EMBED02R to illustrate fixed form RPG IV coding examples.

Transition statement — Next we use embedded SQL to perform the same function as the RPG CHAIN opcode.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Inquiry program: SQL I/O

IBM i

```
FEmbed01D  CF   E             Workstn InDDS(Indicators)

D Indicators      DS
D   Exit                         1N   Overlay(Indicators:03)
D   Cancel                       1N   Overlay(Indicators:12)
D   Error                        1N   Overlay(Indicators:90)

 /free

     Exfmt Prompt;
     Dow Not Exit;

     EXEC SQL
      SELECT VndNbr, ApInvNbr, ApCheck#
      INTO :VndNbr, :ApInvNbr, :ApCheck#
      FROM ApInv_PF
      WHERE PoNbr = :PoNbr;


      Error = (SqlCod <> 0);

      Dow SqlCod = 0;
         Exfmt Detail;
         Select;
         When Exit;
         *InLR = *ON;
         Return;

         When Cancel;
         Leave;
         Endsl;
      Enddo;

      Exfmt Prompt;
      Enddo;

      *InLR = *ON;
      Return;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-8. Inquiry program: SQL I/O OD477.0

Notes:

Notice that the CHAIN has been replaced by an SQL SELECT statement. Error handling must be modified slightly since SQL is now notifying us whether its SELECT was successful or not.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This visual shows the EMBED02RF program. Point out the SELECT performs the exact same function as the CHAIN did. The error indicator is being set when the SQLCOD is not zero rather than using the **%found** BIF.

Point out that the SQL statement is embedded using the EXEC SQL free form statement. This is a free form alternative to using the EXEC SQL and END-EXEC frame surrounding your SQL statement in fixed format RPG IV. The SQL that is coded is the same as you would code in the interactive interface with some additional capabilities.

Also mention (or show) that if you are using fixed form embedded SQL, the SQL prompter is available in SEU by placing the cursor on the SQL statement and pressing F4. The prompt interface is the same here as it was when we used STRSQL.

This source member has a type of SQLRPGLE. This type invokes the SQL precompiler, which builds a temporary RPGLE member in QTEMP (unless you specify otherwise). This temporary member is then processed by the RPG compiler.

In addition, mention the following points:

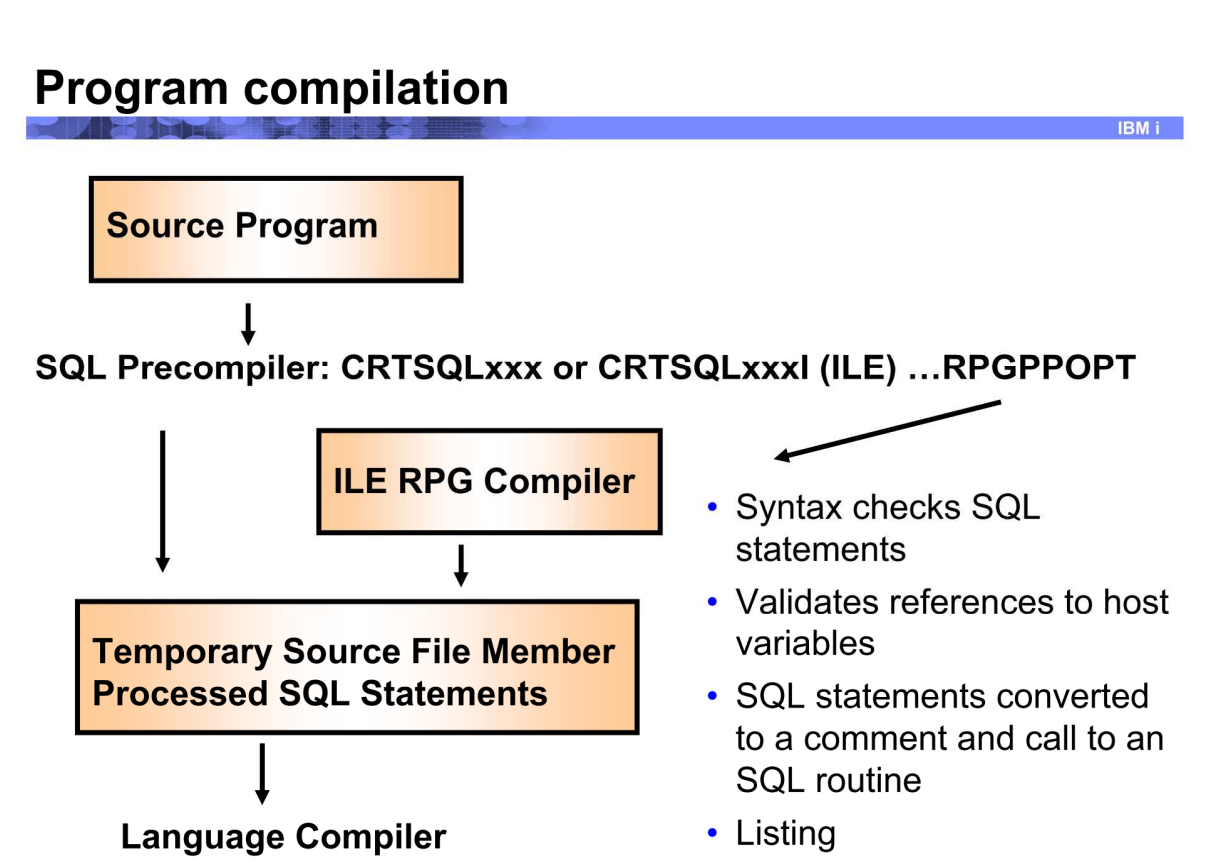
1. The F-spec for the APINV_PF file is no longer required.
2. Since all the columns referenced as host (or program) variables in the SELECT are externally described in the DSPF, no new D-specs are required. This is not always the case.
3. The SELECT statement produces a result set of one row. If it produced more than one row, an SQL error would be returned. You would have to use a cursor to process a multiple row result set.
4. There are no changes made to the APINV_PF table. If you wanted, you could code statements to update data in the table.

Additional information — The result table for a SELECT INTO should contain just one row. Finding more than one row is an error (SQLCod = -811), but one row is still returned. You can control which row will be returned in this error condition by specifying the ORDER BY clause. If you use the ORDER BY clause, the first row in the result table is returned.

Transition statement — Before we go any further, we next briefly cover how to process and compile an HLL that contains embedded SQL.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-9. Program compilation OD477.0

Notes:

The SQL precompiler (or preprocessor), can be invoked using direct call or using option 14 in conjunction with a member type that indicates the source code includes embedded SQL (such as SQLRPGLE).

The command to initiate the precompiler, CRTSQLxxx or CRTSQLxxxI, automatically calls the language compiler, following successful preprocessing, unless the *NOGEN option was selected. For ILE source programs, if the *MODULE option of CRTSQLxxxI is taken, the CRTxxxMOD is automatically called instead of the CRTBNDxxx command.

If an ILE RPG program source member (SQLRPGLE) is compiled with the RPG preprocessor option parameter set accordingly, RPGPPOPT(*LVL1) or RPGPPOPT(*LVL2), the RPG compiler will preprocess the code **before** the SQL precompiler. Preprocessing the source member will allow some compiler directives to be handled before the SQL precompile. The preprocessed source will then be placed in file QSQLPRE in QTEMP and will be used for the SQL precompile.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The SQL precompiler checks the syntax of the SQL statements, turns them into host language comments, and generates the host language statements to call the database manager.

Of note at this point is the fact you must specify that whether you want a listing from the precompiler by entering:

```
Listing output . . . . . . . . . OUTPUT          *PRINT
```

The default is *NONE.

Also of note, the default for the **RPGPPOPT** parameter is *NONE, which means that the RPG compiler is not called for preprocessing.

The SQL precompiler creates a source member (RPGLE) that is passed to the compiler. The language compiler then processes this source member.

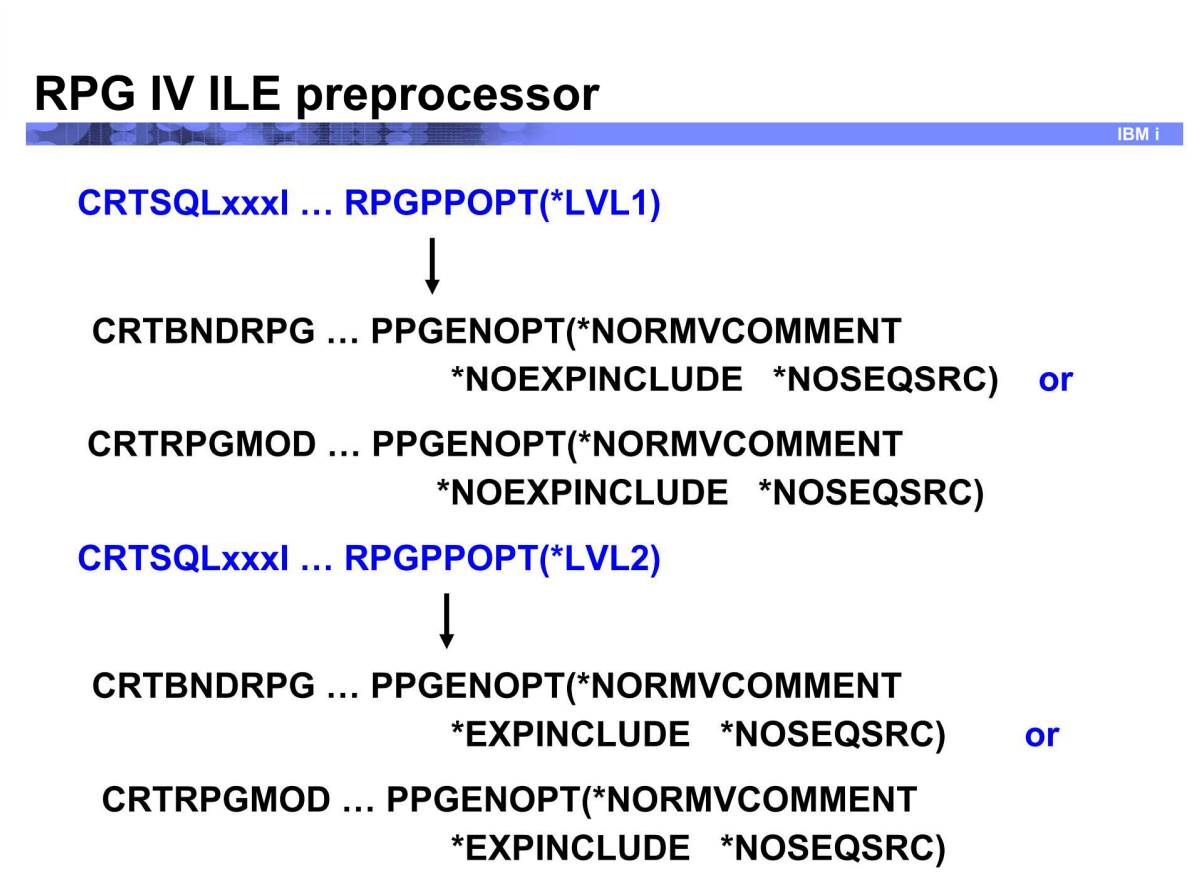
 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Transition statement — We next look at the details of the what the RPG preprocessor does to the source code.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-10. RPG IV ILE preprocessor OD477.0

Notes:

There are two choices (other than *NONE) for RPG preprocessing. By specifying CRTSQLRPGI...RPGPPOPT(*LVL1), you are asking for the corresponding parameter values `PPGENOPT(*NORMVCOMMENT *NOEXPINCLUDE *NOSEQSRC)` to be set for either the `CRTBNDRPG` or the `CRTRPGMOD` commands.

The other choice of CRTSQLRPGI...RPGPPOPT(*LVL2) gives you corresponding parameter values `PPGENOPT(*NORMVCOMMENT *EXPINCLUDE *NOSEQSRC)` for the `CRTBNDRPG` or `CRTRPGMOD` commands.

These values mean the following in terms of preprocessing options:

*NORMVCOMMENT

Preserve comments, blank lines and listing-control directives (for example /EJECT, /TITLE) during preprocessing. Transform source-control directives (for example /COPY, /IF) to comments during preprocessing.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

*EXPINCLUDE

Expand /INCLUDE directives in the generated output file.

*NOEXPINCLUDE

/INCLUDE directives are placed unchanged in the generated output file.

Note:

/COPY directives are always expanded

*NOSEQSRC

If PPSRCFILE is specified, the generated output member has the same sequence numbers as the original source read by the preprocessor

PPSRCFILE

Specifies the source file name and library for the preprocessor output.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — We next look at what the precompiler passes to the RPG IV compiler.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Embedded SQL: Under the covers (1 of 2)

IBM i

```
FEmbed01D  CF   E             Workstn InDDS(Indicators)

D Indicators      DS
D   Exit                         1N   Overlay(Indicators:03)
D   Cancel                       1N   Overlay(Indicators:12)
D   Error                        1N   Overlay(Indicators:90)

D*      SQL Communications area
D SQLCA           DS
D  SQLCAID                       8A   INZ(X'0000000000000000')
D  SQLAID                        8A   OVERLAY(SQLCAID)
D  SQLCABC                      10I 0
D  SQLABC                        9B 0 OVERLAY(SQLCABC)
D  SQLCODE                      10I 0
D  SQLCOD                        9B 0 OVERLAY(SQLCODE)
D  SQLERRML                      5I 0
D  SQLERL                        4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                     70A
D  SQLERM                       70A   OVERLAY(SQLERRMC)
D  SQLERRP                       8A
D  SQLERP                        8A   OVERLAY(SQLERRP)
D  SQLERR                       24A
D   SQLER1                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                     10I 0 DIM(6)   OVERLAY(SQLERR)
D  SQLWRN                       11A
D   SQLWN0                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                       1A   OVERLAY(SQLWRN:*NEXT)
D  SQLWARN                       1A   DIM(11) OVERLAY(SQLWRN)
D  SQLSTATE                      5A
D  SQLSTT                        5A   OVERLAY(SQLSTATE)
D*  End of SQLCA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-11. Embedded SQL: Under the covers (1 of 2) OD477.0

Notes:

This example shows the RPG IV source member (RPGLE type) that is created by the precompiler. By default, this source member is created in QTEMP unless you direct it to be stored elsewhere. By changing the `TOSRCFILE` parameter from the default of QTEMP, the precompiler automatically includes the SQL Communications Area Data Structure in the source. The SQLCA is used to obtain error information.

The **SET OPTION SQLCA=*NO** statement can be embedded in your RPG application program prior to any other SQL statements. This statement specifies that the fields in an SQLCA will be suppressed and therefore not be set after each SQL statement. For SQL error handling, SQLCODE and SQLSTATE should be defined as standalone variables and the SQL statement GET DIAGNOSTICS should be used to capture the return codes after SQL statement execution.

The notes are continued in the next visual.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Show the code generated by the precompiler.

Details — Use the EMBED02SF source member in QRPGLESRC that we saved to show the students what is generated. Point out:

- SQLCA (in this visual)
- SELECT statement is commented out (in next visual)
- RPG statements generated for the SELECT (in next visual)
- Call to QSQROUTE (in next visual)

The SET OPTION SQLCA=*NO statement can be embedded in your RPG application program prior to any other SQL statements. This statement specifies that the fields of the SQLCA will be suppressed and therefore not be set after each SQL statement. For SQL error handling, SQLCODE and SQLSTATE should be defined as standalone variables and the SQL statement GET DIAGNOSTICS should be used to capture the return codes after SQL statement execution.

SET OPTION SQLCA=*NO is only allowed for ILCE C, ILE C++, ILE COBOL, and ILE RPG. This option is not allowed i an SQL function, SQL procedure, or SQL trigger.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Embedded SQL: Under the covers (2 of 2)

IBM i

```
D                 DS
D  SQL_00000               1      2B 0 INZ(128)
D  SQL_00001               3      4B 0 INZ(1)
D  SQL_00002               5      8B 0 INZ(0)
D  SQL_00003               9      9A   INZ('0')
D  SQL_00004              10    128A
D  SQL_00005             129    132P 0 PACKEVEN
D  SQL_00006             133    135P 0
D  SQL_00007             136    143A
D  SQL_00008             144    147P 0 PACKEVEN
 /free

     Exfmt  Prompt;
     Dow  Not Exit;

   //****c SQL
   //****LECT VndNbr, ApInvNbr, ApCheck#
   //****TO :VndNbr, :ApInvNbr, :ApCheck#
   //****OM ApInv_PF
   //****ERE PoNbr = :PoNbr;
 /END-FREE

C                   EVAL      SQL_00005  = PONBR
C                   Z-ADD     -4          SQLER6
C                   CALL      SQLROUTE
C                   PARM                  SQLCA
C                   PARM                  SQL_00000
C     SQL_00003     IFEQ      '1'
C                   EVAL      VNDNBR = SQL_00006
C                   EVAL      APINVNBR = SQL_00007
C                   EVAL      APCHECK# = SQL_00008
C                   END
 /FREE

      Error = (SqlCod <> 0);

      Dow SqlCod = 0;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-12. Embedded SQL: Under the covers (2 of 2) OD477.0

Notes:

The precompiler also:

- Builds a data structure that is designed to map any host variables used to parameters passed to the SQL DB2 processor.
- Comments out the embedded SQL code regardless of which embed method (fixed form or free form) is used.
- Inserts code that calls the SQL DB2 for IBM i database module, “SQLROUTE”. The program variables, know as SQL **Host Variables**, are mapped to the Data Structure and the data structure is passed as a parameter in addition to the SQLCA to SQLROUTE.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Communications Area

IBM i

```
D*      SQL Communications Area
D SQLCA           DS
D  SQLCAID                       8A   INZ(X'0000000000000000')
D  SQLAID                        8A   OVERLAY(SQLCAID)
D  SQLCABC                      10I 0
D  SQLABC                        9B 0 OVERLAY(SQLCABC)
D  SQLCODE                      10I 0
D  SQLCOD                        9B 0 OVERLAY(SQLCODE)
D  SQLERRML                      5I 0
D  SQLERL                        4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                     70A
D  SQLERM                       70A   OVERLAY(SQLERRMC)
D  SQLERRP                       8A
D  SQLERP                        8A   OVERLAY(SQLERRP)
D  SQLERR                       24A
D   SQLER1                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                     10I 0 DIM(6)   OVERLAY(SQLERR)
D  SQLWRN                       11A
D   SQLWN0                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                       1A   OVERLAY(SQLWRN:*NEXT)
D  SQLWARN                       1A   DIM(11) OVERLAY(SQLWRN)
D  SQLSTATE                      5A
D  SQLSTT                        5A   OVERLAY(SQLSTATE)
D*  End of SQLCA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-13. SQL Communications Area OD477.0

Notes:

The SQL Communications Area (SQLCA) can be thought of as SQL's I/O feedback area. It includes error information and return codes.

The SQLCA data structure is automatically built into the RPG IV compiled member by the SQL precompiler.

The SQLCA is updated following the execution of each SQL statement in your RPG program.

The field that we have been using in the coding examples is SQLCod. A successful completion results in a value of zero for SQLCod. However, sometimes a non zero value is expected and is acceptable. For example, a value of 100 means “record not found.” This would be the result expected when you want to add a row to a table.

Note that the SQLCA has SQLCod and SQLCode. Either can be used. We’ll discuss them in more detail in the Error Handling topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Descriptor Area

IBM i

```
D*      SQL Descriptor Area
D SQLDA           DS
D  SQLDAID                 1      8A
D  SQLDABC                 9     12B 0
D  SQLN                   13     14B 0
D  SQLD                   15     16B 0
D  SQL_VAR                       80A   DIM(SQL_NUM)
D                         17     18B 0
D                         19     20B 0
D                         21     32A
D                         33     48*
D                         49     64*
D                         65     66B 0
D                         67     96A
D*
D SQLVAR          DS
D  SQLTYPE                 1      2B 0
D  SQLLEN                  3      4B 0
D  SQLRES                  5     16A
D  SQLDATA                17     32*
D  SQLIND                 33     48*
D  SQLNAMELEN             49     50B 0
D  SQLNAME                51     80A
D*  End of SQLDA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-14. SQL Descriptor Area OD477.0

Notes:

The SQL Descriptor Area (SQLDA) is a special data structure that you must explicitly include in your programs when you are coding Dynamic SQL statements that include SELECT.

Suppose you wrote a general query that allows a user who does not know SQL to specify a WHERE condition interactively. The WHERE condition could be based upon an expression that included an equal, greater than, or less than condition, for example. The host variables used in the condition might change depending upon the option chosen by the user. You could dynamically build the SELECT statement to be executed in your RPG IV program. In other words, your SELECT statement would be different every time it was executed.

Because your SELECT statements are created *"on the fly,"* the precompiler cannot use a data structure that is mapped to your host variables. Rather, you must use the SQLDA to hold the addresses of variables used and their contents.

The SQLDA is discussed in further detail in the Dynamic SQL topic of this unit.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Because RPG III (commonly referred to as RPG/400) does not support pointers, dynamic SQL can only be coded in RPG IV (or ILE RPG).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Change CHAIN to SELECT*



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-15. Exercise OD477.0

Notes:

Perform the machine exercise *Change CHAIN to SELECT*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This is a very basic lab. The students learn how to replace a single I/O opcode with an SQL statement. They are instructed step by step how to do this and they are directed to review all output.

Introduce the lab at this point and explain what they do in this exercise. Reiterate the following points:

1. Students create all of their objects needed for the exercises themselves.
2. The students create personal schemas based on userids assigned to them. They are of the form:

   `User id` `OL38`*nn*
   `Password` **`OL38`**

   `Schema` `OL38`*nn*`COL` **created by student**

   If a student is named team number 01, his or her user id and password would be OL38*01*. The student should follow the exercise instructions and create a schema named OL38*01*COL.
3. Master tables and data are provided to the students. They create some tables and make a copy of others from the student master schema, OL38XXCOL.
4. Remind the students to read each lab thoroughly before performing it.

The students may ask questions of each other or the instructor during the exercise period.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

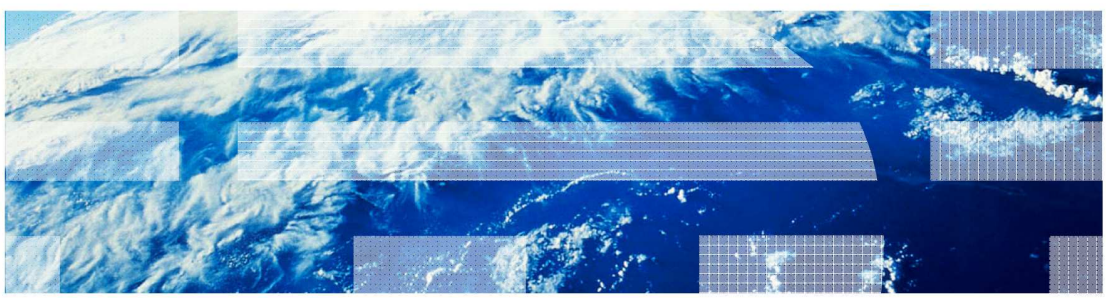
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

9.2. Writing programs that use embedded SQL

Instructor topic introduction

What students will learn — How to code embedded SQL in RPG IV programs and the steps performed by the SQL precompiler.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-16. Topic 2: Writing programs that use embedded SQL OD477.0

Notes:

Let's explore the details of embedding SQL statements and using the precompiler.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — In the first topic, the students received an overview of the capabilities of embedded SQL. In this topic, we will explore the process of coding embedded SQL in more detail.

Details — We explain how to code various SQL statements and how to use the precompiler.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Languages supported

IBM i

- Support for most widely used compilers:
  - RPG IV (ILE RPG) and RPG III (RPG/400)
  - ILE COBOL and OPM COBOL
  - ILE C++ and ILE C
- Also supported:
  - VisualAge C++
  - PL/I
  - REXX interface



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-17. Languages supported OD477.0

Notes:

The languages in the visual support embedded SQL. Of the two RPG compilers, only RPG IV supports dynamic SQL. Dynamic SQL requires a special SQL structure, called the SQL Descriptor Area, or, SQLDA. Since RPG III does not support address pointers, it cannot interface with the SQLDA.

Examples are shown based on RPG IV in this topic.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Depending on your experience with different languages, you may invite questions regarding coding not covered in the course materials. The materials use only RPG IV examples.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Delimiting SQL statements

IBM i

```
C/EXEC SQL    -- inline comment
C+ SELECT VndNbr, ApInvNbr, ApCheck#
C+    INTO :VndNbr, :ApInvNbr, :ApCheck#
C+    FROM ApInv_PF
C* Traditional RPG Comment
C+    WHERE PoNbr = :PoNbr
C/END-EXEC
```

- /EXEC SQL /END-EXEC must frame each SQL statement
- / in column 7
- Can code SQL statement after /EXEC SQL (start in column 17)
- Better to start code on next line
- C+ and at least one blank for each line of SQL
- Can use F4 to prompt SQL on C+ but NOT on C/ lines
- Comments can be inline (--) or on separate line (* in col 7)
- Can INCLUDE SQL source members

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-18. Delimiting SQL statements OD477.0

Notes:

Use the following rules when writing an RPG IV program with embedded SQL statements:

- Enter your SQL statements on the C specification.
- Start your SQL statements using the delimiter /EXEC SQL in positions 7-15, with the / (slash) in position 7.
- You can start entering your SQL statements on the same line as the starting delimiter or on a new line.
- SQL statements can be keyed through position 80.
- Use the continuation line delimiter, a + (plus sign) in position 7 to continue your statements on any subsequent lines.
- Use the ending delimiter /END-EXEC in positions 7-15, with the / (slash) in position 7, to signal the end of your SQL statements.
- Comments can be included within embedded SQL statements. They can be coded on the same line as an SQL statement (or partial statement), using two dashes (**--**); these

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

are *SQL* comments. They can also be included using the RPG convention of an * in column 7.

- You can use the SQL statement INCLUDE (in conjunction with the `INCFILE` parameter of CRTSQLRPGI) to copy SQL statements from a source member into your RPG IV program. We will not experiment with this option, nor will we be using /COPY to copy in RPG IV code that contains embedded SQL in this class.
- If you were to use the /COPY compiler directive to copy in RPG IV code that contains embedded SQL, use the RPG preprocessing option parameter, RPGPPOPT of the `CRTSQLRPGI` command. This parameter determines if the ILE RPG compiler will be called to preprocess the source member (resolving the /COPY directive) before the SQL precompile is run. The options for this parameter are:

  *NONE

  The compiler is not called for preprocessing.

  *LVL1

  The compiler is called for preprocessing to expand /COPY and handle the conditional compilation directives except the /INCLUDE directive.

  *LVL2

  The compiler will be called for preprocessing to expand /COPY and /INCLUDE and handle the conditional compilation directives.

You may code only **one** SQL statement between EXEC SQL and END-EXEC.

SQL statements can be prompted within SEU only when the statement is started on a new (+) line. The prompting is exactly the same as that which is supported in the 5250 STRSQL interactive interface. Prompting for SQL is not supported on the line that includes /**EXEC SQL**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The RPG preprocessing option parameter, `RPGPPOPT` of the `CRTSQLRPGI` command allows you to preprocess your source code with the RPG compiler prior to having the SQL preprocess the code. This would allow you to take advantage of compiler directives like /COPY to copy in RPG IV code that contained embedded SQL. The options for this parameter are:

*NONE

The compiler is not called for preprocessing.

*LVL1

The compiler is called for preprocessing to expand /COPY and handle the conditional compilation directives except the /INCLUDE directive.

*LVL2

The compiler will be called for preprocessing to expand /COPY and /INCLUDE and handle the conditional compilation directives.:

Transition statement — Point out the **Host Variables** (that is, program variables) in this visual. Next we discuss what they are and how to use them in SQL.

Also, point out the two forms of comments that can be coded.

SQL DML statements allow you to retrieve columns of data using SELECT or to modify values in a table using UPDATE, DELETE, or INSERT.

You also need to be able to interact with program variables. Hence, we need **SQL Host Variables** in embedded SQL denoted by adding a leading**:** to the program variable name.

But there may be other issues to think about:

- What if the value of the variable is null? Some languages do not provide null support.

Hence, we need to use Indicator Variables (discussed later) to determine that a value is null.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

V5R4 free-form SQL statements

IBM i

```
EXEC SQL
 SELECT VndNbr, ApInvNbr, ApCheck#
 INTO :VndNbr, :ApInvNbr, :ApCheck#
 FROM ApInv_PF
 WHERE PoNbr = :PoNbr;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-19. V5R4 free-form SQL statements OD477.0

Notes:

With V5R4, embedded SQL can be entirely free format; when you use free-format syntax, each SQL statement ends with a semi-colon. Earlier releases must use the delimiters Exec SQL and End-Exec.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Users of free-form coding on V5R4 systems can use the free-form method of embedding SQL statements.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Host variables

IBM i

```
C/Exec Sql
C+ INSERT INTO APINV_PF VALUES(:PoNbr, :VndNbr:VndNull, :ApInvNbr,
C+ DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, :ApCheck#,
C+ DEFAULT)
C/End-Exec
```

- Defined as variables in your program
- Are referenced in SQL statement:
  - Supply values to SQL statement (INSERT, UPDATE)
  - Supply values to SQL clause (WHERE, HAVING)
  - Receive values in SQL statement (SELECT)
- Prefixed by ':' when referenced in SQL statement
- Can also use SQL SET statement to assign a value to host variable
- Must map to column data type and length and scale
- Host variable names cannot begin with SQ, SQL, RDI, DSN
- Max length of variable name = 64 characters

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-20. Host variables OD477.0

Notes:

Host variables are variables that are defined in your program. For example, they may be fields in externally described files or they may be defined on the D-spec.

Host variables are used to:

- Retrieve data from SQL SELECT statements
- Pass data from your program to SQL INSERT and UPDATE statements
- Use data values supplied by your program to determine the scope of a WHERE clause or a HAVING clause

A host variable used in an SQL statement must be prefixed by a colon (**:**). This must be done in order to differentiate data passed between your program and SQL versus SQL only variables.

Notice in the example how program (host) variables are used in the INSERT statement. We are using host variables to set values in the APINV_PF table. Also, notice that we are loading some columns with the default value for that particular data type.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Also, notice**:VndNbr:VndNull**. This column includes a null indicator that we are using in the program also. We look at null indicators later keep period.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — Focus the students' attention on what Host Variables do; remind them of the naming rules. These same rules apply for SQL variables.

Here students note the use of the null indicator. This indicator can be tested and set so it can be useful, particularly when attempting to avoid errors.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL mapped to host variable data type

IBM i

SQL	RPG IV	D-Spec
CHARACTER	CHARACTER	a or blank
VARCHAR	CHARACTER	a or blank + VARYING
GRAPHIC	GRAPHIC	g
VARGRAPHIC	GRAPHIC	g + VARYING
NUMERIC	ZONED DECIMAL	s
DECIMAL	PACKED DECIMAL	p
SMALLINT	INTEGER	i + length 5
INTEGER	INTEGER	i + length 10
FLOAT (single)	FLOAT	f + length 4
FLOAT (double)	FLOAT	f + length 8
DATE	DATE	d
TIME	TIME	t
TIMESTAMP	TIMESTAMP	z

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-21. SQL mapped to host variable data type OD477.0

Notes:

In the case of date, time, and timestamp data types, SQL creates the date/time subfield using the DATE/TIME format specified on the `CRTSQLRPGI` command or on the H-spec.

ILE RPG does not have variables that correspond to the SQL binary data types.

To create host variables that can be used with these data types, use the SQLTYPE keyword. The SQL precompiler will replace this declaration with an ILE RPG language declaration in the output source member.

Notes regarding the data types:

- RPG IV INTEGER data type length is expressed in digits.
- RPG IV FLOAT data type length is expressed in bytes.
- RPG IV BINARY and VARBINARY data types need the keyword SQLTYPE
- RPG IV must use the keyword SQLTYPE to declare BLOBs and CLOBs

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

BINARY, BLOB and CLOB examples

The following declarations

```
D  MYBINARY      S               SQLTYPE(BINARY:50)
D  MYVARBIN      S               SQLTYPE(VARBINARY:100)

D  MYBLOB        S               SQLTYPE(BLOB_LOCATOR)
D  MYCLOB        S               SQLTYPE(CLOB_LOCATOR)
```

results in the generation of the following code:

```
D  MYBINARY      S          50A
D  MYVARBIN      S         100A   VARYING

D  MYBLOB        S         10U 0
D  MYCLOB        S         10U 0
```

Notes:

1. For BINARY host variables, the length must be in the range 1 to 32766.
2. For VARBINARY host variables, the length must be in the range 1 to 32740.
3. BINARY and VARBINARY host variables are allowed to be declared in host structures.
4. SQLTYPE, BINARY, and VARBINARY can be in mixed case.
5. SQLTYPE must be between positions 44 to 80.
6. When a BINARY or VARBINARY is declared as a standalone host variable, position 24 must contain the character S and position 25 must be blank.
7. The standalone field indicator S in position 24 should be omitted when a BINARY or VARBINARY host variable is declared in a host structure.
8. LOB locators are allowed to be declared in host structures.
9. SQLTYPE, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR can be in mixed case.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

10. SQLTYPE must be between positions 44 to 80.
11. When a LOB locator is declared as a standalone host variable, position 24 must contain the character 'S' and position 25 must be blank.
12. The standalone field indicator S in position 24 should be omitted when a LOB locator is declared in a host structure.
13. LOB locators cannot be initialized.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — We want the students to realize that RPG IV supports the same data types as SQL. RPG/400 supports a subset of these data types and therefore is not as full in function as RPG IV. Also, be aware that RPG/400 does not support pointers, it cannot use embedded SQL that requires use of the SQL Descriptor Area (SQLDA).

Additional information — Unlike RPG III (RPG/400), RPG IV data types map to the SQL data types. When using embedded SQL, be aware of the rules for data types of RPG IV, since this is where the variables are being used. If there is a case where RPG's data type differs from the SQL data type, you should obey the restriction imposed by the lesser of the two.

Transition statement — The next two visuals discuss how to move SQL data into host variables and the data mapping that occurs.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT one row of a table

IBM i

```
C/EXEC SQL

C+   SELECT    EMPNO, LASTNAME, WORKDEPT, SEX
C+    INTO     :EMPNO, :LASTNAME, :WORKDEPT, :SEX
C+      FROM   EMPLOYEE
C+      WHERE  EMPNO = :EMPNO

C/END-EXEC
```

INTO Clause: Used to load *Host Variables with values*

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-22. SELECT one row of a table OD477.0

Notes:

Notice several things in this example:

- We will select a result set of one row because of the WHERE clause.
- The INTO clause takes the row and maps it to the corresponding host variable.
- In the example, the host variables map one-to-one to the SQL columns and match in data type and length.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Ask the class: Do you think that using host variables is a form of dynamic SQL? Many think that this is dynamic SQL. But, this is a form of static SQL and such statements can be prepared by the precompiler.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SELECT INTO host variable

IBM i

- Single Row only
- Values assigned sequentially to host variables
  - Truncation to shorter list
  - Data types must be compatible: No numeric/character conversion
  - Numeric-to-numeric conversion
  - Should check for warnings and errors
- SELECT ..... INTO .... FROM
  - Valid only when return of a single row expected
  - Cannot handle a multi-row result set using this form
  - If multiple rows are returned, you will get:
    - SQL Error Code -811
    - The first row of the set

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-23. SELECT INTO host variable OD477.0

Notes:

Numeric conversion includes conversion between decimal, numeric and binary, as well as adjustment of leading and trailing zeros and truncation of the fractional part of a number.

You can use a variety of statements and clauses to query your data. One way to do this is to use the SELECT INTO statement in a program to retrieve a specific row.

A SELECT INTO statement can include the following:

- The name of each column you want
- The name of each host variable used to contain retrieved data
- The name of the table or view that contains the data
- A search condition to uniquely identify the row that contains the information you want
- The name of each column used to group your data
- A search condition that uniquely identifies a group that contains the information you want

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- The order of the results so a specific row among duplicates can be returned

The INTO clause names the host variables. The value of the first column specified in the SELECT clause is put into the first host variable named in the INTO clause; the second value is put into the second host variable, and so on.

So far, we have seen how to use the SELECT in an RPG IV program. When you code the SELECT in this form, you are expecting only one row to be returned. If your SELECT returns a result set of multiple rows, your program will be sent an SQL error.

We will discuss error handling later.

References:

- *DB2 for IBM i SQL Programming Concepts, Basic Concepts and Techniques*
- *DB2 for IBM i SQL Programming with Host Languages, Common Concepts and Rules for Using SQL with Host Languages*

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — SELECT INTO is used for data retrieval in a program rather than a READ or a CHAIN.

Details — When you are reviewing the rules, you might want to discuss this question:

What if number of columns in the SQL statement is not the same as the number of Host Variables in the INTO clause?

Also, mention that you program differently when you expect a result set of more than one row.

One of the reasons you should always change the default listing option of the precompiler is that this listing tells you of warnings and errors about your SQL statements and your use of host variables.

This visual allows you to point out that coding will be different for a multiple row result set. A cursor must be used to retrieve and then process individual rows. We will cover cursors in a later topic.

We will discuss error handling techniques later. For now, they should be aware that SQL produces its own error messages that you can intercept and handle in your RPG IV programs.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using structure as host variable

IBM i

- Can use structure rather than individual host variables
- Commonly used in:
  - INTO clause of SELECT FETCH
  - VALUES clause of INSERT
- Given EmployeeDS is a Data Structure and value of: EMPNO based on row in EMPLOYEE table:

```
DEmployeeDS       DS
D EmpNo                          6
D GivenName                     12
D MidInt                         1
D SurName                       15
D DeptNo                         3

C/EXEC SQL
C+SELECT * INTO   :EMPLOYEEDS
C+         FROM   EMPLOYEE
C+         WHERE  EMPNO =:EMPNO
C/END-EXEC
```

- To access individual field in structure: EmployeeDS.Surname

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-24. Using structure as host variable OD477.0

Notes:

A structure, consisting of many variables, can be referenced as a single host variable in an embedded SQL statement.

A **host structure** is a grouping of host variables used as the source or target for a set of selected values, for example, the set of values for the columns of a row.

Structures in RPG IV can be based on a data structure. These can be externally described structures.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — We are using only a subset of columns from the employee table. The real table would be more like the one we use in the labs. This structure is only for the purpose of illustration.

Transition statement — A host structure array is an array of host structures used in the multiple-row FETCH statement. We discuss FETCH in a subsequent topic.

Next, let us look again at creating an RPGLE program from a program that includes embedded SQL. We also investigate the role of the SQL precompiler.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL precompiler

IBM i

- Create new SQLRPGLE Source Member
- Code RPG plus embedded SQL statements
- Invoke Precompiler (CRTSQLRPGI) - Create SQL ILE RPG Object – RPG Preprocessor? (RPGPPOPT parm)
  - Invokes RPG ILE compiler for preprocessing (if requested)
  - Inserts copy of SQLCA Data Structure in D-specs
  - Comments out Embedded SQL Statements
  - Inserts Data Structures for Host Variables mapped to SQL variables in embedded SQL statements
  - Adds code to call SQL database routine QSQROUTE

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-25. SQL precompiler OD477.0

Notes:

Source code that contains embedded SQL statements must first be processed by the SQL precompiler. Its job is to replace SQL statements with calls to corresponding SQL function programs.

If you are using compiler directives like /COPY or /INCLUDE and need them to be resolved prior to the SQL precompiler performing the steps below, specify on the `CRTSQLRPGI` command, the parameter `RPGPPOPT(*LVL1) or RPGPPOPT(*LVL2)`. The default for this parameter is `RPGPPOPT(*NONE)`.

The RPG preprocessor performs these steps:

1. Handles compiler directives based upon *LVL1 or *LVL2 request
2. The preprocessed source will be placed in the file, QSQLPRE in QTEMP. This source will be used for the SQL precompile.

The SQL precompiler performs these steps:

1. Creates a copy of the SQL Communications Area data structure in your program

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

2. Builds data structures that will be used to interface between any host variables and SQL variables that you used in your embedded statements
3. Comments out the embedded SQL and replace it with calls to SQL routines, such as QSQROUTE, the main SQL processing routine.
4. Maps host variables to the SQL data structure and passes the appropriate structure plus the SQLCA to the SQL processing routine.

In order to invoke the SQL precompiler, your member type must be SQLRPGLE.

The command executed for an ILE RPG embedded SQL program is `CRTSQLRPGI` (Create SQL ILE RPG Object).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

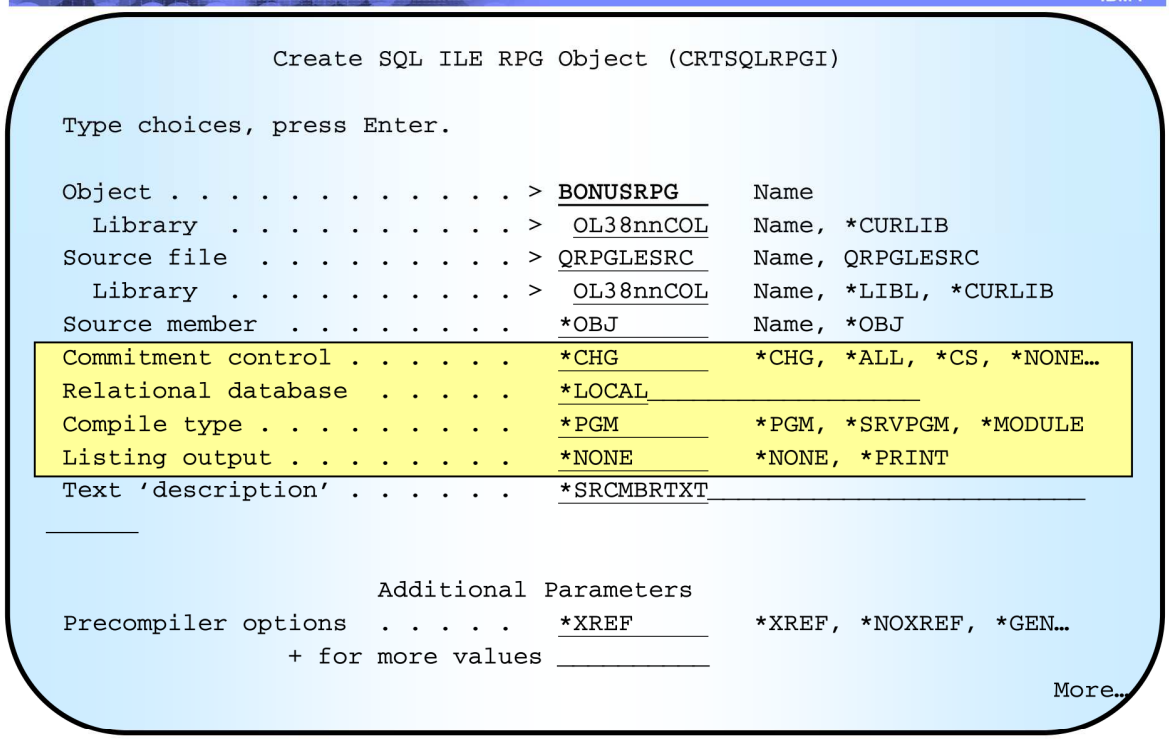
Details — This precompiler is a part of IBM licensed product DB2 Query Manager for IBM i and SQL Development Kit (5770-ST1), which must be installed on the development system. Run time support is included in the operating system.

Transition statement — Next, we take a closer look at how to invoke the precompiler and its parameters.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL precompiler: Parameters (1 of 4)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-26. SQL precompiler: Parameters (1 of 4) OD477.0

Notes:

Some of the more important parameters are highlighted:

- Commitment control (**COMMIT**)

  The default is *CHG. If you are not using commitment control, you should use *NONE.

- Compile type (**OBJTYPE**)

  The default is set to *PGM but if you are using ILE, you may want to specify *MOD or *SRVPGM. When you use PDM option 14 or 15, you can prompt the parameters using F4. When you use option 15 to create a module object, the OBJTYPE will be set to *MOD. This determines if CRTBNDRPG or CRTRPGMOD is run to compile your object.

- Listing output (**OUTPUT**)

  When you are performing initial compiles, set this to *PRINT and the output of the precompiler is placed in your OUTQ. This is very helpful when you have errors. The default is *NONE.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Precompiler options (OPTION)

  This parameter is easily missed. It has a number of options, including the ability to specify the character used for a decimal point plus the naming convention used in your embedded SQL (default is *SYS).

- Close SQL cursor (CLOSQLCSR)

  We have not used a cursor yet, but the default is set to when the activation group ends. You may want to consider the alternate option of closing the cursor when your module (often your executable unit) ends.

  This parameter is used when you do not explicitly close the cursor in your program.

- Date format (DATFMT)

  The default is *JOB. You might prefer to specify the date format to be used, such as *ISO.

- Debugging view (DBGVIEW)

  The ILE source debugger can help you to debug embedded SQL programs, but you must specify *SOURCE to see your source. The source that you see is the original SQLRPGLE member, not the generated RPGLE member.

  It is possible to debug a program even if DBGVIEW(*NONE) was specified in the CRTSQLRPGI command. This can be cumbersome. You would have to save the temporary source created by the precompiler and then compile this source specifying CRTRPGMOD with an appropriate DBGVIEW parameter. Note that CRTSQLRPGI DBGVIEW(*SOURCE) is equivalent to DBGVIEW(*ALL) on CRTRPGMOD or CRTBNDRPG. You are presented with a view of the source code as well. So, the parameter values of *NONE/*SOURCE can be easily misread.

  If DBGVIEW(*SOURCE) is used, you have the choice of view (F15) in STRDBG - including generated source view.

- To source file (TOSRCFILE)

  By default, your generated RPGLE member is stored in QTEMP. While testing and debugging, it might be better to save your generated source in your library.

There is a statement that you can use to set certain options for the precompiler options. It can be thought of as similar to setting ILE RPG options in the H-specification.

For example,

```
SET OPTION COMMIT=*NONE, DATFMT=*ISO
```

Using the SET statement in embedded SQL can prevent you from having to continually modify the CRTSQLRPGI command parameters.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Focus the attention of the students on the parameters that set the level of commitment control (in labs we use *NONE), set the output option to *PRINT (very helpful for debugging), set the format of the date (in labs we use *ISO), and the Debugging level (in lab, we recommend *SOURCE).

Additional information — Note that the compile options used to create a program can be obtained by using DSPPGM/DSPMOD/DSPSRVPGM as well as the SQL command `PRTSQLINF`. The notes apply to this and the next visual.

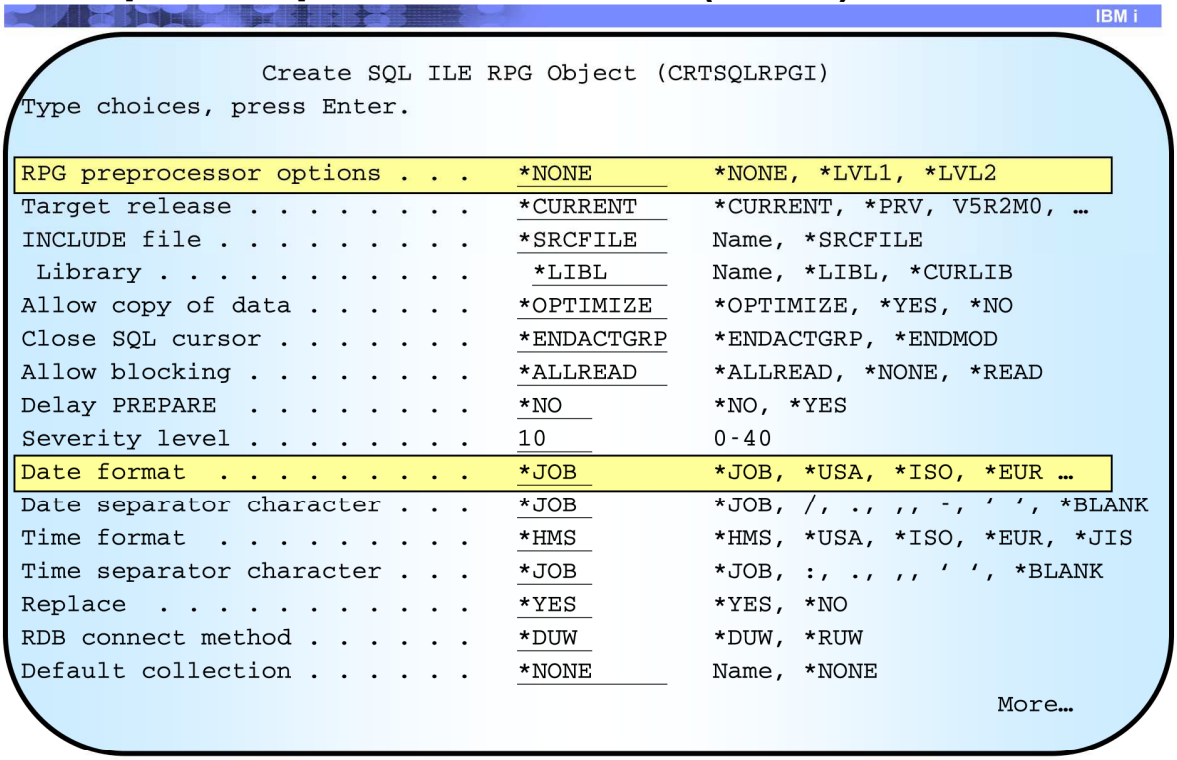
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL precompiler: Parameters (2 of 4)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-27. SQL precompiler: Parameters (2 of 4) OD477.0

Notes:

This second page of SQL compiler parameters highlights the RPG preprocessor options (**RPGPPOPT**) and the date format (**DATFMT**) parameters.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

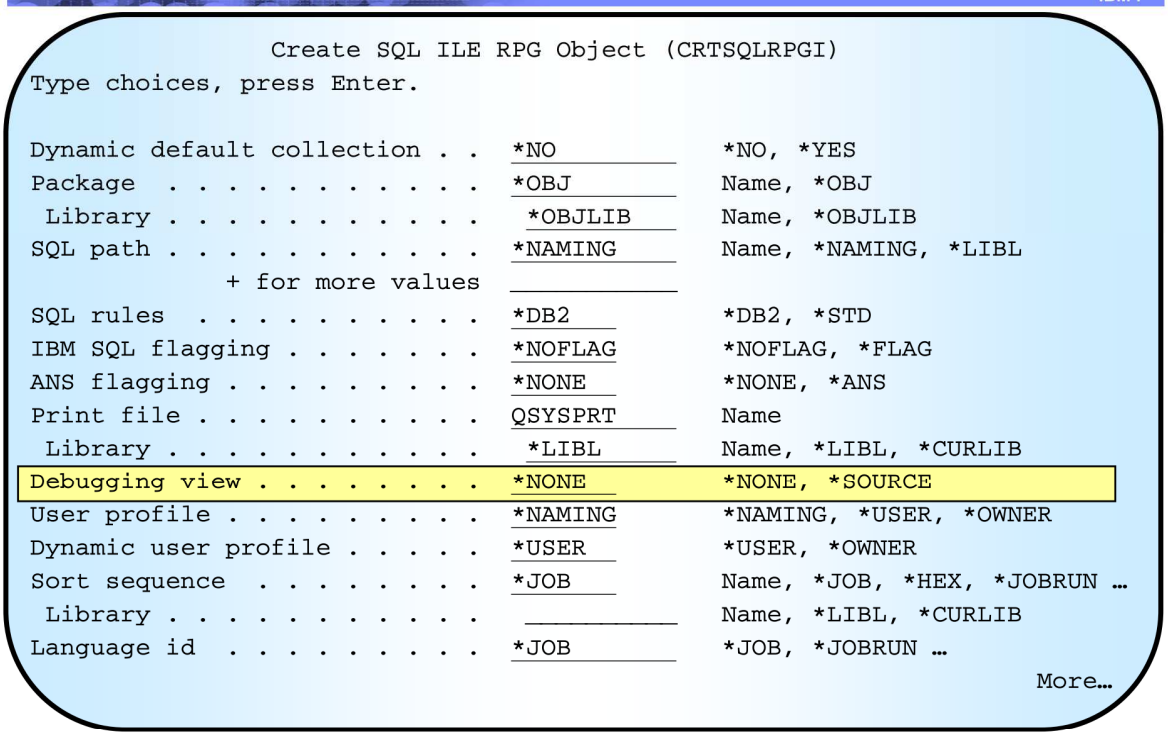
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL precompiler: Parameters (3 of 4)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-28. SQL precompiler: Parameters (3 of 4) OD477.0

Notes:

This third page of SQL compiler parameters highlights the debugging view (**DBGVIEW**) parameter.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

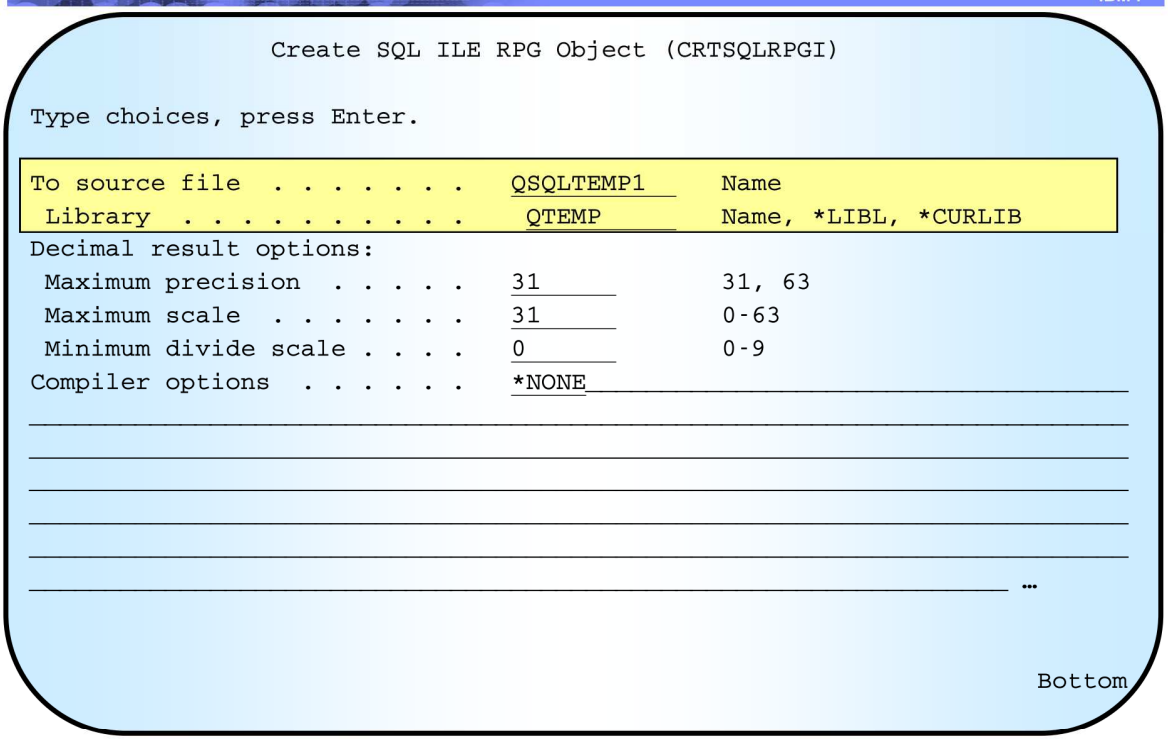
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL precompiler: Parameters (4 of 4)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-29. SQL precompiler: Parameters (4 of 4) OD477.0

Notes:

This fourth page of SQL compiler parameters highlights the source file (**TOSRCFILE**) parameter.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Communication Area

IBM i

```
D*       SQL Communications Area
D SQLCA           DS
D  SQLCAID                       8A   INZ(X'0000000000000000')
D  SQLAID                        8A   OVERLAY(SQLCAID)
D  SQLCABC                      10I 0
D  SQLABC                        9B 0 OVERLAY(SQLCABC)
D  SQLCODE                      10I 0
D  SQLCOD                        9B 0 OVERLAY(SQLCODE)
D  SQLERRML                      5I 0
D  SQLERL                        4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                     70A
D  SQLERM                       70A   OVERLAY(SQLERRMC)
D  SQLERRP                       8A
D  SQLERP                        8A   OVERLAY(SQLERRP)
D  SQLERR                       24A
D   SQLER1                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                     10I 0 DIM(6)   OVERLAY(SQLERR)
D  SQLWRN                       11A
D   SQLWN0                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                       1A   OVERLAY(SQLWRN:*NEXT)
D  SQLWARN                       1A   DIM(11) OVERLAY(SQLWRN)
D  SQLSTATE                      5A
D  SQLSTT                        5A   OVERLAY(SQLSTATE)
D*  End of SQLCA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-30. SQL Communication Area OD477.0

Notes:

The SQLCA is used to feedback the results of the SQL I/O operation. We discuss it further in the following topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — Next, let us review components of the output source member from the precompiler. You can see what the precompiler changes in and adds to your program.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Output of precompiler (1 of 2)

IBM i

SQL I/O Data Structure

```
D                 DS
D  SQL_00000                1      2B 0 INZ(128)
D  SQL_00001                3      4B 0 INZ(1)
D  SQL_00002                5      8B 0 INZ(0)
D  SQL_00003                9      9A   INZ('0')
D  SQL_00004               10    128A
D  SQL_00005              129    132P 0 PACKEVEN
D  SQL_00006              133    135P 0
D  SQL_00007              136    143A
D  SQL_00008              144    147P 0 PACKEVEN
```

- One DS per SQL statement
- Subfields map to host variables used in statement
- SET OPTION SQLCA = *NO
- SQLCODE, SQLSTATE defined as standalone program variables

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-31. Output of precompiler (1 of 2) OD477.0

Notes:

In addition to including a copy of the SQLCA Data Structure in your program, the precompiler adds one data structure that will be used to handle the mapping of host variables to SQL variables. One data structure is built for each embedded SQL statement.

As an alternate to the SQLCA, you could use the SQL statements:

SET OPTION SQLCA = *NO

then using defined, standalone variables like SQLCODE and SQLSTATE you could:

GET DIAGNOSTICS

in order to capture SQL's status after statement execution.

The original SQL statements are commented out and replaced by a call to the SQLROUTE program and a parameter list that includes the SQLCA.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To explain the code generated by the precompiler.

Details — Point out the mapping of the **SQL_nnnn** values to the host variables. Use the data structure and the corresponding assignments using EVAL. For example, before the CALL, we set the value of SQL_0004 to the value of PONBR. After the call, once the SELECT has been performed, the values of VNDNBR is set to the value of SQL_00006, APINVNBR to the value of SQL_00007, and APCHECK# to the value of SQL_00008.

The SQLCA is returned to your program; next, your code can check subfields of the SQLCA for errors and then operate on the data returned by the operation.

The notes apply to this and the next visual.

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Output of precompiler (2 of 2)

IBM i

Generate CALL to QSQROUTE

```
   //****c SQL
   //****LECT VndNbr, ApInvNbr, ApCheck#
   //****TO :VndNbr, :ApInvNbr, :ApCheck#
   //****OM ApInv_PF
   //****ERE PoNbr = :PoNbr;
  /END-FREE
C                   EVAL      SQL_00005  = PONBR
C                   Z-ADD     -4            SQLER6
C                   CALL      QSQROUTE
C                   PARM                    SQLCA
C                   PARM                    SQL_00000
C     SQL_00003     IFEQ      '1'
C                   EVAL      VNDNBR = SQL_00006
C                   EVAL      APINVNBR = SQL_00007
C                   EVAL      APCHECK# = SQL_00008
C                   END
```

- Comment out SQL statements
- Set up parm list
- Pass SQLCA

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-32. Output of precompiler (2 of 2) OD477.0

Notes:

Notice how the SQL statement has been commented and a call to `QSQROUTE` with parameters has been inserted.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Transition statement — In the next topic, we discuss error handling and some of the techniques that you can use to manage problems within your programs.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.3. Error handling

Instructor topic introduction

This topic introduces the facilities available within SQL to assist in anticipation of errors. Because the class is only two days in duration at this time, there is no topic that shows you how to debug errors using the ILE Source debugger. This topic could be added once the class is extended in duration. Further, we do not discuss RPG IV error handling (MONITOR, ILE Condition Handlers, etcetera) in conjunction with SQL error handling facilities.

The database class that follows this, *OL39/OL390*, *DB2 for IBM i Advanced SQL Programming*, considers some of these matters.

What students will learn — The basics regarding what SQL error handling facilities and messages are available, where they are, and how to use them in RPG IV programs.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Figure 9-33. Topic 3: Error handling OD477.0

Notes:

Let's cover some basic error handling techniques.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

What is available?

IBM i

- SQLCA
  - SQLCODE - error codes - to handle specific error
    - Alternate RPG name SQLCOD
  - SQLSTATE - SQL state can isolate a class of error
    - Alternate RPG name SQLSTT
- WHENEVER
  - Used to monitor for errors
  - Can use to monitor for NOT FOUND, SQLERROR, SQL WARNING

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-34. What is available? OD477.0

Notes:

The SQL Communications Area (SQLCA) is your primary tool to use to view feedback about error conditions following the execution of an individual embedded SQL statement.

After the execution of each embedded SQL statement, the SQLCA is updated. Remember that the SQLCA is one of the parameters passed to and from the QSQROUTE program.

There are different ways to handle errors. You can bypass them (CONTINUE), GOTO a label in your program, or handle them explicitly after each SQL statement. We will show you these techniques.

No matter what you do, you will soon become familiar with SQLCODE and SQLSTATE, the subfields of the SQLCA that contain vital error information that you can use in your RPG IV programs.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce the topic. Emphasize that SQLCODE and SQLSTATE are subfields within the SQLCA data structure in an RPG IV embedded SQL program. Also emphasize that it is not necessary to code WHENEVER statements, which tend to be generic in nature (but better than nothing). Explicit error handling can be coded in your program following each SQL statement.

Remind your students that there is an alternative error handling that we've mentioned, but do not discuss in detail in the class. The SQL statement GET DIAGNOSTICS could be used with standalone variables defined in your program. Refer your students to the SQL Reference manual for more details.

Transition statement — Let us look at the SQLCA and specifically the SQLCODE and SQLSTATE variables in further detail.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Communication Area

IBM i

```
D*       SQL Communications Area
D SQLCA           DS
D  SQLCAID                       8A   INZ(X'0000000000000000')
D  SQLAID                        8A   OVERLAY(SQLCAID)
D  SQLCABC                      10I 0
D  SQLABC                        9B 0 OVERLAY(SQLCABC)
D  SQLCODE                      10I 0
D  SQLCOD                        9B 0 OVERLAY(SQLCODE)
D  SQLERRML                      5I 0
D  SQLERL                        4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                     70A
D  SQLERM                       70A   OVERLAY(SQLERRMC)
D  SQLERRP                       8A
D  SQLERP                        8A   OVERLAY(SQLERRP)
D  SQLERR                       24A
D   SQLER1                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                       9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                     10I 0 DIM(6)   OVERLAY(SQLERR)
D  SQLWRN                       11A
D   SQLWN0                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                       1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                       1A   OVERLAY(SQLWRN:*NEXT)
D  SQLWARN                       1A   DIM(11)  OVERLAY(SQLWRN)
D  SQLSTATE                      5A
D  SQLSTT                        5A   OVERLAY(SQLSTATE)
D*  End of SQLCA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-35. SQL Communication Area OD477.0

Notes:

All SQL I/O error handling is based upon what is fed back using the SQLCA. The two main fields are SQLCODE and SQLSTATE. SQLSTATE (SQL state) is part of the ANSI standard and therefore is common across various platforms. If you are implementing applications cross platform, SQL state should be considered.

If you are implementing embedded SQL in the IBM i world, you may find that SQLCODE is more familiar because SQLCODE messages map to the QSQLMSG file.

The subfields in SQLERR and SQLWRN also contain useful information:

- SQLERR: Contains six integer variables that provide diagnostic information. SQLER3, for example, for INSERT, UPDATE, and DELETE statements, shows the number of rows affected. For a FETCH statement, SQLER3 contains the number of rows fetched.
- SQLWRN: Contains 11 warning indicators, each containing a '*W*' or an '*N*'. For example, **sqlwn3** contains '*W*' if the number of columns is larger than the number of host variables.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If we have the situation where we have **STRING TRUNCATION ON COLUMN ASSIGNMENT TO HOST VARIABLE**, the SQLCA provides the following information:

- SQLCOD = 0;
- SQLSTT = '01004';
- SQLWN1 (and SQLWN0) = 'W'

For detailed information regarding the contents of the SQLCA, please reference the documentation noted below.

References:

- *SQL Messages and Codes, IBM i Information Center*

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQLCODE and SQLSTATE

IBM i

SQLCODE	SQLSTATE Class Code	Type of Error
0	'00'	Successful
>0	'01'	Warning
100	'02'	End of Data (Row not Found)
<0	>='03'	Error

- SQLCODE and SQLSTATE are ANSI/ISO names
- SQLSTATE errors grouped by "Class Code" (First two bytes)

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-36. SQLCODE and SQLSTATE OD477.0

Notes:

The SQLCODE (five digits) and SQLSTATE (five characters) fields are part of the SQL Communications Area (SQLCA) and contain SQL return codes.

These values are set by the database manager after each SQL statement is executed.

An application can check the SQLCODE or SQLSTATE value to determine whether the last SQL statement was successful.

When embedded in an HLL program, SQL does not communicate directly with the end user (fortunately). Rather, SQL returns error codes to the application program when a problem occurs.

If SQL encounters an error while processing an embedded SQL statement, the first two characters of the SQLSTATE will not be **'00'**, **'01'** or **'02'**; the SQLCODE will be a negative number.

If SQL encounters a warning that is still a valid condition while processing an embedded SQL statement, the SQLCODE is set to a positive number and bytes one and two of the SQLSTATE are set to **'01'**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If your embedded SQL statement is processed without encountering an error or warning condition, the SQLCODE returned is a 0 and SQLSTATE is **'00000'**.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — Cover the purpose of SQLCODE and SQLSTATE. Make the point that they are part of IBM's implementation of SQL and are cross platform. But, as we will show the class, SQLCODE can vary, while SQLSTATE follows an ANSI standard and is consistent.

Transition statement — Now that we have discussed the purpose of these subfields, we next discuss how they are implemented and used on the IBM i, specifically in RPG IV.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQLCODE and SQLSTATE

IBM i

- SQLCODE varies by platform
- SQLSTATE is standard across platforms
- RPG IV
  - SQLSTATE named SQLSTT = 5-bytes character
  - SQLCODE named SQLCOD = 4-byte binary (numeric) +/- 5-digits
  - Part of SQLCA Data Structure - always included automatically
  - SQLCOD maps to message file QSQLMSG:

SQLCODE	QSQLMSG
30070	SQ30070
00445	SQL0445
-00112	SQL0112

- Examples:

Error Description	SQLCOD	SQLSTT	Error Class
INVALID DATE/TIME STRING	00181	'01534'	Warning
JOIN EXPRESIION INVALID	-00338	'42972'	Error

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-37. SQLCODE and SQLSTATE OD477.0

Notes:

SQLCODE and SQLSTATE are returned to your program in RPG IV's SQLCA data structure. SQLSTATE follows the SQLSTATE ANSI standard and provides application programs with common return codes for common error conditions found among the IBM relational database systems.

It is important to know that the values of SQLCODE and SQLSTATE are different. Generally, programmers use SQLSTATE to handle a class of error as a group and SQLCODE when they want to handle a specific error. Every SQLCODE maps to a corresponding message in the QSQLMSG message file in library QSYS.

The message ID for a message in QSQLMSG is constructed by appending the **absolute value** (5 digits) of the SQLCODE to SQ and changing the third character to an **L** when the first digit of the SQLCODE is a **0**. For example, if the SQLCODE is 30070, the message ID is SQ30070.

The SQLCA is optional. One could use the SET OPTION SQLCA = *NO SQL statement and define standalone variables for SQLCODE and SQLSTATE in place of the SQLCA.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Examples of SQLCODE and SQLSTATE

IBM i

Error Condition	SQLCODE	SQLSTATE
SQL statement not allowed	-84	'42612'
More than one row in set	-811	'21000'
Successful	000	'00000'
# of INTO host vars incorrect	+30	'01503'
No row found	+100	'02000'

- SQLSTATE: First two characters are error class code

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-38. Examples of SQLCODE and SQLSTATE OD477.0

Notes:

The values of SQLCODE and SQLSTATE can be used in your RPG program to handle errors:

- A DoW loop can be executed while SQLCODE = 0. While we experience no SQL errors, we continue in the loop.
- A record can be inserted when SQLCODE = 0100 or SQLSTATE = '02000'. In some cases, these values also signal end of data. For example, when you are processing a multiple record result set using FETCH NEXT, SQLCODE = 0100 signals that you have completed processing the set.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Give the students the understanding that SQLCOD, SQLCODE, and SQLSTT, SQLSTATE can be used interchangeably.

Additional information — Point out the use of “SQLCOD” or “SQLCODE” for RPG IV.

I can think of other things that would help, such as making the CALLs prototyped CALLPs and organizing at least some of the routines that are called in service programs.

Transition statement — Often when you want to handle a class of errors generically, SQLSTATE is better because errors are grouped in categories using the first two digits.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL state error classes

IBM i

Class Code	Description
00	Unqualified Successful Completion
01	Warning
02	No Data
03	SQL Statement Not Yet Complete
07	Dynamic SQL Error
08	Connection Exception
0A	Feature Not Supported
09	Invalid Token
:	
26	Invalid SQL Statement Identifier
:	
2F	SQL Function Exception
:	
34	Invalid Cursor Name
:	
3C	Ambiguous Cursor Name
:	

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-39. SQL state error classes OD477.0

Notes:

This visual shows some of the class errors that are fed back in SQLSTATE. If you research class “02”, you will discover that many data exception errors are covered in this category.

However, you will also find that the messages are the same for many of the values of SQLSTATE.

The following subset of error codes are meant to guide you in your use of SQLCODE versus SQLSTATE. SQLCODE has more specific error codes. However, SQLSTATE can be used to generically handle a class of message.

SQLSTATE 22007 (SQLCODE -181) -

Explanation: Value in date, time, or timestamp string dropped.

SQLSTATE 22001,22003, 22023, 22024 (SQLCODE -302)

Explanation: Conversion error on input host variable.

SQLSTATE 24507 (SQLCODE -226)

Explanation: Current row deleted or moved for cursor

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQLSTATE 22001, 42806 (SQLCODE -303)

Explanation: Host variable &1 not compatible with SELECT item.

SQLSTATE 22003, 22023, 22504 (SQLCODE -304)

Explanation: Conversion error in assignment to host variable &2.

SQLSTATE 22021 (SQLCODE -331)

Explanation: Character conversion cannot be performed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Give the students a perspective on using error message codes in SQLSTATE versus SQLCOD or SQLCODE. There are certain situations where one is a better fit than the other. If portability is important, use SQLSTATE. SQLSTATE is also useful for handling exceptions that are grouped in a class. You can handle a class with very little code.

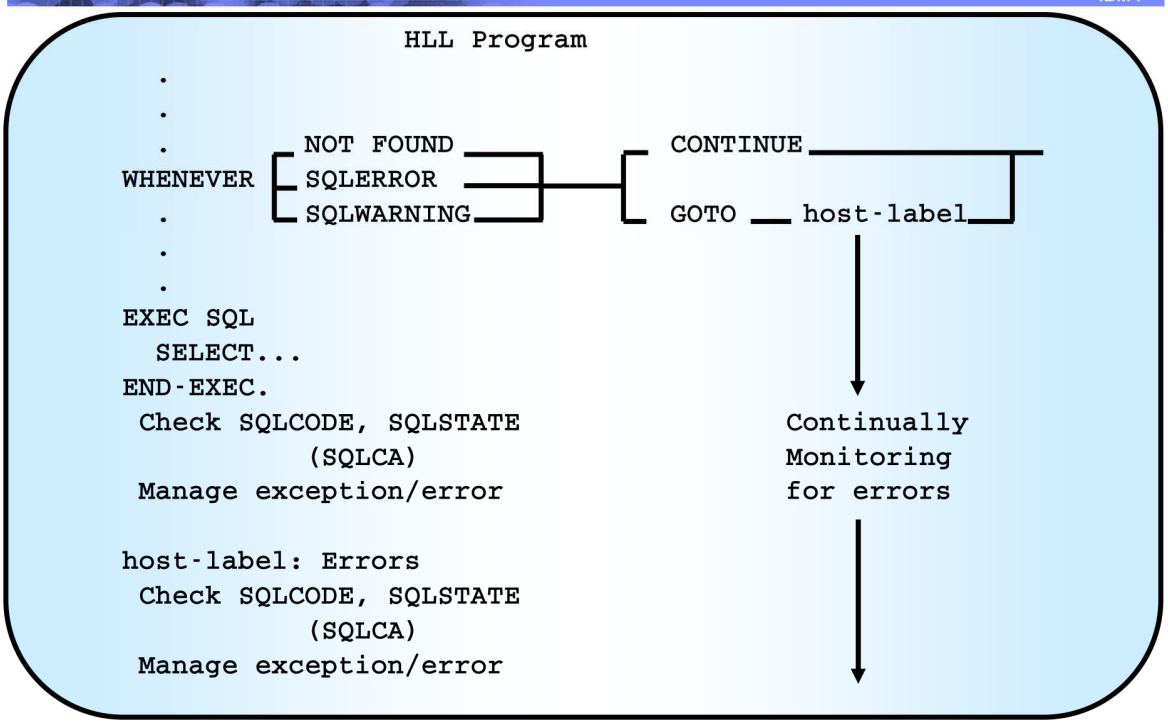
Otherwise, SQLCOD and SQLCODE are more precise.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Monitoring for errors using WHENEVER

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-40. Monitoring for errors using WHENEVER OD477.0

Notes:

The WHENEVER statement continually checks for errors. For instance, if a WHENEVER NOT FOUND statement is coded, it is in effect for all NOT FOUND conditions until another WHENEVER NOT FOUND is encountered. This is according to the listing sequence of the statements in the program, not the execution sequence. There may be one of each type of WHENEVER statement in effect:

```
WHENEVER NOT FOUND
WHENEVER SQLERROR
WHENEVER SQLWARNING
```

In the absence of a WHENEVER being specified, there is an implicit

```
WHENEVER    condition       CONTINUE
```

in effect. In other words, the condition is ignored.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out that SQL takes a different approach to error handling than RPG does. Whereas an un-handled RPG error will cause the Default Error Handler to appear on your screen, SQL will continue in the event of an error because of the implicit WHENEVER.... CONTINUE.

If you have unique WHENEVERs coded for the three classifications of errors, every SQL error will be handled by the appropriate WHENEVER regardless of where it appears in your program.

Assume you have more than one WHENEVER that handles the same class of error. For example, a WHENEVER NOT FOUND is coded at line 25 and a second WHENEVER NOT FOUND is coded at line 50. SQL will use the first WHENEVER when it is executing code between lines 26 and 49. For line 50 and after, it will use the second WHENEVER. For errors encountered in lines 1 through 24, the implicit WHENEVER will apply.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Scope of WHENEVER

IBM i

```
C/EXEC SQL
C+ whenever sqlerror continue
C/end-exec

:
:
:

C/exec sql
C+ whenever sqlerror goto hdlerr
c+end-exec
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-41. Scope of WHENEVER OD477.0

Notes:

Since WHENEVER can include a GOTO, it is sensitive to where it is placed in your code. You must manage going back to the point in your code where you want to restart execution. Also, each form of WHENEVER:

1. SQLERROR
2. NOT FOUND
3. SQLWARNING

is in force until the next WHENEVER of the same form is encountered in the execution. This is true regardless of the flow of execution. You must be extremely careful in how you use WHENEVER.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — The use of WHENEVER can cause problems. This is a caveat based on my experience that you can use or, if you have better ideas, use your own (and forward them to me!)

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using SQLCODE (1 of 4)

IBM i

```
FEmbed03D  CF   E             Workstn InDDS(Indicators)

Indicators      DS
D   Exit                         1N   Overlay(Indicators:03)
D   DltRow                       1N   Overlay(Indicators:04)
D   Cancel                       1N   Overlay(Indicators:12)
D   AllowDlt                     1N   Overlay(Indicators:80)
D   Error                        1N   Overlay(Indicators:90)

C                   Exfmt      Prompt
C                   Dow        Not Exit

C/EXEC SQL
C+ SELECT VndNbr, ApInvNbr, ApCheck#
C+    INTO :VndNbr, :ApInvNbr, :ApCheck#
C+    FROM ApInv_PF
C+    WHERE PoNbr = :PoNbr
C/END-EXEC
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-42. Using SQLCODE (1 of 4) OD477.0

Notes:

This example is a purchase order maintenance application. The user enters a purchase order number which is used in a SELECT to retrieve the row, and then can add, delete, or change a purchase order.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Point out that WHENEVER is not used in this program and that SQLCODE provides excellent in program error handling capabilities.

Details — Run the program EMBED03R in the class schema. Demonstrate the Add, Change, and Delete scenarios. Make sure you can explain the logic of the program to the students.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using SQLCODE (2 of 4)

IBM i

```
** 1
C                   Eval      Error = (SqlCode <> 0 And SqlCode <> 100)
 ** 2
C                   If        Not Error
 ** 3
C     3             If        (SqlCode = 100)
C                   Clear                   Detail
 ** 4
C                   Eval      AllowDlt = *OFF
C                   Else
 ** 5
C                   Eval      AllowDlt = *ON
C                   Eval      PoNbr2   = PoNbr
C                   Exfmt     Detail
 **
C                   Select
C                   When      Exit
C                   Eval      *InLR = *ON
C                   Return
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-43. Using SQLCODE (2 of 4) OD477.0

Notes:

This program illustrates the use of SQLCODE to determine what action to take:

1 — We set the **Set Error Indicator**.

2 — We test whether there is an error condition; if there is no error, we continue.

But, if there is an error condition, SQLCODE is neither 0 (successful) nor 100 (record not found). We can be certain that we have some sort of error and we handle it with the error indicator, simply return to the caller. Perhaps we might display an *Error* message, and then prompt the user for another Purchase Order number.

3 — We test whether the value of **SQLCODE = 100**.

If **SQLCODE = 100**, we have a no record found situation. If this is the case, we can assume that the user wants to add a new purchase order and we present a cleared detail format on the terminal.

4 — There is another situation we can handle here also. If there is no record found, then a delete is not possible, so we set the *AllowDlt* indicator off.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

5 — We also handle the opposite case here and if a record is found, that is **SQLCODE <> 100**, we set the *AllowDlt* indicator on.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using SQLCODE (3 of 4)

IBM i

```
 ** F12 pressed - ignore
C                   When      Cancel

 ** F4 pressed - DELETE row
C                   When      DltRow
C/Exec Sql
C+ DELETE FROM APINV_PF WHERE PoNbr = :PoNbr
C/End-Exec

 ** Row not found - INSERT
C                   When      (SqlCode = 100)
C/Exec Sql
C+ INSERT INTO APINV_PF VALUES(:PoNbr, :VndNbr, :ApInvNbr,
C+ DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT,
C+ :ApCheck#,DEFAULT)
C/End-Exec
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-44. Using SQLCODE (3 of 4) OD477.0

Notes:

Page three of our coded example shows the logic that handles F12, F4 and actual insert of a new row to the table logic. Note how the SQL INSERT is conditioned on SQLCODE = 100.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using SQLCODE (4 of 4)

IBM i

```
**    6
**  Row found - UPDATE
C                   When      (SqlCode = 0)
C/Exec Sql
C+ UPDATE APINV_PF SET PONBR = :PoNbr, VNDNBR = :VndNbr,
C+ APINVNBR = :ApInvNbr, APCHECK# = :ApCheck#
C+ WHERE PONBR = :PoNbr
C/End-Exec
C                   Endsl
C                   Endif

C                   Exfmt     Prompt
C                   Enddo

C                   Eval      *InLR = *ON
C                   Return
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-45. Using SQLCODE (4 of 4) OD477.0

Notes:

6 SQLCODE = 0 — If the SELECT returns an SQLCODE = 0, then we found the row and can update the purchase order.

Once indicators have been set, we use a SELECT/WHEN group to execute the desired changes to the APINV_PF table.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Error handling: Considerations

IBM i

- SQLCODE: IBM i like; maps to QSQLMSG file
- SQLSTATE: ANSI standard
- WHENEVER: Very generic; 'avoids' handling error
- Subroutine or service program?
- Typical Errors
  - SQLCODE = 0 or SQLSTATE = '00000' (all OK)
  - SQLCOD E= 100 or SQLSTATE = '02000' (record not found)
  - SQLCODE < 0 (there is a problem)

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-46. Error handling: Considerations OD477.0

Notes:

How should you handle errors? As a start, you should check errors immediately after each and every SQL statement. While WHENEVER is better than nothing, it is a generic handler and may not allow you to handle and work around an error situation in your applications.

Many programmers code error handling in a subroutine. Over time, as they add more and more logic, the subroutine method may not be flexible enough to handle errors encountered across many programs. At this point, an error handling procedure is a good choice.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — State options and some initial ideas and considerations.

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Handling SQL errors using SQLCODE

IBM i

```
:
:
C+ SELECT VndNbr, ApInvNbr, ApCheck#
C+     INTO :VndNbr, :ApInvNbr, :ApCheck#
C+     FROM ApInv_PF
C+     WHERE PoNbr = :PoNbr
C/END-EXEC

C                   If        Not (SqlCode <> 0 And SqlCode <> 100)
C                   If        (SqlCode = 100)
C                   Clear                   Detail
C                   Eval      AllowDlt = *OFF
C                   Else
C                   Eval      AllowDlt = *ON
C                   ExSR      SQLErrSub
C                   Endif
:
:
C/Exec Sql
C+ DELETE FROM APINV_PF WHERE PoNbr = :PoNbr
C/End-Exec

C                   If        SQLCODE <> 0
C                   ExSR      SQLErrSub
C                   EndIf
:
:
C     SQLErrSub     BegSR
C                   Eval      Error = (SqlCode <> 0 And SqlCode <> 100)
C                   Eval      ErrMsg = ErrMsg + %char(SQLCode)
C                   EndSR
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-47. Handling SQL errors using SQLCODE OD477.0

Notes:

A good practice to implement would be to check for SQL errors following each and every SQL statement. Error codes that are acceptable in some situations might not be acceptable in others. For example, SQLCODE = 100 (row not found) is not OK after an attempted DELETE statement.

A row not found error can be acceptable following a SELECT statement and will influence the logic flow following the SELECT.

Notice that this example uses SQLCODE and an error handling subroutine.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To show the students an error handling strategy.

Details — Emphasize that it is important to test for errors following each and every SQL statement. WHENEVER is similar to using a *PSSR and is not really specific. Deliberate testing for all the errors that could occur after a statement is executed is a much better approach. This process is iterative and the error handling subroutine will be enhanced over time.

Notice that we are using SQLCODE in this example. Another choice is to use SQLSTATE since it is the industry standard.

Also, perhaps a subroutine is not the best way to code error handling. Over time, the students should consider coding an SQL error handling procedure that can be bound to embedded SQL programs.

Perhaps this module could be registered as an ILE Condition Handler as well.

Additional information —

Transition statement — Next, we look at error handling using SQLSTATE.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Handling SQL errors using SQLSTATE

IBM i

```
:
C+ SELECT VndNbr, ApInvNbr, ApCheck#
C+     INTO :VndNbr, :ApInvNbr, :ApCheck#
C+     FROM ApInv_PF
C+     WHERE PoNbr = :PoNbr
C/END-EXEC

C                    If        Not (SqlState <> '00000'
C                                   And SqlState <> '02000')
C                    If        (SqlState = '02000')
C                    Clear                  Detail
C                    Eval      AllowDlt = *OFF
C                    Else
C                    Eval      AllowDlt = *ON
C                    ExSR      SQLErrSub
C                    Endif
:
:
C/Exec Sql
C+ DELETE FROM APINV_PF WHERE PoNbr = :PoNbr
C/End-Exec

C                    If        SQLState <> '00000'
C                    ExSR      SQLErrSub
C                    EndIf
:
:
C     SQLErrSub      BegSR
C                    Eval      Error = (SqlState<> '00000'
C                                       And SqlState <> '02000')
C                    Eval      ErrMsg = ErrMsg + SQLState
C                    EndSR
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-48. Handling SQL errors using SQLSTATE OD477.0

Notes:

Many shops are writing their programs to be compliant with SQL standards. This example shows the use of SQLSTATE rather than SQLCODE. In addition to being the standard across SQL-based platforms, SQLSTATE offers the advantage of being able to identify a class of error.

For example, SQLSTATE class code ‘01’ means that a warning was issued. You could then write the code to handle this class of error in a specific subroutine.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — Next, we discuss using an error subroutine versus an error procedure.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Subroutine or CallP error procedure

IBM i

- Subroutine
  - Good start
  - Can use across programs
  - May not be as generic as separate procedure
  - May not truly handle (percolate?) errors
- Error procedure
  - More generic
  - Can pass program name where error occurred
  - Can pass label (SELECT01, DELETE02) to point to statement in error
  - Can register as ILE condition handler (handle, percolate as necessary)

```
Change ExSR SQLErrSub
CallP SQLErrPrc(SQLStt : 'SELECT02' : PGMNam : ErrorMsg : : )
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-49. Subroutine or CallP error procedure OD477.0

Notes:

As we stated earlier, you will probably get to a point in your error handling code that a subroutine is no longer performing the job that you want. At this point, consider a separate error handling procedure that can be called from any embedded SQL program.

The error handling procedure offers you the option of registering the procedure as an ILE condition handler. This allows you to handle an error in your code or to percolate an error up the call stack. You can avoid invoking the default error handler.

Further, the error procedure means you can pass parameters that provide more information about the error. You could identify the name of the program that experienced the error. You could label the statement that caused the error; this could assist you in debugging the error.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Encourage the students to discuss the options available when handling errors.

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Using indicator variables

IBM i

- SELECT
  - Required if null capable variables may be returned
  - Returns code to indicate whether or not null value returned and more!
    - Yes = -1
    - No = 0
    - Data mapping error = -2
    - String trunc = length of original

```
select workdept, empno, lastname
       into :workdept:workind, :empno,
      :lastname
       from employee
       order by workdept, empno
```

- SET (in UPDATE or INSERT)
- Assign value to indicator variable in program
  - If program var = null, assign -1 to indicator
  - If program var not null, assign 0 to indicator
- Define as two-byte integer

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-50. Using indicator variables OD477.0

Notes:

An indicator variable is a two-byte integer. You can define stand alone fields on the D-spec or you can also specify an array composed of integer variables.

On retrieval (using SELECT or FETCH), an indicator variable is used to show if its associated host variable has been assigned a null value.

On assignment to a column (using SET), a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are required if null-capable columns present. Otherwise, they are optional. They can, however, be used to obtain useful information for non null-capable columns:

- No problem, Indicator = 0
- Column contains null value, Indicator = -1
- Test for data mapping error, Indicator = -2

  For example:

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

  - Divide by zero in expression
  - Invalid Date conversion (Specifying a month with a value of 13)
  - Numeric overflow
  - Substring out of range
- String truncation, Indicator = length of original string(+N)

  For example:
  - Original length of character string that was truncated
  - Seconds portion of time when it is truncated on assignment to a Host Variable

Also note that a column of the result set can be tested for a null value:

- In the SQL statement by specifying **IS NULL**
- In program (Host Indicator = -1)

The SQL statement SET OPTION EXTIND = *YES or the compile parameter `OPTION(*EXTIND)` can be used to specify that extended indicator variables are enabled and non-updatable columns are allowed in the implicit or explicit UPDATE clause of a select-statement.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exercise

IBM i

Lab: *Change native to SQL I/O*



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-51. Exercise OD477.0

Notes:

Perform the machine exercise *Changing native to SQL I/O*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — We have completed this topic and next discuss how to use a cursor to process rows from a multiple row result set.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.4. Using a cursor for multiple row result set

Instructor topic introduction

In this topic, we show the student how to use both a serial cursor and a scrollable cursor. Dynamic scrollable cursors are also mentioned.

What students will learn — How to use cursors to process a result set of more than one row. Both serial and scrollable cursors will be discussed.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Figure 9-52. Topic 4: Using a cursor for multiple row result set OD477.0

Notes:

So far we've explored single row processing with SQL. Let's explore multiple row processing with its different techniques.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

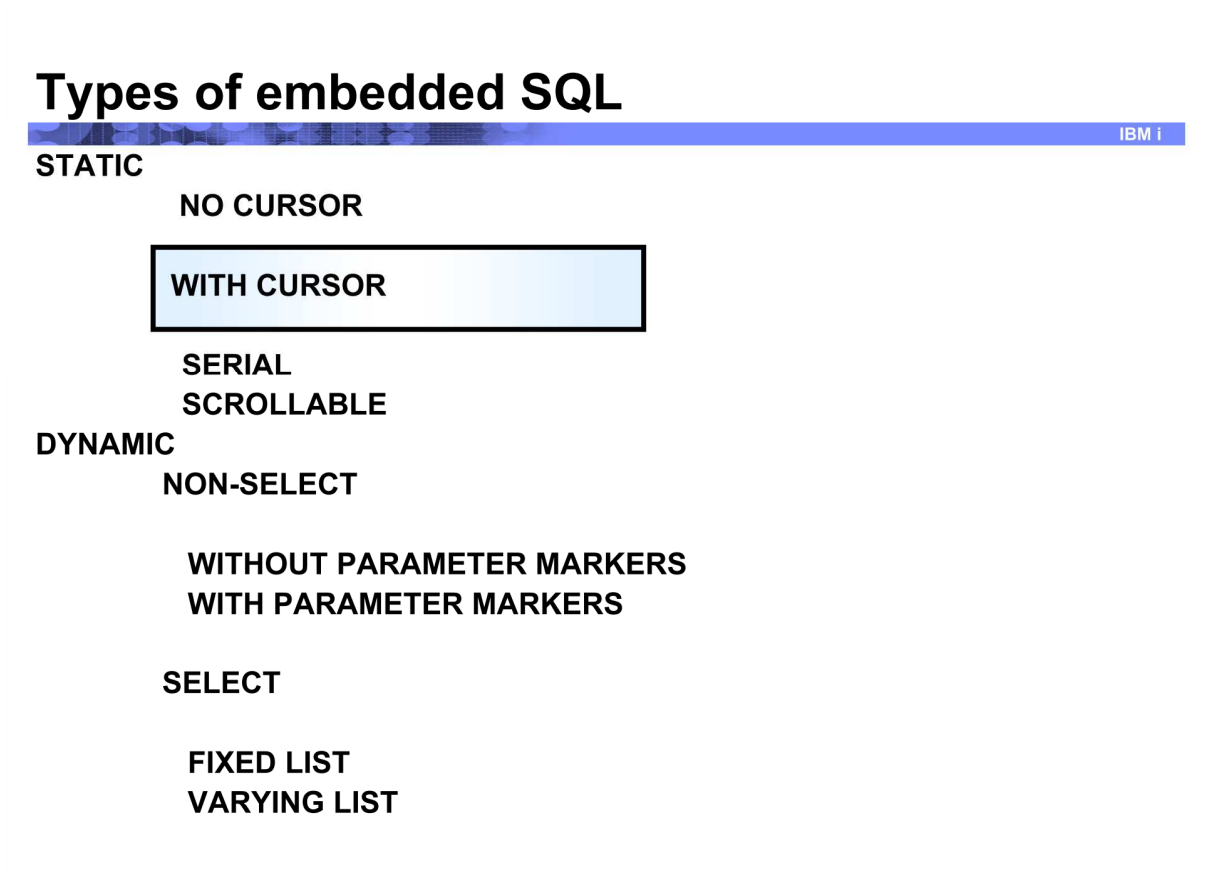


Figure 9-53. Types of embedded SQL OD477.0

Notes:

When you expect a SELECT to return more than one row, a cursor is required. In this topic, you will learn how to process a result set using a cursor and how to use a cursor to limit the scope of an UPDATE or DELETE statement.

There are two ways of declaring a cursor:

Serial The result set is processed in arrival sequence and can be processed only once per open of the cursor.

Scrollable The result set can be processed as many times as required and rows can be accessed in any sequence you want.

 © Copyright IBM Corp. 2010, 2013

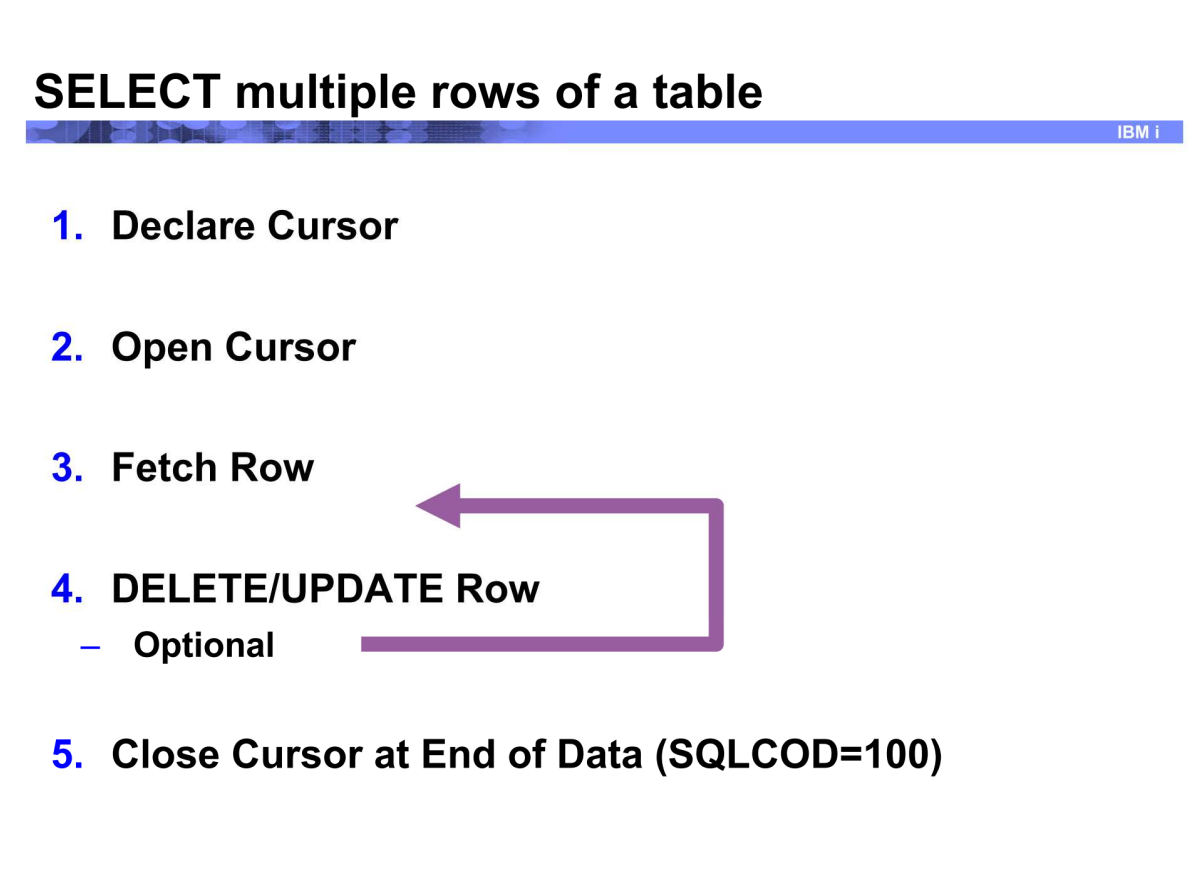
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — So far, we have been using SELECT statements that return a single row. Now, let us look at the situation where a SELECT returns more than one row.

Details — Whenever you expect a SELECT to return more than one row in its result set, you must use a cursor. This topic shows you how to set up a cursor and how to code your logic to use a cursor.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-54. SELECT multiple rows of a table OD477.0

Notes:

If your SELECT statement may return multiple rows of a table, a cursor for the selected rows **must** be defined.

The cursor is opened when the rows are to be selected and processed. The FETCH operation reads a row. That row could be updated, deleted, printed, etcetera. Unless SCROLL is specified, one row is read per open of the cursor and FETCH behaves like a sequential read operation.

In the situation described in this visual, we are using a **serial cursor**, that is, one that is defined without the SCROLL keyword.

For a serial cursor, each row of the result table can be fetched only once per OPEN of the cursor. When the cursor is opened, it is positioned before the first row in the result table. When a FETCH is issued, the cursor is moved to the next row in the result table. That row is then the current row. If host variables are specified (with the INTO clause on the FETCH statement), SQL moves the contents of the current row into your program's host variables.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

This sequence is repeated each time a FETCH statement is issued until the end-of-data (SQLCODE = 100) is reached. When you reach the end-of-data, close the cursor. You cannot access any rows in the result table after you reach the end-of-data. To use the cursor again, you must first close the cursor and then re-issue the OPEN statement. You can never back up.

We discuss SCROLL later in this unit.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — Next we review the steps of using a serial cursor and show an example.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a serial cursor (1 of 5)

IBM i

```
FEmbed04D  CF   E             Workstn Sfile(Data:Rrn)

D Indicators      DS                  Based(IndPtr)
D   Exit                         1N   Overlay(Indicators:03)
D   SflEnd                       1N   Overlay(Indicators:40)
D   SflDspCtl                    1N   Overlay(Indicators:85)
D   SflDsp                       1N   Overlay(Indicators:95)

D Rrn             S              5P 0 Inz(1)
D IndPtr          S               *
D ErrorMsg        S             50A   Inz('An error has occurred +
D                                           - SQLCODE = ') Varying

 /free

        IndPtr = %Addr(*In);
        Write  FncKey;
        SflEnd = *ON;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-55. Using a serial cursor (1 of 5) OD477.0

Notes:

The program on this and the following four visuals shows all the steps to use a cursor that were just described in the previous visual. This program loads and displays a subfile with rows of purchase order data. The following steps are followed in the logic shown in the five visuals:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This is a complete program to use to show how a serial cursor can be used. It is a long visual and it may be better to use the actual code in source member EMBED04RF. You can use the *PGM object to run the program and demonstrate its function.

Each of the steps is highlighted. Focus on the steps that are followed in the program and do not emphasize program logic unless you have questions about the logic.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a serial cursor (2 of 5)

IBM i

```
Exec SQL
   Whenever SQLERROR goto ErrorLbl;

Exec SQL
  1 Declare APInvCsr cursor for
    Select PONbr, VndNbr, APInvnbr, POTotAmt, APNetPaid,
        APDueDate, APDatePaid
    From APinv_PF
    Order by PONbr;

Exec SQL
  2 Open ApInvCsr;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-56. Using a serial cursor (2 of 5) OD477.0

Notes:

1 — Declare the cursor for the result set and SELECT the rows:

```
Exec SQL
   Declare APInvCsr cursor for
      Select PONbr, VndNbr, APInvNbr, PoTotAmt, APNetPaid,
         APDueDate, APDatePaid
      From APInv_PF
      Order by PONbr;
```

Your result set contains the rows returned by the SELECT and the cursor, APINVCSR, has been declared. Notice that the cursor is a variable within SQL only and is not used within the RPG code.

2 — Open the cursor for the result set:

```
Exec SQL
   Open ApInvCsr;
```

The cursor is opened in order to process the result set. Because we declared it as a serial cursor, we can only process the set sequentially and once per open.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a serial cursor (3 of 5)

IBM i

```
Exec SQL
3a Fetch Next from APInvCsr
     Into :PONbr, :VndNbr, :APInvNbr, :PoTotAmt,
          :APNetPaid, :APDueDate, :APDatePaid;

 Dow SqlCod <> 100;
     Write  Data;

Exec SQL
3b Fetch Next from APInvCsr
     Into :PONbr, :VndNbr, :APInvNbr, :PoTotAmt,
          :APNetPaid, :APDueDate, :APDatePaid;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-57. Using a serial cursor (3 of 5) OD477.0

Notes:

3a and 3b — Fetch rows from the result set and load the subfile with these rows until end of data (SQLCOD = 100):

```
Exec SQL
   Fetch Next from APInvCsr
      Into :PONbr, :VndNbr, :APInvNbr, :PoTotAmt, :APNetPaid,
      :APDueDate, :APDatePaid;
```

Since we have performed the SELECT we are now ready to fetch rows from the set. Using the cursor, we use fetch next to process the first and all subsequent rows until we reach end of data. When we have SQLCOD = 100, we drop out of the DoW. If we encounter any SQL errors (SQLCOD < 0) in the SELECT, FETCH, or other SQL statements, we immediately GOTO a label in the RPG program and post a message.

```
Exec SQL
   Whenever SQLError Goto ErrorLbl;
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a serial cursor (4 of 5)

IBM i

```
Rrn = Rrn + 1;
Enddo;

Select;

  When Rrn > 1;
    SflDspCtl = *ON;
    SflDsp = *ON;

  Dow NOT Exit;
    Exfmt Control;
  Enddo;

  When Rrn <= 1;
    SflDspCtl = *ON;
    SflDsp = *OFF;
    Write Msg;
    Exfmt Control;

Endsl;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-58. Using a serial cursor (4 of 5) OD477.0

Notes:

Page four of our code shows the program logic that handles display of the subfile.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a serial cursor (5 of 5)

IBM i

```
 Exec SQL
    Close APInvCsr;
4
     *InLR = *ON;
     Return;

 /end-free

C     ErrorLbl      Tag
C                   Eval      ErrorMsg = ErrorMsg +
C                                 %TrimL(%EditC(SqlCod:'1'))
C     ErrorMsg      Dsply     '*REQUESTER'

 /free

  Exec Sql
    Close APInvCsr;
5

    *InLR = *ON;
    Return;
/end-free
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-59. Using a serial cursor (5 of 5) OD477.0

Notes:

4 and 5 — Close the cursor at the end of data or if any SQL error occurred:

```
Exec SQL
   Close APInvCsr;
```


Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

UPDATE / DELETE current row

IBM i

```
C/Exec Sql
C+ UPDATE APINV_PF SET PONBR = :PoNbr, VNDNBR = :VndNbr,
C+ APINVNBR = :ApInvNbr, APCHECK# = :ApCheck#
C+ WHERE current of apinvcsr
C/End-Exec

C/Exec SQL
C+  Delete from apinv_pf
C+         where current of APINVCSR
C/End-Exec
```

- WHERE limits update/delete
- Only current row updated or deleted
- Cursor must be repositioned by another FETCH NEXT

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-60. UPDATE / DELETE current row OD477.0

Notes:

Remember that if you do not specify a WHERE, the DELETE or UPDATE operation applies to all rows. If WHERE is specified, the UPDATE or DELETE applies to only those rows identified by the WHERE.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

FOR UPDATE OF clause

IBM i

```
C/Exec Sql
C+ DECLARE APINVCSR CURSOR FOR
C+   SELECT PONBR, VNDNBR, APINVNBR, POTOTAMT, APNETPAID,
C+          APDUEDATE, APDATEPAID
C+     FROM APINV_PF
C+     FOR UPDATE OF APDUEDATE
C+     ORDER BY PONBR
C/End-Exec
```

- Restrict columns that can be UPDATEd.
- Without the FOR UPDATE OF clause, all columns could be updated.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-61. FOR UPDATE OF clause OD477.0

Notes:

FOR UPDATE OF identifies the columns that can be updated in the UPDATE that is executed next in sequence. Each column name must identify a column of the table or view identified in the first FROM clause of the fullselect. If the UPDATE clause is specified without column names, all columns of the table or view that can be updated are included. These columns must be identified in the first FROM clause of the select.

The FOR UPDATE OF clause is required if an ORDER BY is specified in the SELECT that declares the cursor. If not present, the cursor is read-only.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — FOR UPDATE OF limits the scope of any updates of the table. It is a form of security and data integrity within SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using SCROLLable cursors

IBM i

1. DECLARE apinvcsr SCROLL CURSOR for SELECT ....
2. OPEN CURSOR apinvcsr
3. FETCH (NEXT, PRIOR, LAST....)
4. DELETE/UPDATE row
5. CLOSE CURSOR
   - You decide when processing complete



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-62. Using SCROLLable cursors OD477.0

Notes:

The following points are some desirable features that you will get when you use a **SCROLL**able versus a nonscrollable cursor:

- **Programmer controls cursor**, where it points, and what is processed using various forms of FETCH noted below.
- You can **process result set as many times as you want**. You are not restricted to the previous limit of once through the set.
- You **do not have to close the cursor** until you have completed all processing of the set.
- To simply declare a cursor as SCROLLable means that the cursor is READ ONLY. You can declare the cursor to be **DYNAMIC SCROLL** and then **you can update** the value of the cursor.

When SCROLL is used, FETCH can do more than a sequential read. While a sequential read is still the default, **FETCH can position the cursor and/or read a row**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

For a scrollable cursor, the rows of the result table can be fetched many times. The cursor is moved through the result table based on the position option specified on the FETCH statement. When the cursor is opened, it is positioned before the first row in the result table. When a FETCH is issued, the cursor is positioned to the row in the result table that is specified by the position option. That row is then the current row. If host variables are specified, using the INTO clause on the FETCH statement, SQL moves contents of the current row into the program's corresponding host variables.

With a scrollable cursor, you control how the set is processed in the FETCH statement:

NEXT	Positions the cursor on the next row. This is the default if no position is specified.
PRIOR	Positions the cursor on the previous row.
FIRST	Positions the cursor on the first row (like SETLL + READ).
LAST	Positions the cursor on the last row.
BEFORE	Positions the cursor before the first row.
AFTER	Positions the cursor after the last row.
CURRENT	Does not change the cursor position.
RELATIVE n	Evaluates a host variable or integer **n** in relationship to the cursor's current position. For example, if n is -1, the cursor is positioned on the previous row of the result table. If n is +3, the cursor is positioned three rows after the current row.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — Emphasize that there is really no difference in the logic flow. The only difference is additional flexibility in processing the result set. You can loop within the set in whatever way you want.

Emphasize that FETCH not only positions the cursor but also returns a row. Compare this to RPG where you must position the cursor in one statement followed by the I/O operation (READ).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Blocked fetch

IBM i

```
D APIvnDS         DS                    OCCURS(25)
D  PONumber                        7P 0
D  VendorNumber                    7P 0
D  APInvoice                       8
D  APDateInv                        D
D  POTotal                         7P 2
D  APDiscAmt                       5P 2
D  APNetAmtPaid                    7P 2
D  APStatusCode                    1
D  APDateInvPaid                    D
D  APCheckNumber                   7P 0
D  APPaymentDue                     D

C/Exec Sql
C+ FETCH apinvcsr FOR 25 ROWS INTO :apinvDS

C/End-Exec
```

- Fetch 25 rows in one operation
- Feedback in SQLCA:
  - SQLERRD(3) subfield contains # rows fetched
  - SQLERRD(5) subfield contains 100 if last row of set fetched
- Excellent performer!

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-63. Blocked fetch OD477.0

Notes:

FETCH can read multiple rows. A host structure must be set up to hold the received rows. The SQL communications area, SQLCA, holds information about the FETCH.

SQLERRD is a six-element array and is part of the SQLCA.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

CLOSE the cursor

IBM i

- More to consider than:

```
C/Exec Sql
C+ CLOSE APINVCSR
C/End-Exec
```

- Explicit CLOSE:
  - Perform in program
  - If you call program over and over, best to close cursor explicitly
- Implicit Close:
  - CLOSQLCSR on CRTSQLRPGI command
    - *ENDMOD: When the module terminates
    - *ENDACTGRP: When activation group terminates

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-64. CLOSE the cursor OD477.0

Notes:

There are two ways to close the SQL cursor:

1. Explicitly: In your program using a CLOSE SQL statement:
2. Implicitly: Let the system close it.

We recommend that the cursor be closed explicitly whenever possible and that it be closed as soon as possible in your code. “As soon as possible” would be immediately after you have reached the end of a result set (SQLCOD = 100) for a serial cursor or once you have completed processing the result set with a scrollable cursor.

SQL cursors are explicitly closed when the user issues the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

With an implicit close, SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released.

The values for the parameters of CRTSQLxxxI are:

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

***ENDACTGRP**	SQL cursors are closed and SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.
***ENDMOD**	SQL cursors are closed and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the first SQL program on the call stack ends.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exercise

IBM i

Lab: *Using a cursor*



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-65. Exercise OD477.0

Notes:

Perform the machine exercise *Using a cursor*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.5. Dynamic embedded SQL

Instructor topic introduction

What students will learn — How to use Dynamic SQL and the differences involved in coding it versus static SQL.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

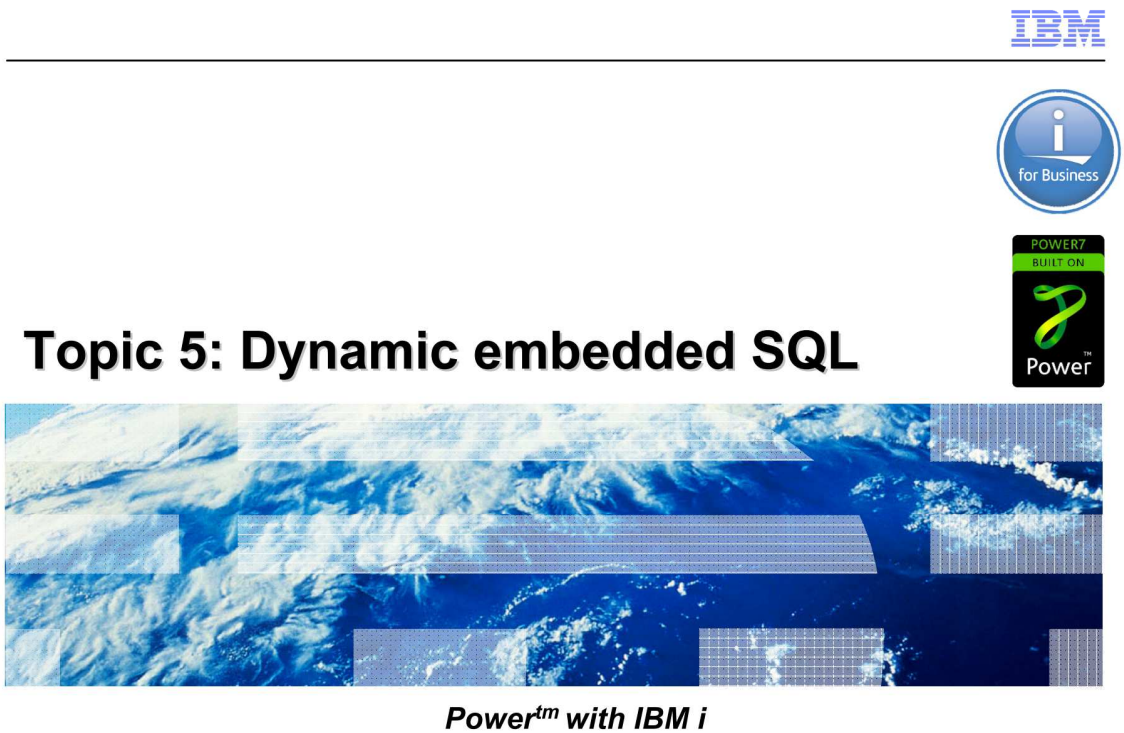


Figure 9-66. Topic 5: Dynamic embedded SQL OD477.0

Notes:

There are different techniques used when the embedded SQL is dynamic.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

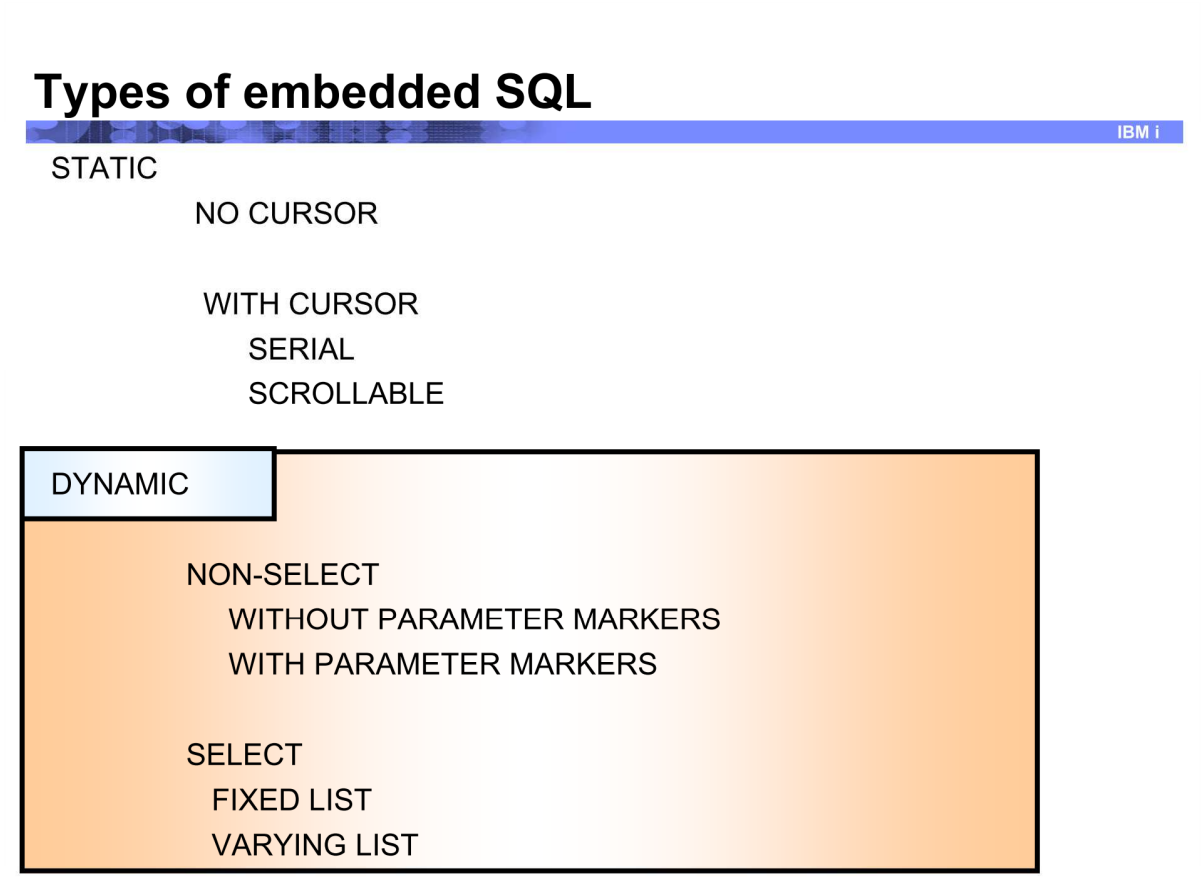
Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-67. Types of embedded SQL OD477.0

Notes:

Now we will cover Dynamic SQL. You will learn how to use parameter markers in Dynamic SQL and how to prepare SQL statements during execution rather than at compile time.

The most complex subject that we will discuss is the use of the **Dynamic Varying List SELECT**.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Use this visual as a checkpoint of where we are, what we have done, and what we will do in this topic.

Details — The students will realize that in many cases it is better to code static SQL statements and to provide the flexibility you need by coding static SQL in RPG IV subprocedures rather than use Dynamic SQL. This is especially true when you are considering the option of coding **Dynamic Varying List Select**.

Mention that you can mix static and dynamic SQL in a program. The precompiler handles the static statements, as we have already seen.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Types of dynamic SQL

IBM i

- Embedded in an application program
- Call level interface (ODBC)
- 5250 STRSQL interface
- 5250 RUNSQLSTM
- System i Navigator RUN SQL scripts
- ....and others

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-68. Types of dynamic SQL OD477.0

Notes:

Using dynamic SQL within your programs is only one form of dynamic SQL. There are other forms. This topic focuses only on the embedded form of dynamic SQL.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Static SQL versus dynamic SQL

IBM i

- Static SQL Statement
  - Statement form and tables known when program is coded
  - Complete SQL statement coded in the program
  - Host variables correspond to SQL variables
  - Statement prepared by precompiler
  - Access plan determined by precompiler
- Dynamic SQL Statement
  - Statement form and/or tables unknown when coding program
  - Precompiler cannot prepare statement
  - Statement created as a variable in program
  - Prepared at execution
  - Access plan determined dynamically
- Can use Static AND Dynamic SQL in your program

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-69. Static SQL versus dynamic SQL OD477.0

Notes:

When you code static embedded SQL, the SQL statement (Select, Update), table and column names (in the “SELECT list” and the “WHERE” clause) are known and coded in the program. The precompiler can prepare these SQL statements.

Static SQL means that there will be and cannot be any changes made to such statements during program execution.

As part of the precompiler process, access paths will be created also.

Dynamic SQL must be used if the SQL statement type, or the table names, or the column names (and definitions) are unknown, or some combination of these factors is unknown.

If the unknown factors are finite and not too many, you could write a static SQL statement for every possibility. If there are too many combinations desired and static SQL code would be overly tedious, then dynamic SQL may be considered as an alternate means of meeting the desired result.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — We want to make the students understand that Dynamic SQL offers great flexibility. But, flexibility includes a price of code that is more complex and that usually takes more system resource to execute than static SQL. Dynamic SQL cannot be prepared by the precompiler. It must be prepared during execution.

Remind the class that it is true that, in most cases, a programmer can code the necessary options using static techniques rather than dynamic SQL. And, a program can include any combination of static and dynamic statements.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Dynamic SQL statements

IBM i

- Not prepared by preprocessor
- No host variables
  - Parameters marked by ?
- Must be prepared during execution of program
- Ready to execute following preparation
- Steps to follow:
  - Build SQL statement in character variable
  - Prepare the SQL statement using PREPARE SQL statement
  - Execute the prepared statement using EXECUTE SQL statement
  - Logic flow for cursor is the same as static
- May use EXECUTE IMMEDIATE for non-SELECT dynamic SQL

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-70. Dynamic SQL statements OD477.0

Notes:

Dynamic SQL is very different than static SQL. Dynamic statements are coded as character variables. These character strings can be defined as variable in length.

No host variables can be included in dynamic SQL statements. You accomplish the same result by using parameter markers that are denoted by question marks (**?**) wherever a parameter is to be passed to the statement.

Once your statement has been built in the character string, it can be prepared using the SQL PREPARE statement.

Following preparation, you specify the SQL EXECUTE statement. For non-SELECT statements, you can use the EXECUTE IMMEDIATE statement of SQL to prepare and execute the statement in the character string in one step.

SELECT statements must be handled differently because they pass data to the program.

Be aware that preparing SQL statements during execution means that you pay the performance price at this time rather than during the compilation process.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

For your information, the `CRTSQLRPGI` command has a parameter that permits you to delay the preparation of all statements until execution time. If you change the default of the **DLYPRP** parameter to ***YES**, then all statements, including static SQL statements, are prepared during execution.

In general, delaying preparation is not a good idea.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Dynamic non-SELECT: No parameter markers

IBM i

```
D Uplift          S              20A
D Department      S               3A
 ** 1
D SqlString       S             100A


C                   Exsr      Prompt
 ** 3
C                   Dow       Department <> 'END'
 ** 4
C                   Eval      SqlString =
C                               'UPDATE EMPLOYEE ' +
C                                 'SET   SALARY = ' + Uplift +
C                                 'WHERE WORKDEPT = '''+
C                                           Department + ''''
 ** 5
C/Exec Sql
C+ EXECUTE IMMEDIATE :SqlString
C/End-Exec

C                   Exsr      Prompt
C                   Enddo
C                   Eval      *InLR = *ON

C     Prompt        Begsr
 ** 2
C     'Uplift Expr' Dsply     '*REQUESTER'  Uplift
C     'Department?' Dsply     '*REQUESTER'  Department
C                   Endsr
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-71. Dynamic non-SELECT: No parameter markers OD477.0

Notes:

The UPDATE statement is constructed in a variable **SQLString** using an EVAL. The “variable” components of the statement, *Uplift* and *Department* are prompted variables that are included in the final SQL statement. *Department* can be any valid department number, using a value of *END* to terminate the program. *Uplift* can be any valid expression.

Notice that we can **PREPARE** and **EXECUTE** the SQL statement using an **EXECUTE IMMEDIATE**. We can only do this because there are no parameter markers used in constructing the SQL statement variable.

1 — A character variable is defined that will be used to construct the SQL statement

2 — We prompt for two values, one for a *Department* and second for the value of an expression named *Uplift* to the character string.

3 — A *Department*value = 'END' is used to end the program.

4 — The SQL statement is constructed in the character variable, *SQLString*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

5 — The SQL statement is prepared. This includes validating syntax, the table's existence, the columns in the statement. The access plan is determined. Because there are no parameter markers, the EXECUTE IMMEDIATE statement may be used.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — To demonstrate the use of EXECUTE IMMEDIATE.

Details — This program, **DYNOHV**, can be demonstrated. It will prompt you for two parameters as you can see (based on the EMPLOYEE table). Use a value of upper case **END** when prompted for department to end the program. Note that your SALARY column for all rows in a valid department will be updated. You may want to restore your data in your tables after you are done.

Some areas of discussion:

- What would the UPDATE statement variable look like when DEPARTMENT = 'A00' and UPLIFT = 'SALARY*1.05'? Have the students answer this question and construct the statement on the board.

```
UPDATE EMPLOYEE SET SALARY = SALARY*1.05
       WHERE WORKDEPT = 'A00'
```

- Optionally, take the discussion further by asking what would happen if either variable resulted in an invalid SQL construct. For example, what if we set *Uplift* = SALRY*1.05?

  In the joblog, a message would be issued as the Database Manager was attempting to PREPARE the statement to be executed:

```
Message ID . . . . . :   SQL0206       Severity . . . . . :   30
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   03/09/01      Time sent  . . . . . :   09:39

Message . . . . :   Column SALRY not in specified tables.
Cause . . . . . :   SALRY is not a column of table EMPLOYEE in library
                    *LIBL
```

Transition statement — Next, we modify this example slightly and use parameter markers rather than variables in the construction of the SQL statement.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Dynamic non-SELECT: Parameter markers (1 of 2)

IBM i

```
D Uplift          S              20A
D Department      S               3A
D SqlString       S             100A


C     'Uplift Expr' Dsply     '*REQUESTER'  Uplift
 ** 1
C                   Eval      SqlString =
C                               'UPDATE EMPLOYEE ' +
C                                 'SET   SALARY = ' + Uplift +
C                                 'WHERE WORKDEPT = ?'
 ** 2
C/Exec Sql
C+ PREPARE SqlStmt FROM :SqlString
C/End-Exec
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-72. Dynamic non-SELECT: Parameter markers (1 of 2) OD477.0

Notes:

The example above uses variables, that is, includes a parameter marker (**?**). If it did not use parameter markers, we could have used **EXECUTE IMMEDIATE** to prepare and execute the SQL statements as in the previous visual.

But, if we are coding dynamic SQL, we are doing it to provide function and flexibility beyond static SQL.

The logic has been rearranged slightly to demonstrate the use of parameter markers.

Note the steps that are followed in the above example:

1 — We have modified the program in the previous example and replaced the *department* variable with a parameter marker (**?**).

2 — The SQL statement is prepared. This includes validating syntax, the table's existence, the columns in the statement. The access plan is determined. The value of the *department* variable is unknown at this point so we cannot perform an EXECUTE IMMEDIATE.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — This example is designed to show the basic components of dynamic SQL, parameter markers, PREPARE and EXECUTE statements, and the character string variable used to construct the statement.

Details — Explain how dynamic SQL works using the example. The code is DYNHV.

It is important to stress that I could have easily used static SQL to do the same. But, what if the user wanted even more flexibility, such as:

- UPDATE based on last name?
- DELETE based on manager?
- A different SQL statement such as an INSERT?

You do make trade-offs when you use dynamic SQL. Perhaps the most important consideration is that you pay a performance price when the dynamic statement is prepared. On the other hand, if you prefer the convenience of coding dynamic SQL, perhaps this cost is worth paying.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Dynamic non-SELECT: Parameter markers (2 of 2)

IBM i

```
C                   Exsr      Prompt
C                   Dow       Department <> 'END'
 **3
C/Exec Sql
C+ EXECUTE SqlStmt USING :Department
C/End-Exec

C                   Exsr      Prompt
C                   Enddo
C                   Eval      *InLR = *ON

C     Prompt        Begsr
C     'Department?' Dsply     '*REQUESTER'  Department
C                   Endsr
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-73. Dynamic non-SELECT: Parameter markers (2 of 2) OD477.0

Notes:

3 — After prompting for the value Department, we EXECUTE the prepared SQL statement, and provide the value of the *department* variable in the USING clause.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Dynamic fixed-list SELECT with parameter markers (1 of 2)

IBM i

```
D Department      S              3A
D Sequence        S             10A
D SqlString       S            100A
D Msg             S             50A

D EmployeeDS    E DS                  ExtName(EMPLOYEE)

C     'Sequence?'   Dsply     '*REQUESTER'  Sequence
C                   Eval      SqlString =
C                               'SELECT EMPNO, LASTNAME, WORKDEPT ' +
C                                 'FROM EMPLOYEE ' +
C                                 'WHERE WORKDEPT = ?' +
C                                 'ORDER BY ' + Sequence

C/Exec Sql
C+ PREPARE SqlStmt FROM :SqlString
C/End-Exec
C/Exec Sql
C+ DECLARE SqlCsr CURSOR FOR SqlStmt
C/End-Exec


C                   Exsr      Prompt
C                   Dow       Department <> 'END'

C/Exec Sql
C+ OPEN SqlCsr USING :Department
C/End-Exec
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-74. Dynamic fixed-list SELECT with parameter markers (1 of 2) OD477.0

Notes:

Dynamic fixed list SELECT is coded in the same fashion as a non-SELECT statement. This example uses a SELECT with the table and columns known. This form of statement is known as **Dynamic Fixed-List SELECT**.

In this example, the user provides a column name that determines the sequence in which the set is ordered. The *Sequence* variable is an RPG IV variable (not a host variable, nor parameter). However, *department* is passed to SQL using a parameter marker. Notice where *:department* is highlighted in the **USING** clause.

Notice that we are using a cursor to process the result set. Whether you expect to be returned a single row or not, you **must** use a cursor to process a dynamic fixed list result set.

Also, remember how we coded for a cursor in static SQL? The process is the same as Dynamic Fixed-List SELECT. The SELECT statement that is contained in the character string is part of the DECLARE and thus the SELECT is executed at that point. A separate EXECUTE would be redundant and would result in an error message.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Notice the USING clause in the OPEN CURSOR statement. At this point, you specify the host variable, **:Department**.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This program is DYNFIXSEL. You may demonstrate it. It is similar in design to the previous two programs, but because it is a SELECT statement that will produce a result set, **a cursor must be used to process the set**.

Review this example thoroughly, focusing the students' attention on the DECLARE statement and the USING clause in the OPEN CURSOR. Some will be confused because we just said an EXECUTE follows a PREPARE. This is not true in this form of SELECT.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Dynamic fixed-list SELECT with parameter markers (2 of 2)

IBM i

```
C                   Dow       SqlCod <> 100
C                   Eval      Msg = Empno + ',' + %TrimR(LastName) +
C                                     ',' + Workdept
C     Msg           Dsply     '*REQUESTER'

C/Exec Sql
C+ FETCH SqlCsr INTO :Empno, :LastName, :Workdept
C/End-Exec
C                   Enddo

C/Exec SQL
C+ CLOSE SqlCsr
C/End-Exec

C                   Exsr      Prompt
C                   Enddo
C                   Eval      *InLR = *ON

C     Prompt        Begsr
C     'Department?' Dsply     '*REQUESTER'  Department
C                   Endsr
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-75. Dynamic fixed-list SELECT with parameter markers (2 of 2) OD477.0

Notes:

The FETCH and CLOSE SQL statements complete the remainder of the program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Dynamic varying list SELECT

IBM i

- Constructs of SELECT unknown
- Table(s) unknown
- Columns unknown
  - Data type unknown
  - Length / scale unknown



- Solution:
  - Dynamic Varying List SELECT!
  - SQL Descriptor Area (SQLDA)

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-76. Dynamic varying list SELECT OD477.0

Notes:

We hope that you think that so far Dynamic SQL is fairly easy to understand. But, when you do not know anything about the data being passed to you from the database, you cannot code the SELECT in its entirety. You have insufficient information that does not allow you to code a static or Fixed-List SELECT.

When you do not know what columns are to be selected and when you do not know the characteristics of the data, you must use a more complex form of SQL known as **Dynamic Varying List SELECT**. This form of SELECT uses the SQL Descriptor Area to communicate information about the data to your program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Do not present Dynamic Varying List SELECT as a possible solution when columns, and so forth, are unknown. It is difficult to implement and because of this, it may be simpler and quicker to write extra code to use Dynamic Fixed List SELECT. In several visuals, we include some sample code with some honest comments by an experienced RPG IV programmer regarding some issues he experienced when attempting to implement Dynamic Varying List SELECT.

Transition statement — Before we discuss how you code this type of SELECT, we discuss how the SQLDA is used.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Dynamic varying list SELECT: The process

IBM i

1. Define character variable that will hold SQL statement
2. INCLUDE or define SQLDA data structure
3. Construct SQL statement FROM character variable
4. PREPARE the SQL statement FROM character variable
5. DESCRIBE the SQL statement INTO the SQLDA
6. Set up your (host) variables to receive rows and provide addresses to SQLDA
7. FETCH rows using the SQLDA

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-77. Dynamic varying list SELECT: The process OD477.0

Notes:

At this point, your coding will become far more complex.

Let us assume that the user keys in the complete SELECT statement. Assuming it is correct, we first have to prepare the statement.

Since you do not know which columns in a table will be accessed, you cannot relate any host variables to SQL variables in the SELECT.

What you must do instead is interact with the SQLDA. The process involves a new SQL statement, **DESCRIBE**. The DESCRIBE statement asks the database manager to fill the SQLDA with information about the variables received after your statement has been prepared.

Once you have this information, you can work with the SQLDA and your program variables. In order to process the result set, you use a FETCH that uses the information in the SQLDA to process each row.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This function of SQL can be extremely complex. I do not intend to cover all the detail but instead will provide a sufficient overview so that the students can understand the level of complexity. Unfortunately, the iSeries SQL Programmers' Guide is weak in its explanation. Some of the books referenced in the appendix at the back of this notebook are much better.

I recommend that, as a general rule, this type of SQL be carefully weighed against less complex alternatives.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using the SQLDA (1 of 2)

IBM i

- DESCRIBE into SQLDA for each column in SELECT:
  - Column name
  - Data type
  - Length
  - Scale if numeric
- SQLDA
  - Contains an area for each column
  - You must estimate how many columns you expect to pass to SQLDA
- Map columns to program variables to process each row
  - Allocate host variables
  - Place addresses of host variables into SQLDA
- FETCH row into program using SQLDA

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-78. Using the SQLDA (1 of 2) OD477.0

Notes:

You use the SQLDA to interface with the database because you know nothing about the data that is being passed to your program.

The characteristics of the data and addresses (pointers) are held in the SQLDA. Pointers are supported by RPG IV (not in RPG III).

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — The use of address pointers is mandatory when manipulating the SQLDA.

Additional information —

Transition statement — Next, we review the process showing the interface with the SQLDA.

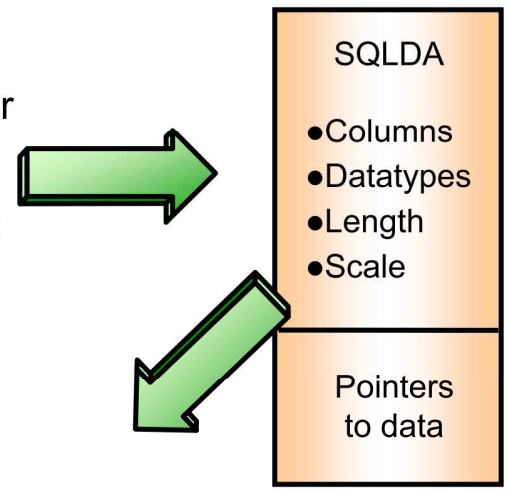
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using the SQLDA (2 of 2)

IBM i

1. Construct SELECT statement in character string (:chrstr) based on runtime specifications
2. PREPARE sqlstm FROM :chrstr
3. DESCRIBE sqlstm INTO :sqlda
4. DECLARE csr FOR sqlstm
5. OPEN csr
6. FETCH csr USING DESCRIPTOR :sqlda



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-79. Using the SQLDA (2 of 2) OD477.0

Notes:

The contents of the SQLDA are updated by the Database Manager. You use the SQLDA in your program to interface with the database when your program knows nothing about the data that is being passed to your program or the construct of the SELECT statement is built in the program in a character string.

In order to use the SQLDA data structure, you must INCLUDE SQLDA in your RPG IV program. A sample program is included in the notes for your reference. The SQLDA can be copied from the member accessed by the precompiler. However, you will notice that the version used by the precompiler does not leverage the features of the D-spec. It would be better to build your own SQLDA source member and include it in your programs.

The characteristics of the data and address pointers are stored in the SQLDA.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

This visual shows you the process of interfacing with the SQLDA using the DESCRIBE and the FETCH statements:

DESCRIBE into SQLDA for each column in SELECT:

- Column name
- Data type
- Length
- Scale if numeric

- SQLDA
  - Contains an area for each column
  - You must estimate how many columns you expect to pass to SQLDA
- Map columns to program variables to process each row
  - Allocate host variables
  - Place addresses of host variables into SQLDA
- FETCH row into program using SQLDA

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — To show the students how to interact with the SQLDA.

Details — Emphasize that SQLDA is not automatically copied into the program. You can use INCLUDE SQLDA to obtain the data structure depicted in following visual.

However, this layout has many limitations. Dynamic Varying List SELECT is not easily implemented in RPG IV and may need to be programmed using another HLL such as C++.

Transition statement — Next, we look at the contents of the SQLDA.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQLDA data structure

IBM i

```
D SQL_NUM         C                   CONST(4)

D*      SQL Descriptor area
D SQLDA           DS
D  SQLDAID                 1      8A
D  SQLDABC                 9     12B 0
D  SQLN                   13     14B 0
D  SQLD                   15     16B 0
D  SQL_VAR                       80A   DIM(SQL_NUM)
D                         17     18B 0
D                         19     20B 0
D                         21     32A
D                         33     48*
D                         49     64*
D                         65     66B 0
D                         67     96A
D*
D SQLVAR          DS
D  SQLTYPE                 1      2B 0
D  SQLLEN                  3      4B 0
D  SQLRES                  5     16A
D  SQLDATA                17     32*
D  SQLIND                 33     48*
D  SQLNAMELEN             49     50B 0
D  SQLNAME                51     80A
D*  End of SQLDA
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-80. SQLDA data structure OD477.0

Notes:

This visual shows you the fields included in the SQLDA. Some of the main ones to notice are:

- **SQLD**

  SQLD is a 2-byte integer that specifies the actual number of occurrences of SQLVAR; that is, the number of host variables described by the SQLDA. This field is set by SQL on a DESCRIBE or PREPARE statement. In other statements, this field must be set prior to use to a value greater than or equal to 0 and less than or equal to SQLN.

- **SQLN**

  SQLN is a 2-byte integer that specifies the total number of occurrences of SQLVAR. It must be set prior to use by any SQL statement to a value greater than or equal to 0.

- **SQL_VAR**

  You must allocate a size to this multi-occurrence data structure. There is one data structure for each column processed by the SELECT. Notice that the number of elements is a variable, SQL_NUM. This variable is a standalone constant that you must

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

define on the D-spec. If your estimate of the number of columns to be processed is low, you will receive a message. You can handle this message and change the value of SQL_NUM and then process the SELECT again.

- **SQLTYPE**

  SQLTYPE is a 2-byte integer that specifies the data type of the host variable as shown in the table below. Odd values for SQLTYPE show that the host variable has an associated indicator variable addressed by SQLIND.

- **SQLLEN**

  SQLLEN is a 2-byte integer variable that specifies the length attributes of the host variables.

There is a detailed table in the Programmers' Guide that describes the values for SQLLEN and SQLTYPE.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details —

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL387U02** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL387U02** and click play.

c. Ask the students to indicate using green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL387U02** and ask the students to call out the answers to the questions on the screen.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: All host variables used in an SQL statements must be declared.

2. True or False: To embed SQL into an RPG IV program, the member type must be SQLRPGLE.

3. True or False: The SQL Communications area is required in any program that uses embedded SQL.

4. True or False: Embedded SQL in RPG IV programs can process multiple records in a single I/O operation.

5. True or False: A serial cursor allows for more than sequential reading of rows in a table.

6. True or False: Dynamic non-SELECT embedded statements may or may not use parameter markers.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-81. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: All host variables used in an SQL statements must be declared.
   The answer is True.

2. True or False: To embed SQL into an RPG IV program, the member type must be SQLRPGLE.
   The answer is True.

3. True or False: The SQL Communications Area is required in any program that uses embedded SQL.
   The answer is False.

4. True or False: Embedded SQL in RPG IV programs cannot process multiple records in a single I/O operation.
   The answer is False.

5. True or False: A serial cursor allows for more than sequential reading of rows in a table.
   The answer is False.

6. True or False: Dynamic non-SELECT embedded statements may or may not use parameter markers.
   The answer is True.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Exercise

IBM i

Lab: *Dynamic embedded SQL*



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-82. Exercise OD477.0

Notes:

Perform the machine exercise *Dynamic embedded SQL*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — To let the students try a mix of static and dynamic SQL.

Details — This can be a lengthy lab, so allow plenty of time to perform it. This lab consolidates the two programs that the students have already written into a single program.

A suggested way to approach the exercise would be to leave the existing logic of the two programs as-is. Test the result, and once it works, the students can add some dynamic SQL or rewrite some of the logic to make the program better.

When helping the students with problem solving, encourage them to use the debugger and the job log. Very often, they will find the cause of a problem in QSQLMSG. The SQL messages are composed of “SQL0” concatenated with SQLCOD. For example, SQL0501 is SQLCOD -501.

If the student codes dynamic SQL and uses a character string, they may need to check the value of the character string using the debugger to confirm that they are preparing a valid SQL statement. Also, they might need to modify the length of the string and use VARYING.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- Describe how to perform I/O operations in an RPG IV program using embedded SQL
- Describe how to modify an existing RPG IV program to use embedded SQL
- Code embedded SQL statements in an RPG IV program
- List the differences between coding native and SQL-based I/O
- Create an executable program from an embedded SQL RPG IV program
- Use SQL error handling facilities in an RPG IV program
- Explain the purpose of the SQL Communications Area
- Explain the purpose of the SQL Descriptor Area
- List the differences between Static SQL and Dynamic SQL

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 9-83. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit 10. An overview of stored procedures

Estimated time

01:45

What this unit is about

This unit describes Stored Procedures and how they can be used.

What you should be able to do

After completing this unit, you should be able to:

- List two types of stored procedures
- Explain the environments in which stored procedures can be used
- Create and run a simple stored procedure

How you will check your progress

Accountability:

- Checkpoint questions
- Exercises

References

DB2 for IBM i SQL Programming Concepts

SG24-6503 *Stored Procedures and Triggers on DB2 Universal Database for iSeries*

SG24-6393 *Modernizing IBM EServer iSeries Application Data Access - A Roadmap Cornerstone*

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- List two types of stored procedures
- Explain the environments in which stored procedures can be used
- Create and run a simple stored procedure

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — Introduce stored procedures.

Details — In this unit, we discuss what stored procedures are and how they can be used. We explain that stored procedures are well suited to Client/Server environments supported by ODBC and JDBC. Since stored procedures can be called from any SQL interface, they are great tools for integrating multiple platforms, delivering IBM i database to Windows and they are an attractive means of bringing legacy applications to the web.

In the first topic we tell the students that there are two types of SPs. In order to show the students how simply they can be implemented, we show them an example that calls a batch RPG IV program. The second topic looks more closely at External SPs and adds passing a parameter and returning a result set.

In the third topic, we describe SQL SPs.

At the end of the unit, the students performs a lab where they have an existing RPG IV program that has been written using embedded SQL. The program is designed to return as a result set. The students write the SP, create the procedure, and then test their SP using the Run SQL Scripts function of iSeries Navigator.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

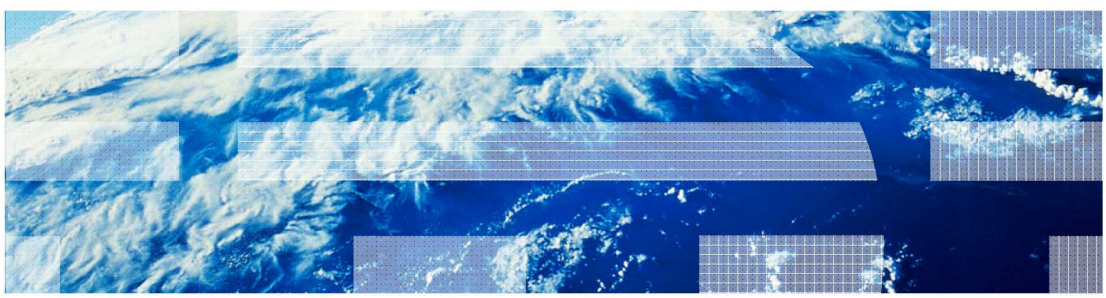
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

10.1.Introducing stored procedures

Instructor topic introduction

If you intend to demonstrate stored procedures, you should open windows for Adobe for the PDF presentation, a 5250 emulation session and a Run SQL Scripts session. Or you could use System i Navigator's Run SQL Scripts.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



Topic 1: Introducing stored procedures

Power^{tm} with IBM i

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

8.1

Figure 10-2. Topic 1: Introducing stored procedures OD477.0

Notes:

This topic explains stored procedures and the environment in which they may be used.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Why CALL?

IBM i

- CALL another HLL program
- SQL program = SQL stored procedure
- Can call SQL stored procedure from SQL or SQL-based applications:
  - 5250 STRSQL
  - Java/VisualAge
  - CLI/ODBC
  - Operations Navigator RUN SQL SCRIPTS
- Must CREATE stored procedure

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-3. Why CALL? OD477.0

Notes:

When you want to run another program from within your program on the IBM i, you issue a call to the external program (CALL, CALLP, or CALLB are the calls that you use regularly).

When you want to run a program from a command line, you simply call that program or call a CLP.

Suppose you want to run an RPG IV (or other HLL) program from an SQL-based interface. You can issue a CALL from SQL, but, in order to do so, the program to be called must be “registered” in the SQL catalog as a **stored procedure**.

Once a program has been registered as a stored procedure, you can call it from a PC client using ODBC (Call Level Interface) or Java (JDBC). Since System i Navigator uses these interfaces, it can be used to call Stored Procedures.

The Run SQL Scripts facility of System i Navigator is an excellent interface to use to develop Stored Procedures and to test them.

You will have an opportunity in a simple machine exercise to experiment with this tool.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Emphasize that stored procedures enable you to encapsulate RPG IV programs as SQL objects that can be called from SQL applications. In addition to what is mentioned in the notes, you can also use the CALL in embedded SQL in an HLL program. In this topic, we define and introduce External as well as SQL stored procedures. In the other two topics, we spend more time discussing how to write and call stored procedures. A simple lab follows the end of the next topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

What is a stored procedure?

IBM i

- An SQL program that contains:
  - HLL program with or without embedded SQL
  - SQL language
- Stored procedure created by CREATE PROCEDURE SQL statement
- All procedures registered in SYSPROCS in QSYS2
- Can pass parameters
- Can return parameters, result set
- Widely used in client/server applications to reduce traffic

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-4. What is a stored procedure? OD477.0

Notes:

A **stored procedure** is an SQL object on the System i that is based upon an IBM i HLL language program or one or more SQL statements. Hence, there are two types of stored procedures:

- Those based on an HLL program are called **external procedures**.
- Those written using SQL statements are known as **SQL stored procedures**.

These procedures are well suited for a client server environment. They are executed based on a call (SQL-based) from the client to the stored procedure on the server. Result set support allows ODBC and SQL Call Level Interface (CLI) applications to open multiple cursors for each invocation of a single stored procedure. Also, result set support allows ODBC and SQL CLI applications to use native I/O to perform database access on the System i (as we did with embedded SQL) and to return a result set to the calling CLI application.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

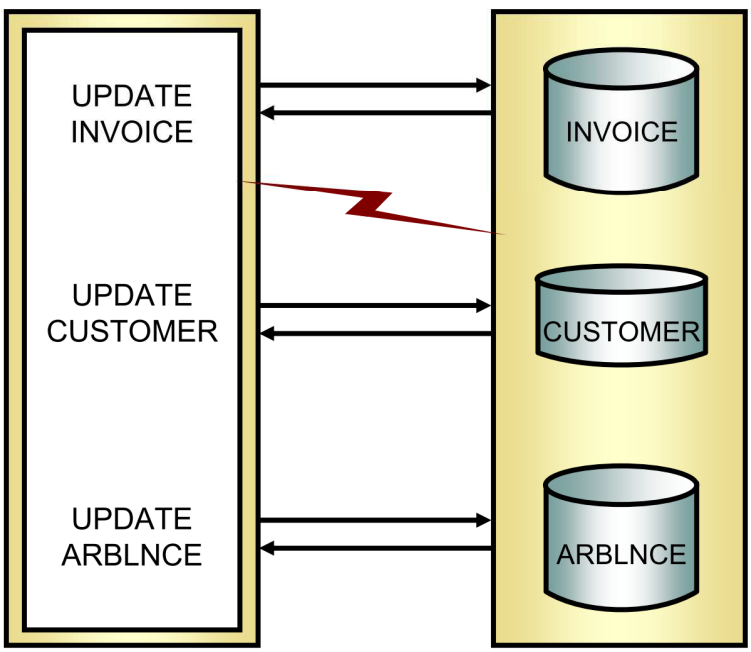
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Call from client: Without stored procedure

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-5. Call from client: Without stored procedure OD477.0

Notes:

Let's look at a typical example where stored procedures can be effective. A company runs its business on a server located at the headquarters and on client systems located at every branch office. A user at a branch office is working with an invoicing application, which has to update three tables on the server:

- The invoice table, INVOICE.
- The customer table, CUSTOMER.
- The account receivable balance table, ARBLNCE.

When an invoice record is paid, it is flagged "cleared". After that, the corresponding CUSTOMER record is updated by deducting the invoice amount from the current accounts receivable total amount. Finally, the accounts receivable balance record is updated as well.

The visual shows a distributed application for the invoice clearance process that was implemented without resorting to stored procedures. The client system has to access the server database several times for every update event, sending and receiving data across

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

communication lines for every request. In addition, all the application logic is implemented at the client site.

Note that each request for a database transaction is sent across the network to the IBM i.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — These transactions could be entered interactively as individual SQL statements or as individual SQL statement calls from a client application. While these applications are feasible, they do generate more line traffic and may become a performance bottleneck over time.

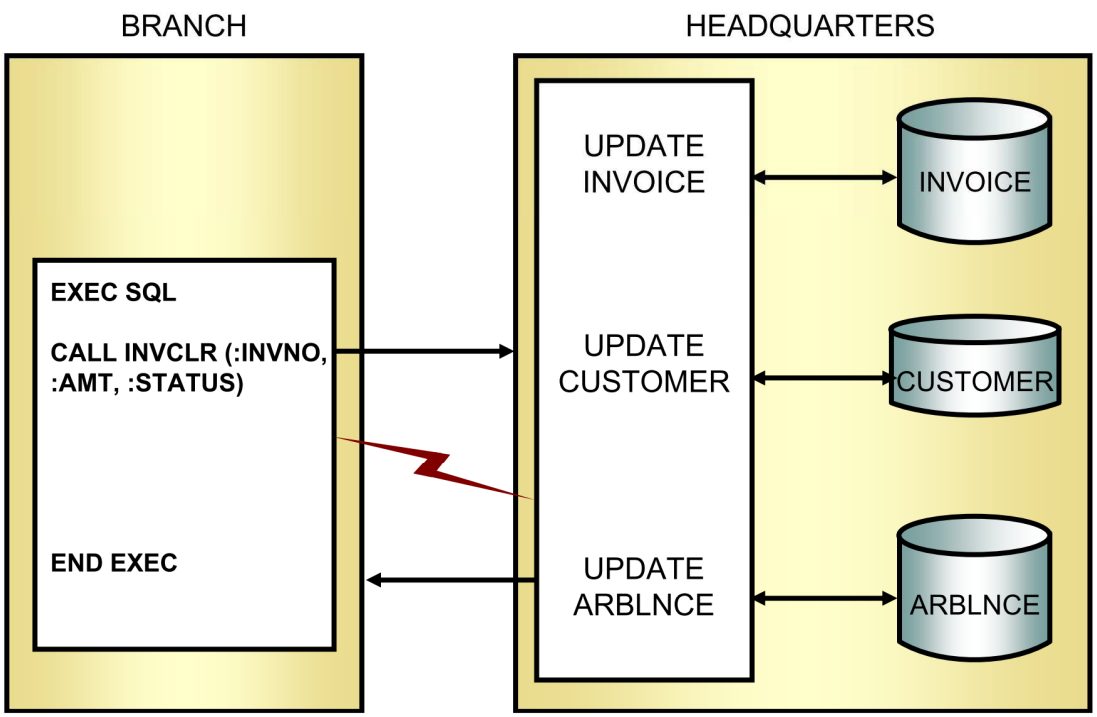
Transition statement — Stored procedures, which send one request to process SQL transactions, can relieve performance concerns and actually can be easier to implement than coding individual client called SQL transactions.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Call from client: With stored procedure

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-6. Call from client: With stored procedure OD477.0

Notes:

When a stored procedure is called, one request is sent to the iSeries. All I/O is handled by a Stored Procedure which can return parameters and result sets to the client.

This way is not only efficient, it will perform better because less load is placed on the network.

The same application functions can be carried out by calling a single stored procedure that runs at the server site. The communications dialog is greatly reduced, and the network resources are better balanced by splitting the application logic.

Modularity in application development is also encouraged by using stored procedures. This makes application maintenance easier and improves code re-usability.

When you think about stored procedures, they are similar in concept to other options such as:

- SBMRMTCMD
- DRDA coding

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Trigger programming
- Remote *DTAQ
- MQ-Series

There are pluses and minuses to each option.

If we focus at this time solely on stored procedures, we can say that:

- They are easily implemented.
- They can use existing applications (external procedures).
- When implementing external stored procedures, you leverage your existing HLL skills.
- When implementing SQL stored procedures, you also use existing skills (SQL). No new skills are required.

Other alternatives, such as trigger programming, require that new skills be learned.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

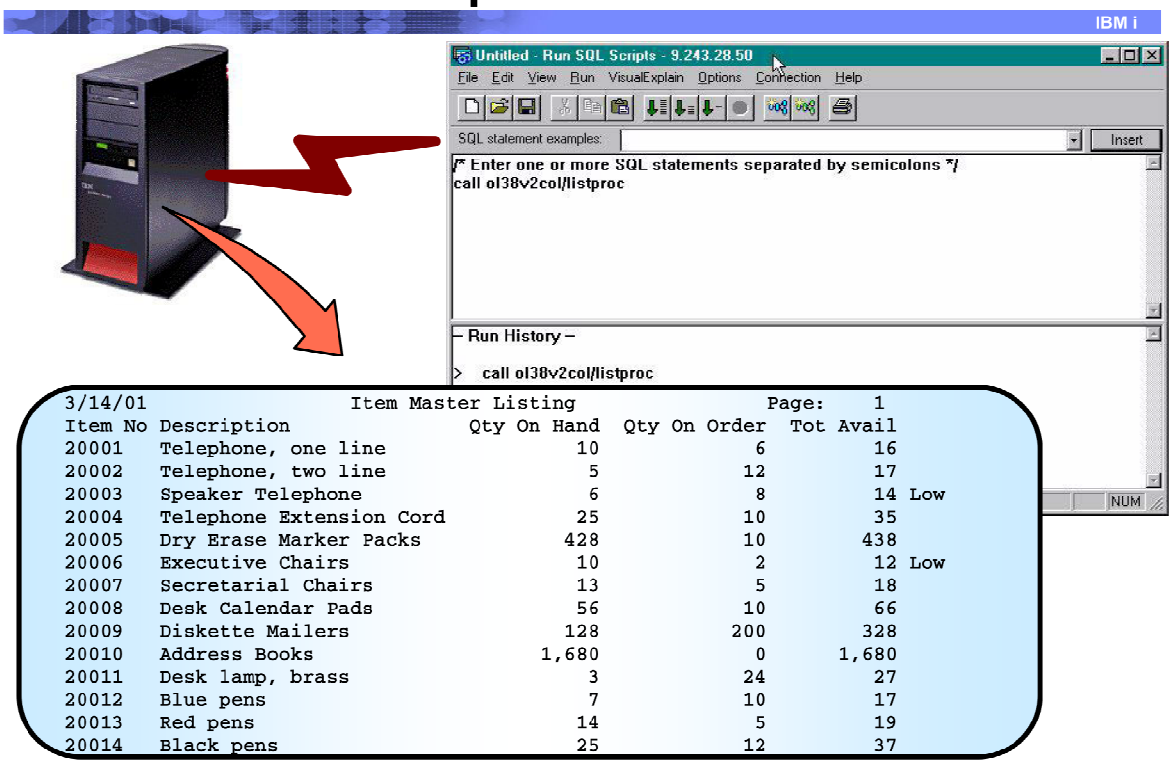
Instructor notes:

Details — No details are included on alternative methods that you might use. I leave that to you to discuss based on your background. We want to focus attention on stored procedures and emphasize how easy they are to implement.

Transition statement — Next, let's look at a very basic example. In the example, we call a stored procedure that executes an existing batch program that produces a report.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Call external stored procedure



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-7. Call external stored procedure OD477.0

Notes:

Suppose you wanted to run a program that produced a listing. In this visual and the next, we show you how to call the program from SQL and then show you how to create the SQL stored procedure that calls the RPG IV program.

This example shows you that we can call an external procedure named LISTPROC from System i Navigator. The basic steps that are followed are:

- Call stored procedure using Run SQL Scripts.
- Client PC passes request to IBM i Database Manager.
- Database Manager checks SQL System Catalog for existence of stored procedure.
- Stored Procedure calls RPG IV listing program.
- RPG IV program executes and places listing in OUTQ.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — You can demonstrate this if you wish. The name of the Stored Procedure is LISTPROC in the course library. LISTPROC calls an RPG program, PORLISTS, in the same library. No parameters are required. We are simply calling the program from SQL! If you run the program, show the students the report file in your OUTQ.

Discuss other ways we could call the program besides using a CL command. One way would be to use Run SQL Scripts, which enables running System i jobs using the iSeries Navigator interface. Another way would be to CALL the Stored Procedure from the 5250 STRSQL interface. A third way would be to embed the CALL in a SQLRPGLE program.

Transition statement — Now, let's look at how the stored procedure was created that called the report program.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Create external stored procedure

IBM i

```
create procedure ol38nncol/Listproc
       language rpgle
       external name ol38nncol/porlists
```

- Encapsulates *PGM so that it can be called from SQL
- Registers procedure in SYSPROCS in QSYS2
- Declares parameters to be passed to procedure on CALL
- HLL can be native I/O or embedded SQL I/O or both
- Can return optional parameters, result sets
- Easy way to develop SQL applications

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-8. Create external stored procedure OD477.0

Notes:

The CREATE SQL statement is used to create a procedure. If the stored procedure is external, as in this case, you must provide the name of the *PGM and the library where it is located.

In this visual, you can see the statement that was used to create the procedure, LISTPROC. There are more complex procedures that we can create but this one allows us to call and execute a listing program. Once the procedure is created, an entry is added to the SYSPROCS SQL catalog file in QSYS2.

While not used in this example, parameters may be passed. They must be declared in the CREATE PROCEDURE statement.

Also, you have written RPG IV code using embedded SQL that produces single or multiple row result sets. Stored procedures have the capability of handling these result sets and returning them to the PC Client.

Notice that the name of the stored procedure can be different than the actual Program Name (short). A good example is provided in QSYS. This stored procedure is shipped with

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

the system and can be used at your site to create and load sample tables (corporate database):

```
CALL QSYS.CREATE_SQL_SAMPLE
```

Refer to Appendix page A-1 of the *SQL Programming Concepts* manual for further details.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — In the next visual, we cover the various parameter types and then we discuss passing parameters in the topic that follows.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Parameter styles

IBM i

- Defines what parameters are passed
- GENERAL
  - Traditional parameter passing
- GENERAL WITH NULLS
  - Traditional parameters plus indicators for NULLS
- SQL
  - Like GENERAL WITH NULLS plus procedure information
  - Also includes SQLSTATE

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-9. Parameter styles OD477.0

Notes:

The PARAMETER STYLE clause of the CREATE PROCEDURE statement defines the parameter passing convention for an external procedure. The style that you specify will determine how the stored procedure is to be called.

- **GENERAL**: A general call to the procedure is performed and the list of parameters on the CALL matches those defined in the external procedure.

  This parameter style most closely matches the conventional style of parameter passing.
- **GENERAL WITH NULLS**: A general call to the procedure is performed as above plus an additional argument is passed for indicator variables.
- **SQL**: All applicable parameters are passed. The parameters are defined to be in the following order:

  1) The first `N` parameters are the input parameters that match those specified on the CREATE PROCEDURE statement.
  2) Another `N` parameters for indicator variables for the input parameters.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

3) A CHAR(5) parameter for SQLSTATE.
4) The fully qualified procedure name.
5) The specific name.
6) The message text.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL stored procedures

IBM i

```
create procedure
       ol38nncol/addbonus
       language sql
       update employee set bonus = (bonus + 100)
```

- Use SQL language
- Stored procedure portability
- SQL programmers can be productive on the IBM i
- IBM i implementation follows SQL standards
- Invoked the same way as external procedures

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-10. SQL stored procedures OD477.0

Notes:

IBM i programmers have the option of writing an entire program in SQL. SQL programmers who are new to the IBM i can produce DB2 for IBM i stored procedures without having to learn these system-specific operations.

The SQL procedural language should not be considered a new language for the IBM i.

Stored procedures can be leveraged best in network computing environments where processing is divided between the client and the server. They provide an easy way to place related database operations into a single object to reduce network traffic.

For example, consider an application that requires that you perform a select, update and insert operation in sequence. The client could request each of these operations separately from the server and cause three separate round trips across the network to the server.

It is much more efficient to bundle these three related database operations into a stored procedure and then have the client call the stored procedure. Using a stored procedure requires only a single trip to and from the server.

The stored procedure will ensure that the transactions are executed in the desired order.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Transition statement — The next topic discusses external stored procedures in more detail.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

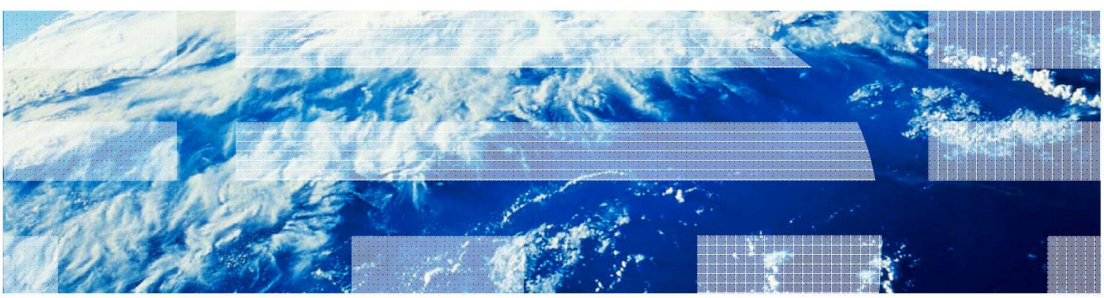
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

10.2.External stored procedures

Instructor topic introduction

In this topic we discuss external stored procedures that are based on HLL programs. We explore the example that was used in the first topic. Also, we include a second example that accepts a parameter and returns a result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-11. Topic 2: External stored procedures OD477.0

Notes:

Let's explore external stored procedures in more detail.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Developing an external stored procedure

IBM i

IBM i server:
- Write an HLL program
  - May be an existing program with native I/O
  - May mix embedded and native I/O
  - May declare parameters
    - prototyped or PARM
  - May return one or more result sets
- CREATE PROCEDURE
  - References the HLL program
  - Adds entry in SYSPROCS (SYSPARMS if parameters)

Client PC:
- Write client application or use Run SQL Scripts
  - ODBC or JDBC - include SQL CALL to server procedure

Testing facilities:
- System i Navigator
  - Result sets supported
  - Can receive parameters
  - Superior testing tool
- Limited test with 5250 STRSQL
  - No parameters returned
  - No result sets
  - Can test report like or programs that update/delete rows

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-12. Developing an external stored procedure OD477.0

Notes:

Once you have coded your server-based RPG IV programs, most of the work is done; all you have to do is to register the program by creating an external Stored Procedure that can be called from a PC client SQL application. Some modifications to the RPG IV source may be required but, as you will see, they are relatively simple.

Once the HLL program has been coded, we must create the stored procedure. This is done by the SQL CREATE statement. This statement can be executed from the interactive SQL environment or from a source member using RUNSQLSTM.

On the client side, usually a PC, you would code a program that issues an SQL CALL to the stored procedure on the IBM i.

Testing of a stored procedure can be performed on a limited basis using 5250's STRSQL; a far superior tool is System i Navigator, which is included in IBM i Access. All testing scenarios can be performed using RUN SQL SCRIPTS before you test by running the client-based ODBC or JDBC application, for example.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

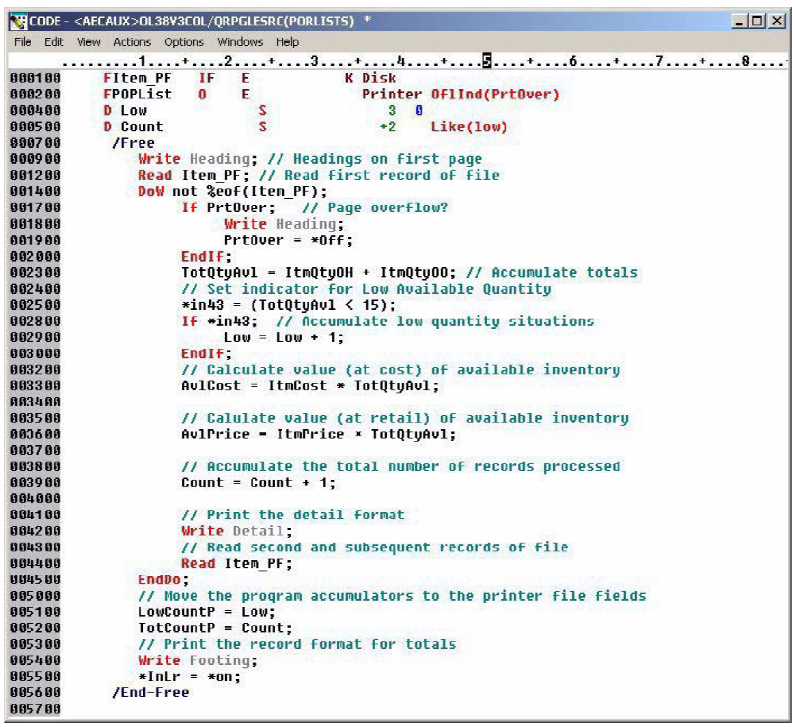
Purpose — Review what was introduced in the first topic and add further explanation.

Details — Review how an external stored procedure is created and accessed from a client application.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stored procedure to call report program (1 of 3)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-13. Stored procedure to call report program (1 of 3) OD477.0

Notes:

This is the source for a very simple report program. It reads a file and prints a report, using a PRTF.

We want to be able to call it from SQL. Notice that:

1. There are no parameters.
2. All I/O is done by RPG IV. (We could use embedded SQL.)
3. There is no result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — We want the students to realize that you can call even a simple RPG program from SQL.

Details — This is the source of the program that we called in the previous topic.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stored procedure to call report program (2 of 3)

IBM i

```
create procedure ol38nncol/Listproc
       language rpgle
       external name ol38nncol/porlists
```

- No parameters declared
  - Program does not accept parameters
- No result sets returned
- CREATE statement is run in SQL (STRSQL or RUNSQLSTM)
- Entry added to SYSPROCS in QSYS2

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-14. Stored procedure to call report program (2 of 3) OD477.0

Notes:

In order to be able to run the RPG IV program from a client, we create an SQL stored procedure. This is the same one used in the previous topic. Notice that:

- The program expects no parameters and therefore none are declared in the CREATE PROCEDURE statement.
- No result set is returned by the program; only a report is produced. Our output will still be placed in our OUTQ even though we are calling this SP from SQL.
- The CREATE PROCEDURE statement may be run in 5250 STRSQL or System i Navigator Run SQL Scripts.
- Once the procedure is created successfully, an entry is added to the SQL system catalog SYSPROCS in QSYS2.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Additional information — We do not cover this in this class, but, you may decide to discuss the fact that creating a stored procedure is optional. For an External Stored Procedure, you do not have to CREATE (nor DECLARE procedure, which we have not covered at all) the procedure in SQL. If you are prepared to live within the limits of these defaults when you run the “procedure”, you may call the external program directly.

A static CALL statement from SQL without a corresponding CREATE PROCEDURE statement is processed with the following rules:

- All host variable arguments are treated as INOUT type parameters.
- The CALL type is GENERAL (no Indicator Variables are permitted for detecting null values).
- The program to call is determined based on the procedure name specified on the CALL, and, if necessary, the naming convention.
- The language of the program to call is determined based on information retrieved from the system about the program.

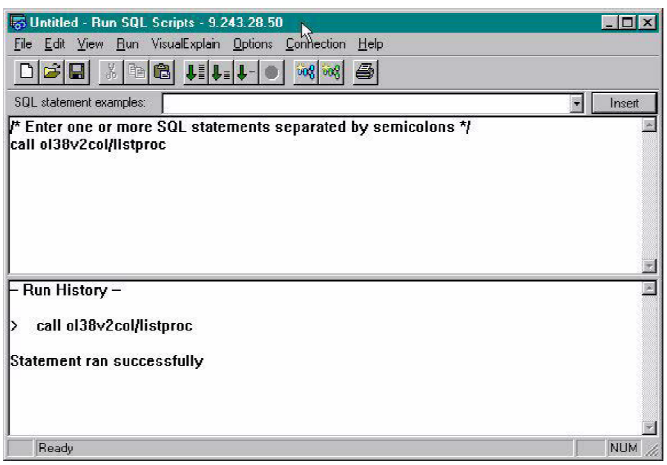
For additional information, please see of the *SQL Programming Concepts* document. Please see *Using Embedded CALL Statement where no procedure definition exists*. (For V5R2, see page 168).

Transition statement — Now, we look at an example that accepts a parameter and returns a single row result set.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stored procedure to call report program (3 of 3)

IBM i



- CALL executed from PC client
- No parameters passed
- No result set returned
- Message issued that "Statement ran successfully"
- Can view output of program in OUTQ

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-15. Stored procedure to call report program (3 of 3) OD477.0

Notes:

In this visual, we use the **Run SQL Scripts** option of System i Navigator to call the SP on the IBM i server.

Now that we have reviewed the process, let's add some more function and consider an example that includes a parameter and a result set.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

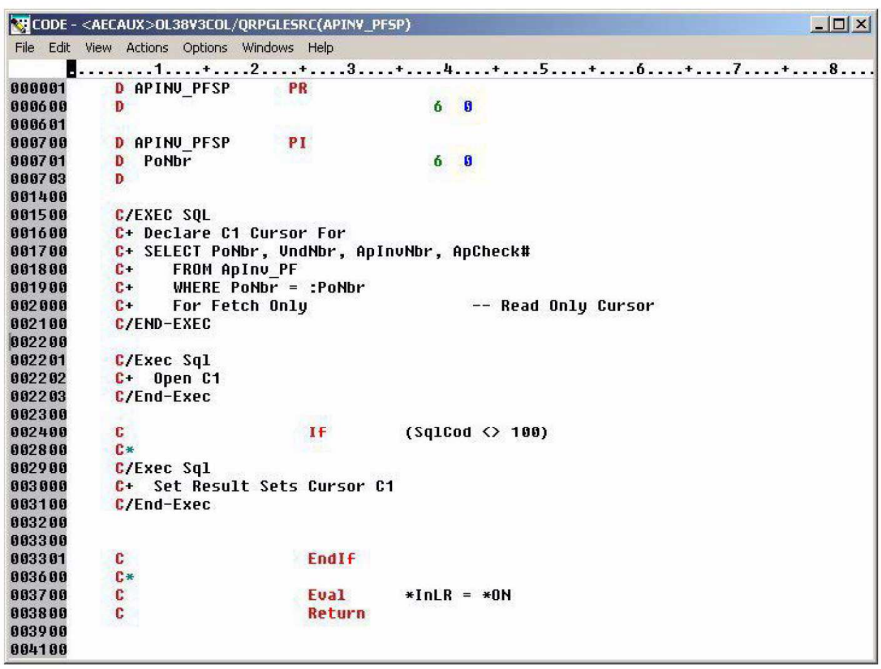
Purpose — This completes the overview of the process of creating and executing a stored procedure.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stored procedure with parameter (1 of 3)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-16. Stored procedure with parameter (1 of 3) OD477.0

Notes:

Let us now consider a case where we have an embedded SQL RPG IV program that has been designed to accept a single parameter as well as return a result set (which contains one row):

1 — This is a program that accepts a parameter, `PoNbr`.

2 — Because we are returning a result set to the calling client application, we use a cursor. We DECLARE the cursor and then open it.

3 — We have included very simple error checking. If SQLCOD = 100, we know that we have an empty result set and we will set on LR and return to the caller.

4 — If there is a result set, we must use the SET statement to specify that we want to return it and that the set uses the cursor C1.

5 — Notice that we **do not close the cursor** in the program. Because we are returning a result set, we will let the system handle closing the cursor. If you were to close the cursor in the program, no result set would be returned to the client.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The result set produced by the program must leave the cursor open in order for the Stored Procedure to be able to pass the set to the client. If the cursor is closed by the program, then there is no cursor that can be used by DB2 to process the result set and return it to the client.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — This process is straightforward. Notice that we do not close the cursor. The reason is that the cursor must be open for the result set to be passed and processed by the DB Manager. The DB Manager processes the result set using the open cursor and then presents it to the caller, which in our case is usually *Run SQL Scripts*.

When SQL runs a SELECT statement, the resulting rows comprise the result set. The cursor is used to maintain a position in the result set. SQL uses a cursor to work with the rows in the result table and make them available to the program.

Leaving the cursor open is a design mandate that is consistent across DB2 platforms.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Stored procedure with parameter (2 of 3)

IBM i

- Create module using PDM option 15 (CRTSQLRPGI/CRTRPGMOD)
  - Specify *NONE for commitment control if not used
- Create RPG program using PDM option 26 (CRTPGM)
- CRTPGM PGM(OL38V3COL/APINV_PFSP) ACTGRP(*CALLER)
  - If not *CALLER, will not return result set
- Create stored procedure:

```
create procedure
      ol38nncol/poproc (in ponbr decimal(6,0))
      result set 1
      language rpgle
      external name ol38nncol/apinv_pfsp
      general
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-17. Stored procedure with parameter (2 of 3) OD477.0

Notes:

You should follow these guidelines in order to successfully create your stored procedure:

- When you create the program, it will be an ILE program. First, you need to create an ILE module. If you are not using commitment control in your program, specify ***NONE** for the `COMMIT` parameter (default is *CHG).
- Create your ILE program (CRTPGM) and specify ***CALLER** for the `ACTGRP` parameter (default is *NEW).

  When you execute your Stored Procedure, you must be in the same activation group in order for the result set to be passed to the client PC.

  Activation group and cursor

  In order to be able to see the result set, both the Stored Procedure and the RPG IV program must run in the same activation group. The reason is that the cursor would be closed by the system when the activation group holding the RPG IV program ends. If the SP is in a different activation group, it would not see the cursor.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

If you get a message that your SP ran successfully, but see no result set, you have usually missed one of the above two items.

- When you create the stored procedure, you must declare the parameter(s). In this example, we have an input parameter. Notice that you must match the data type and scale of the RPG IV variable. Information about all parameters is maintained in the SQL catalog file, SYSPARMS, in QSYS2.

  Also, notice that we are using the `GENERAL` parameter style that is much like the traditional style of parameter passing.

- Since we are expecting a result set, we include information in the CREATE PROCEDURE statement to tell it how many sets to expect. The *Result Set* clause may be more than one, for example, if you expect three sets, code *Result Sets 3*.

If you decide to delete the procedure, you must use the SQL **drop procedure** statement.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Details — For an OPM program object, external stored procedures just point at the existing program objects whether they are written in RPG, C, or others.

If the external procedure is written in an ILE language, then a special flag is set in the program object, so that on a restore to another system, DB2 will automatically execute a CREATE PROCEDURE to register it as an external stored procedure.

SQL procedures must be called using the SQL CALL statement. When called, the SQL procedure runs in the activation group of the calling program. The procedure name is used as the name of the member in the source file and the name of the program object, if it is a valid system name. If the procedure name is not a valid system name, a unique name is generated. If a source file member with the same name already exists, the member is overlaid. If a module or a program with the same name already exists, the objects are not overlaid, and a unique name is generated. The unique names are generated according to the rules for generating system table names.

Important! We do not cover several things in class that may come up because you have a student with a DB2 or SQL background. I discovered some things while testing Stored Procedures:

1. Modifies SQL Data (default) or No SQL/Contains SQL/Reads SQL Data) is a clause of the Create Procedure statement. Please read the Create Procedure command of the SQL Ref. page 489: **Specifies which SQL statements, if any, may be executed in the procedure or any routine called from this procedure. See Appendix F, “Characteristics of SQL Statements” on page 845 for a detailed list of the SQL statements that can be executed under each data access indication**. The default is **Modifies SQL Data**. Notice that if you select the intuitive choice, Contains SQL, no result set will be returned from the RPG program!
2. Create Procedure creates an entry in the SYSPROCS table in QSYS. One of the columns documents where the external program can be found. However, when running SQL Scripts, you must have the library where the external RPG IV program is located in the library list (see the server tab of the JDBC setup). If the library list is incomplete, the RPG IV program will not be found. The bind to the RPG IV program occurs when you call the stored procedure.

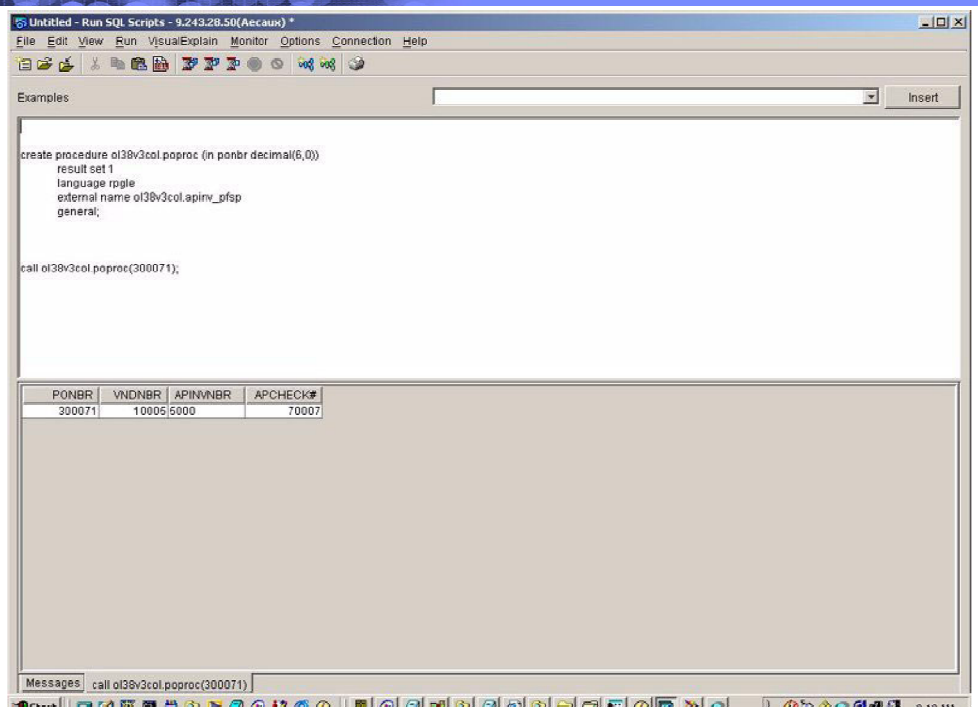
Additional information — ILE programs are more easily ported as SQL stored Procedures than OPM programs. With an OPM program, you would have to execute a CREATE PROCEDURE statement again to register the object.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stored procedure with parameter (3 of 3)

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-18. Stored procedure with parameter (3 of 3) OD477.0

Notes:

The stored procedure is now ready for the final step of the development process, testing.

We show Run SQL Scripts as the client tool to test our new application. To run the application re-enter the SQL statement:

Call library/poproc (300071) where **300071** is a valid purchase order number.

If you are not familiar with this tool, here is the step-by-step process that you would use to run this script:

1. Double-click the **System i Navigator** icon on your desktop.
2. Right-click or expand (**+**) the server name that you want to open.
3. Sign on to the server when prompted.
4. To expand, click the **Database** icon.
5. Under **Database**, right-click the system name.
6. From the menu, select **Run SQL Scripts**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

7. In the **Run SQL Scripts** menu, click **Connection**.
8. Click **JDBC Setup**.
9. Click the **Format** tab and make sure that the **naming convention** is set for your environment. In our example, we are using *SYS naming convention.
10. When you are finished, click **Apply** and **OK**.
11. Enter your command in the white space. We enter **Call library/poproc (300071)**. Note that when you enter a single statement, the semicolon (**;**) is not mandatory. You **must** enter a semicolon if you enter a series of SQL statements.
12. Use the **Run** menu option to execute one or more statements.

Additional information — If your statement executes successfully, you will receive a message, **Statement Ran Successfully**. Depending upon your settings, either a tab appears that you can select to display the results or a new window opens.

If you were to call a stored procedure from a client program, you would code an SQL CALL statement to invoke the stored procedure.

The parameter list that you code in the CALL must match the parameter list specified in the HLL program. Each `IN`, `OUT`, or `INOUT` parameter that you defined in the CREATE PROCEDURE statement must be specified as an SQL parameter or variable. The number of arguments specified must be the same as the number of parameters defined by your procedure.

You can use the interactive SQL interface or an SQL script in System i Navigator to test your stored procedure. If the procedure executes basic SQL statements that modify data, you can check to see whether an UPDATE, DELETE, or INSERT was performed correctly. However, if the procedure returns a result set, or passes parameters, you will not see it when you use STRSQL nor when you call the procedure from an SQL Script within System i Navigator.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out that you can pass NULL, a constant or a special SQL register as well as a variable as a parameter.

RUN SQL Scripts - Note that System i Navigator was ODBC-based until V4R5. It was modified in V5 to use JVMs. Now, RUN SQL SCRIPTS is JDBC based.

Transition statement — We have now discussed the basics of creating external stored procedures. Stored Procedures can also be coded in SQL. In the next topic, we talk about how you can do this. First, we perform a simple exercise.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exercise

IBM i

Lab: *Create a stored procedure*



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-19. Exercise OD477.0

Notes:

Perform the machine exercise *Create a stored procedure*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

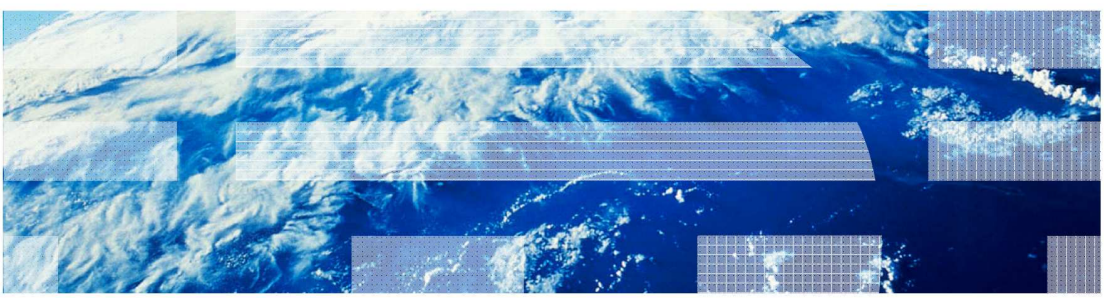
10.3.SQL stored procedures

Instructor topic introduction

In this topic, we discuss writing SQL stored procedures using the SQL language. We show you how to create and run an SQL SP. Since all SQL SPs are coded in the SQL language, we also cover a few of the extended aspects of the language.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-20. Topic 3: SQL stored procedures OD477.0

Notes:

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL stored procedures

IBM i

- Same concept as external procedure
- No existing *PGM object
- SQL code is written inline in the procedure

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-21. SQL stored procedures OD477.0

Notes:

SQL stored procedures have been available since V4R4 of OS/400. Rather than having to code an HLL program to access the database, all the code is written in SQL within a self-contained procedure.

The stored procedure is created and called in the same way as an external stored procedure.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

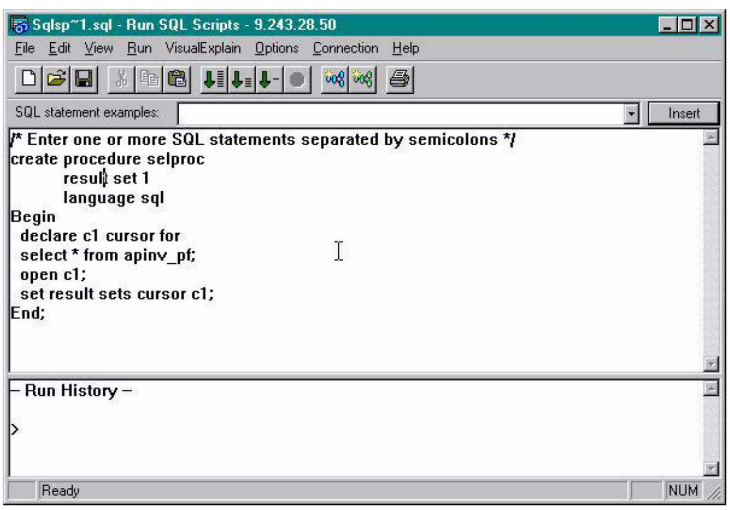
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Code and create SQL stored procedure

IBM i



- SQL stored procedure converted to C *PGM
- Must install:
  - OS/400 system openness includes

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-22. Code and create SQL stored procedure OD477.0

Notes:

Notice that the entire procedure can be specified within the SQL create procedure statement. In this example, you see that you have all the capabilities that you would expect from a normal programming language such as local variables and if tests.

The CREATE PROCEDURE statement is used in exactly the same way it was used to create an external stored procedure. The procedure is named in the same way. The procedure here, SELPROC, is not fully qualified, although we could have done so. Focus on the differences that you can see above:

- LANGUAGE **SQL**
- **BEGIN** indicates start of inline coding
- SQL statements are coded in the procedure and are separated by semicolons (**;**)
- **END** statement indicates the end of the inline coding and the end of the procedure. It is also followed by a semicolon (**;**). Ending each SQL statement with a semicolon is a rule that must be followed within the SQL SP. It is a good rule to follow when keying SQL statements in the Run SQL Scripts window.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

While not shown in this example, parameters may also be declared in the same way as we did when creating external procedures.

When you run this create procedure statement, DB2 for IBM i will create an ILE C program object that implements the requirements coded in your SQL stored procedure. This includes executing the SQL precompiler and ILE C compiler. Therefore, you must make sure that you install on the V5 development system:

1. DB2 Query Mgr and SQL Development Kit for IBM i
2. Option 13: OS/400 System Openness Includes

on the system where you will develop the stored procedures.

For those customers using V7R1, the ILE C and all other compilers are grouped in 5770-WDS (Application Development Toolset).

For those on V4 systems, compilers are packaged separately. You would purchase and install:

- 5769-ST1: DB2 Query Mgr and SQL Development Kit for iSeries
- 5769-CX2: ILE C for iSeries
- 5769-SS1 Option 13: OS/400 System Openness Includes

Once the C program object is created, it can be restored onto any other system and run with or without the SQL product and C compiler (with the same or higher level of OS/400).

ILE C Compiler: On version 7servers, the ILE C compiler is not required. IBM i has the needed modules for creating and compiling the intermediary ILE C code for SQL stored procedures and triggers. Also on version 7 servers, the SQL Development Kit for IBM i and Openness Includes are required only on the development system.

Once the ILE C object is created, it can be restored onto any V4R2 or higher system and run without the SQL Development Kit and ILE C compiler.

You can also use the GUI iSeries Navigator interface to create and run both SQL and external stored procedures. You can Run SQL Scripts as shown in this example or you could also follow these steps to create the procedure:

1. Right-click one of the servers library icons (the one where you want to create the stored procedure).
2. Select **New** > **Procedure** > **SQL**.
3. Use the notebook to name the procedure AND enter the parameter information and the SQL statements (or the external procedure information).

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out that the statements must be delimited by semicolons (**;**) as shown.

Also, point out the use of the BEGIN and END SQL statements to indicate the start and end of the SQL SP.

The BEGIN and END allow you to include more than one statement in your SQL SP. They are optional. You can code a single statement stored procedure. However, you would be limited to a single and very straightforward SQL statement such as a DELETE or an UPDATE.

Question:

Discuss this question with the class if there is time.

Would you be able to code a stored procedure that contained only a single SELECT statement and thus not require the BEGIN / END?

You could not code this. It would be an invalid SP and would not be created.

Since a SELECT produces a result set (even if only one row), you would have to code a DECLARE and an OPEN for the cursor, as well as the SET RESULTS SETS statement.

The remaining code is the same as we used for external SPs.

When an SQL procedure is created, SQL creates a temporary source file that will contain C source code with embedded SQL statements. A program object is then created using the `CRTSQLCI` and `CRTPGM` commands. The options used to create the program are the options that are in effect at the time the CREATE PROCEDURE statement is executed. The DB2 Query Manager and SQL Development Kit and ILE C products must be installed on the system when the SQL procedure is created.

Additional information — When the ILE C program is created, the Activation Group parameter is set to *CALLER.

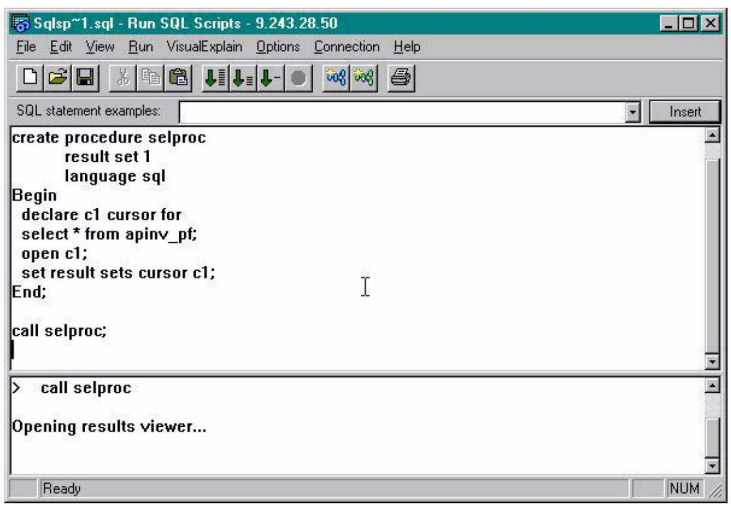
Transition statement — Later in this topic, we discuss how to code looping and error handling constructs in SQL Stored Procedures.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Call SQL stored procedure

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-23. Call SQL stored procedure OD477.0

Notes:

To run the procedure, we use the CALL statement as we did with an external procedure.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — You should point out that in order to run the CALL statement, we highlighted it and then specified **Run from Selected**. Point out the semicolon after the END SQL statement for the SP.

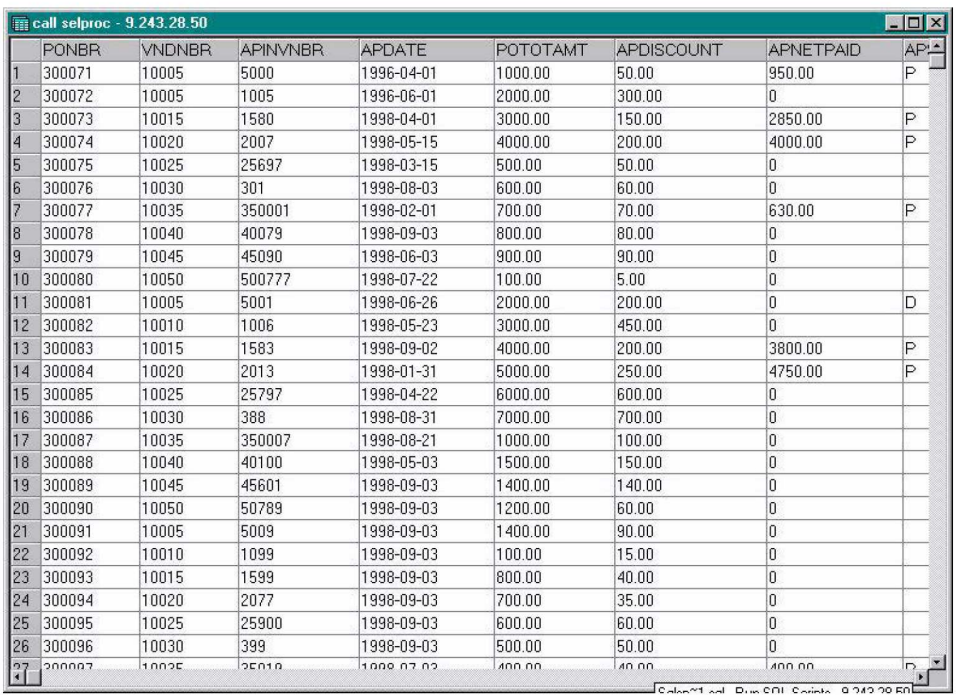
Without this semicolon, **Run from Selected** would have started from the CREATE PROCEDURE. It would have treated this as one SQL statement rather than two.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Result set

IBM i



	PONBR	VNDNBR	APINVNBR	APDATE	POTOTAMT	APDISCOUNT	APNETPAID	AP
1	300071	10005	5000	1996-04-01	1000.00	50.00	950.00	P
2	300072	10005	1005	1996-06-01	2000.00	300.00	0	
3	300073	10015	1580	1998-04-01	3000.00	150.00	2850.00	P
4	300074	10020	2007	1998-05-15	4000.00	200.00	4000.00	P
5	300075	10025	25697	1998-03-15	500.00	50.00	0	
6	300076	10030	301	1998-08-03	600.00	60.00	0	
7	300077	10035	350001	1998-02-01	700.00	70.00	630.00	P
8	300078	10040	40079	1998-09-03	800.00	80.00	0	
9	300079	10045	45090	1998-06-03	900.00	90.00	0	
10	300080	10050	500777	1998-07-22	100.00	5.00	0	
11	300081	10005	5001	1998-06-26	2000.00	200.00	0	D
12	300082	10010	1006	1998-05-23	3000.00	450.00	0	
13	300083	10015	1583	1998-09-02	4000.00	200.00	3800.00	P
14	300084	10020	2013	1998-01-31	5000.00	250.00	4750.00	P
15	300085	10025	25797	1998-04-22	6000.00	600.00	0	
16	300086	10030	388	1998-08-31	7000.00	700.00	0	
17	300087	10035	350007	1998-08-21	1000.00	100.00	0	
18	300088	10040	40100	1998-05-03	1500.00	150.00	0	
19	300089	10045	45601	1998-09-03	1400.00	140.00	0	
20	300090	10050	50789	1998-09-03	1200.00	60.00	0	
21	300091	10005	5009	1998-09-03	1400.00	90.00	0	
22	300092	10010	1099	1998-09-03	100.00	15.00	0	
23	300093	10015	1599	1998-09-03	800.00	40.00	0	
24	300094	10020	2077	1998-09-03	700.00	35.00	0	
25	300095	10025	25900	1998-09-03	600.00	60.00	0	
26	300096	10030	399	1998-09-03	500.00	50.00	0	

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-24. Result set OD477.0

Notes:

With very little work, we have the result of the SELECT statement that was run in our stored procedure.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

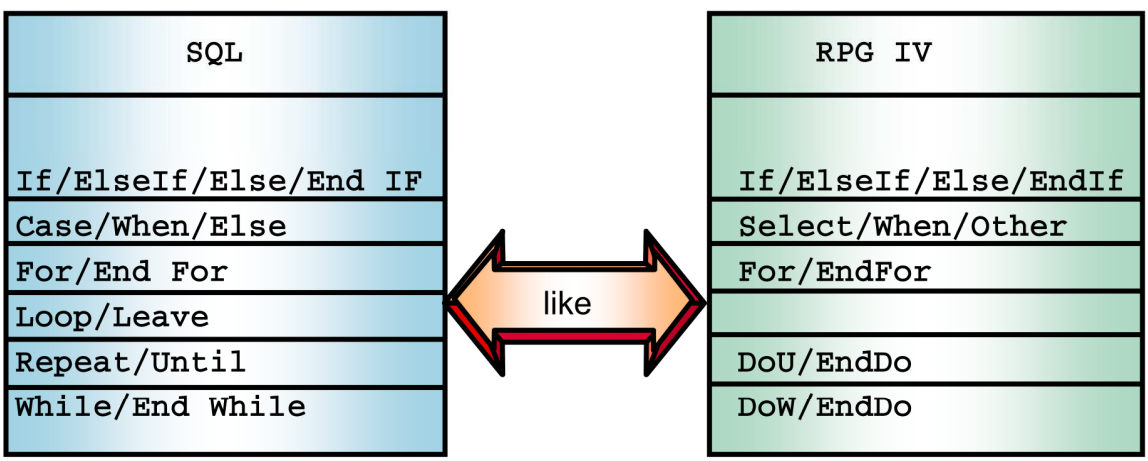
Transition statement — Next, we look at some of the additional language features available in SQL that make stored procedures very powerful.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Logic flow control in stored procedures

IBM i



© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-25. Logic flow control in stored procedures OD477.0

Notes:

These SQL statements are used to control the flow of an SQL stored procedure. They are not covered individually in class, but you can view details and examples in:

References:

- *Stored Procedures and Triggers on DB2 Universal Database for iSeries (ITSO Redbook)*
- *DB2 UDB for IBM i SQL Reference*

This visual allows you to compare the SQL statement to a similar statement in RPG IV. They may not be exactly the same, so it will be necessary to check the reference manual before you use them.

In the case of LOOP/LEAVE, they do not map directly to any IV statements. LOOP/LEAVE could be compared to the *do-forever until the program says stop*.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

The closest equivalent version 4 RPG IV code follows:

```
D Index           S              30P 0
D Quit            S               1A   Inz('N')

C                   For       Index = *Loval to *Hival

C     'Finish Y/N?' Dsply     '*REQUESTER'  Quit
C                   If        Quit = 'Y'
C                   Leave
C                   Else
C                   Do        100
C                   Enddo
C                   Endif
C                   Endfor

C                   Eval      *InLR = *ON
```

The closest equivalent of version 5 RPG IV code follows:

```
D Index           S               3P0  Inz(10)

 /Free
  For Index;                              // Indefinite Loop from 10
  Endfor;                                 // in steps of 1 - requires LEAVE

  *InLR = *On;

 /End-free
```

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose — Contrast SQL Logic flow control statements with RPG IV.

Details — This visual may stimulate some discussion. No detailed notes are provided. Be prepared to compare any of these to the RPG IV equivalent opcodes.

Transition statement — Next, let us look at an example that uses an SQL CASE construct.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Logic flow example in stored procedure (1 of 2)

IBM i

```
Create Procedure Give_Raise
                 (IN Employee_Number Char (6),
                  IN Rating Int )
                  Language SQL
       Case Rating
            When 1 Then
            Update employee
                       Set salary = salary * 1.10,
                       bonus = 1000.00
            Where Empno = Employee_number;

            When 2 Then
            Update employee
                       Set salary = salary * 1.05,
                       bonus = 500.00
            Where Empno = Employee_number;

            Else
            Update employee
                       Set salary = salary * 1.02,
                       bonus = 0.00
            Where Empno = Employee_number;

        End case;
```

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-26. Logic flow example in stored procedure (1 of 2) OD477.0

Notes:

The SQL CASE/WHEN/ELSE is like the RPG IV SELECT/WHEN/OTHER construct.

In this example, we have coded the logic to give a specific employee a raise and a bonus based on the prior year's performance rating. Both the employee number and the rating are input parameters to the stored procedure.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Details — Point out how the CASE works. This is one of two forms of a CASE construct (that we taught in OL37/OL370). This form is called a **Simple WHEN clause** since there is an expression between the CASE and the WHEN.

Transition statement — We next look at what the student can do to further his or her knowledge of SQL and stored procedures.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Logic flow example in stored procedure (2 of 2)

IBM i

If salary and bonus for employee number '000100'= 26,150.00 / 0.00

```
call give_raise('000100', 2);
```

Salary and bonus = 27,457.00 / 500.00

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-27. Logic flow example in stored procedure (2 of 2) OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

What's next?

IBM i

- Code more advanced stored procedures
  - Error handling
  - Debugging
  - More use of result sets
  - Try other logic flow statements

- Evaluate OL39/OL390 - DB2 for IBM i Advanced SQL Programming
  - Define and code SQL triggers
  - Advanced query features such as table expressions, views, subqueries, and OLAP specifications on SQL DML
  - Advanced join and hierarchical processing with SQL DML
  - User Defined Types
  - Define and code user-defined SQL functions
  - Integrating DB2 and XML data with SQL
  - Advanced features of SQL procedures including error handling and passing result sets

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-28. What's next? OD477.0

Notes:

For those who want more education in the use and development of stored procedures, attendance at **OL39**/**OL390**: *DB2 for IBM i Advanced SQL Programming* is recommended. The following information describes the course and its content:

DURATION: 3.0 days

PURPOSE:

This class is designed for IBM i programmers who want to learn how to use triggers and SQL stored procedures in DB2 for IBM i. It also teaches the additional topics.

Coding examples and machine exercises use the ILE RPG IV and ILE C languages.

AUDIENCE:

This course is designed for experienced IBM i programmers who have experience writing applications that use SQL to access the database.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

PREREQUISITES:

Before attending this class, the student should be able to:

- Use SQL Data Definition Language (DDL) to describe tables and views
- Use SQL UPDATE, INSERT, ALTER and DELETE statements to modify SQL tables
- Use the SQL SELECT statement to extract data from tables and views
- Navigate IBM i menus and use IBM i commands
- Use the System i Navigator for Database related functions
- Use PDM/SEU or LPEX Editor to code programs
- Use the ILE Source Debugger
- Code RPG IV and/or C programs on the IBM i

OBJECTIVES:

After completing this class, the student should be able to:

- Define and code SQL triggers
- Advanced join and hierarchical processing with SQL DML
- Code advanced features of SQL procedures, including error handling, and passing result sets
- Defining advanced SQL DDL syntax when creating tables and indexes to leverage DB2 for i data-centric features
- Advanced query features such as common table expressions, views, subqueries, and OLAP specifications on SQL DML (Data Manipulation Language) statements
- Define and code SQL user-defined functions
- Integrating DB2 and XML data with SQL

CONTENTS:

1) Advanced SQL DDL
   - Table
   - Index
   - Sequence
   - Global Variables
2) Advanced SQL DML
   - Common table expressions and derived tables
   - Views and subqueries
   - Joins and hierarchal processing
   - OLAP specifications
3) Procedural SQL Programming
   - User Defined Functions
   - Triggers
   - Advanced Stored Procedure coding
4) DB2 and XML integration

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose — In this class, we have only discussed some of the capabilities of SQL stored procedures. We did not talk about error handling or debugging. There is always more to cover, but we decided to offer more advanced capabilities as part of the next course in the IBM i SQL curriculum.

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OL387U03** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OL387U03** and click play.

c. Ask the students to indicate using green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OL387U03** and ask the students to call out the answers to the questions on the screen.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: Once a program has been registered as a stored procedure, you can SQL call it from a PC client using ODBC or Java.

2. Which of the following are types of stored procedures?
   a. Embedded procedure
   b. External procedure
   c. SQL stored procedure
   d. All of the above

3. An SQL CALL command
   a. Targets an external stored procedure
   b. Targets an SQL stored procedure
   c. Targets an RPG IV program with embedded SQL
   d. All of the above

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-29. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: Once a program has been registered as a stored procedure, you can call it from a PC client using ODBC or Java.

   The answer is True.

2. Which of the following are types of stored procedures?
   a. Embedded procedure
   b. External procedure
   c. SQL stored procedure
   d. All of the above

   The answers are External procedure and SQL stored procedure.

3. An SQL CALL command
   a. Targets an external stored procedure
   b. Targets an SQL stored procedure
   c. Targets an RPG IV program with embedded SQL
   d. All of the above

   The answers are Targets an external stored procedure and Targets an SQL stored procedure.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- List two types of stored procedures
- Explain the environments in which stored procedures can be used
- Create and run a simple stored procedure

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 10-30. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Unit 11. Database tools with System i Navigator

Estimated time

01:45

What this unit is about

This unit discusses the powerful database tools available with System i Navigator.

What you should be able to do

After completing this unit, you should be able to:

- List the System i Navigator database tools
- Use the System i Navigator database tools

How you will check your progress

- Checkpoint questions

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit objectives

IBM i

After completing this unit, you should be able to:

- List the System i Navigator database tools
- Use the System i Navigator database tools

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-1. Unit objectives OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

How do I access the database tools?

IBM i

- System i Navigator included in IBM i Access for Windows
  - Windows based interface (Windows *look and feel*)
  - Full and rich set of database administration tools
  - Manages multiple systems
  - Windows install required
- New in V6R1: Included in the IBM Systems Director Navigator for i
  - Web browser based tool for managing IBM i
  - Alternative to System i Navigator
  - Not including Mgmt Central, Run SQL Scripts, Visual Explain, Database Navigator
    - Manages single system
    - Automatically installed with IBM i
  - Example: https://yoursystem:2005/ibm/console/logon.jsp
- IBM Systems Director
  - Includes Systems Director Navigator for i

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-2. How do I access the database tools? OD477.0

Notes:

The web-based Systems Director Navigator for i offering is the strategic interface for IBM i. Available since 6.1, using the web console is as simple as opening a web browser and pointing it to your IBM i system (at the 2001 port). Director Navigator for i is considerably enhanced in i 7.1. Everything is included in the V6R1 and V7R1 IBM i operating system. Just ensure the HTTP Admin Server is running on your system (STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)), enter the above URL, and away you go! While most of the same database tools are available to users of Systems Director Navigator for i and System i Navigator, note that there are some exceptions. Management Central, Run SQL Scripts, Visual Explain and Database Navigator are *not* in Director Navigator for i.

System i Navigator is the Windows client and is packaged as part of the i Access for Windows product. It is the more traditional systems management console, and continues to be supported and enhanced. For V7R1, it contains the full set of database tools *and* Run SQL Scripts, Visual Explain and Database Navigator. This graphical user interface has been in place for several releases and since V5R3, functions have continuously been added at each new release.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

IBM Systems Director provides multi platform (z, p, and i IBM servers), distributed storage and network management capabilities with a single-point-of-control. It includes System Director Navigator for i and all of the same database tools.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement — Let's look at the database tools of System i Navigator.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Which tool should I use?

IBM i

- System i Navigator
  - The key pieces
    - Visual Explain
    - RUN SQL Scripts
- IBM Systems Director Navigator for IBM i
  - Key system performance tools
    - DB monitor
    - Plan cache
    - Object information
  - No client software to install

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-3. Which tool should I use? OD477.0

Notes:

System i Navigator and IBM Systems Director Navigator are typically used by the following:

- DB Engineers creating and maintaining the database
- Programmers analyzing the database and embedding SQL into HLL programs
- Technical support personnel troubleshooting problem areas

The method chosen typically depends upon the hardware and software configuration of the client PC and the required database functions.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

About System i Navigator

IBM i

System i Navigator ships with IBM i access for Windows.

- The database graphical user interface is a component of System i Navigator.
- System i Navigator comes with the base operating system for no extra charge.
  - None of the database GUI support requires an IBM i Access license.
- When installing IBM i Access on your PC
  - Database is not selected by default. Select **Custom** install.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-4. About System i Navigator OD477.0

Notes:

For general information, downloads, service packs and order/upgrade specifics on IBM i Access for Windows, check the following website:

http://www.ibm.com/systems/power/software/i/access

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

How do I get database fixes?

IBM i

- Fixes are provided using PTFs just like other operating system fixes.
- There is a special package for database group PTFs.
  - Can be used as part of your preventative maintenance
  - Convenient packaging of database fixes
  - Every group PTF is thoroughly tested as a set to ensure quality.
- Fix information and PTFs can be found on:
  - http://www-304.ibm.com/systems/support/i/fixes/index.html
- PTFs can be managed using Management Central
  - Included with System i Navigator for Windows

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-5. How do I get database fixes? OD477.0

Notes:

The database group fixes should become part of your routine PTF maintenance schedule.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

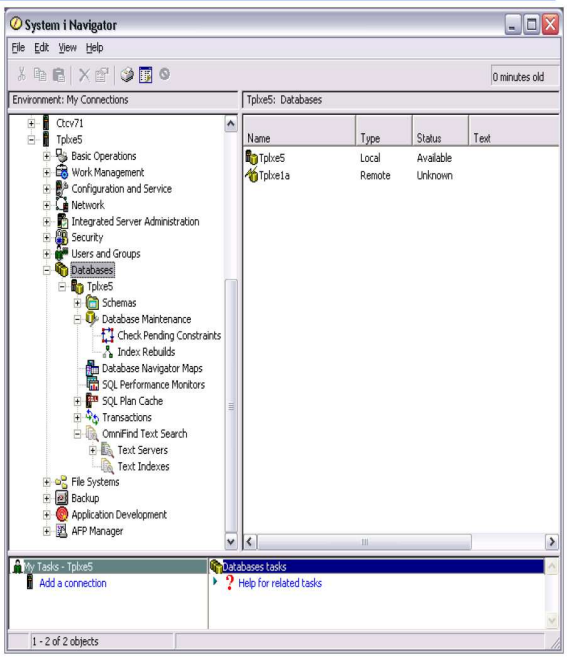
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

How do I start the database?

IBM i

- It is simple! If the system is up, the system database is up.
- Support for multiple database instances per server (since V5R2)
  - Requires special setup of Independent Auxiliary Storage Pools (IASPs)
- Allows you to work on multiple databases on a single system
  - Expand each database to work at that level
  - No need to *switch* back and forth when working with multiple databases on a specific system
- See relational database directory entries



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-6. How do I start the database? OD477.0

Notes:

The integrated relational database does not need to be started. System i Navigator gives you access to all databases defined to the server by using relational database directory entries.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

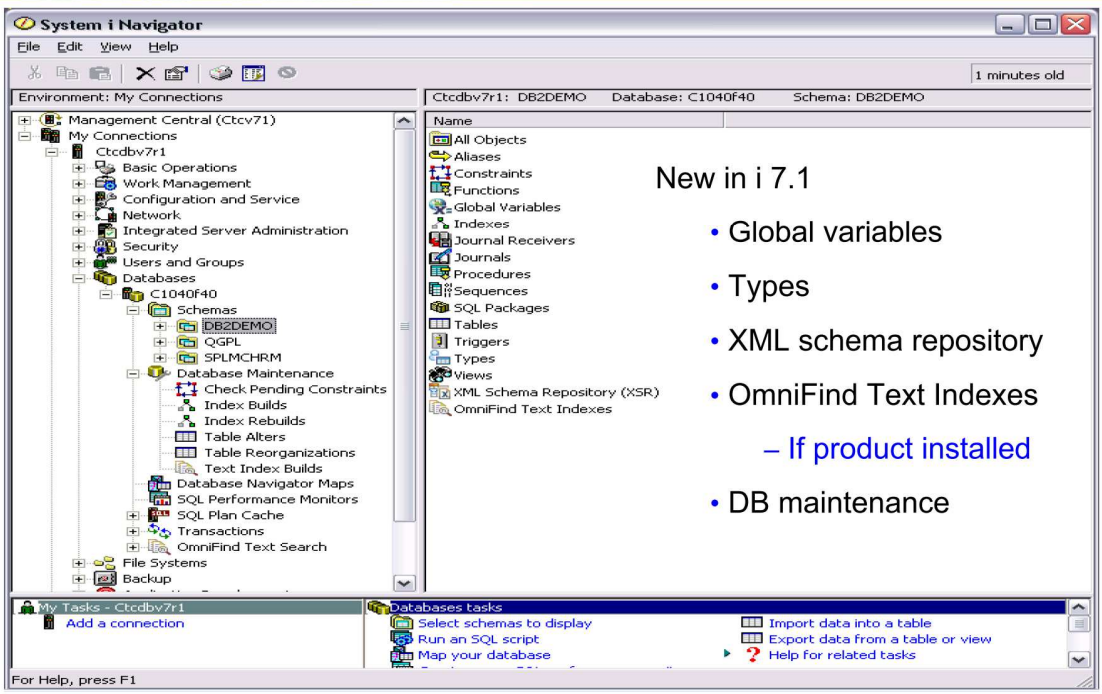
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Explorer-like interface



Folders under each schema (library), for each object type.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-7. Explorer-like interface OD477.0

Notes:

Navigator's Explorer-like interface allows users to leverage their knowledge of the graphical interface…translating into ease of use and relatively short learning curves.

System i Navigator supports the following i 7.1 enhancements to DB2 for i:

- Rich XML (Extensible Markup Language)
  a. XML data type stores XML documents
  b. Generate XML documents from existing relational data
  c. Decompose XML documents into relational columns

  The XML schema repository (XSR) is a repository for all XML artifacts used to process XML instance documents stored in XML columns.
- Global variables can be named, defined and used between multiple SQL statements

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- OmniFind Text provides support for searching XML documents
  a. Search elements of an XML document (for example, customer name = Smith)
  b. SQL statements can use OmniFind to search for the XML documents
  c. Available with IBM I for no additional charge

Database maintenance of constraint repair, index building, journaling, and reorganizing tables can be performed using System i Navigator.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — Windows Explorer-like interface.

Additional information —

Transition statement —

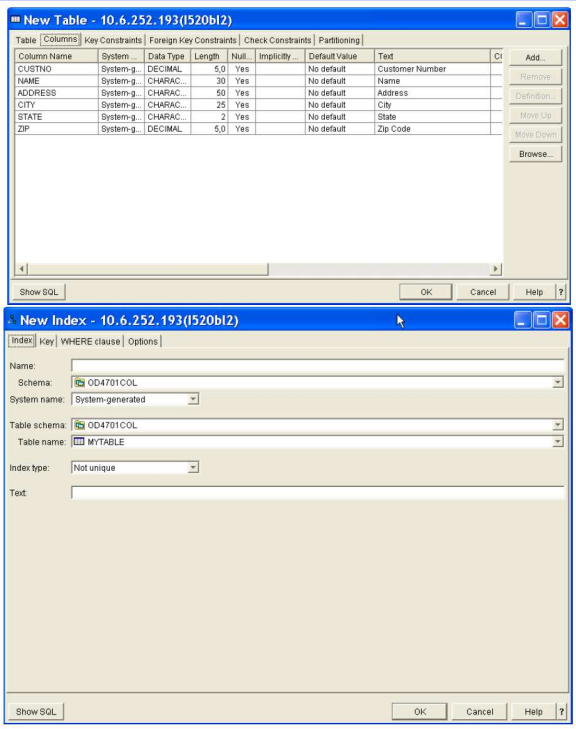
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Database object creation

IBM i

- Create database objects with no storage considerations (including tables!)
- Browse function for reusing existing column definitions (field reference)
- Robust Alter Table support
  - No need for special utilities
- How do I rebalance indexes?
  - It is simple! The database does it for me.
- Can create SQL versions of DDS logical files



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-8. Database object creation OD477.0

Notes:

When creating database objects, data is automatically striped across the disk. There is no need to worry about data extents nor data fragmentation. The system keeps track of the data and its location using single level storage and storage management. Even reorganizing tables so as to remove deleted records can be avoided by having the system automatically reuse deleted record space.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

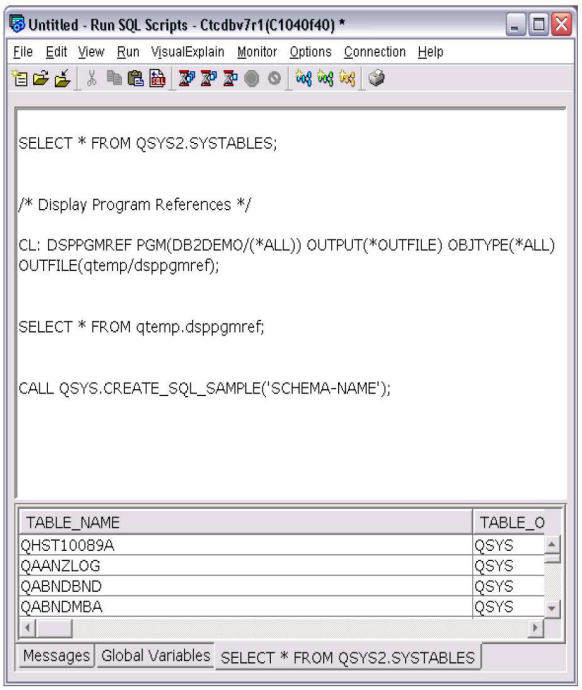
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Run SQL Scripts

IBM i

- Create/save scripts for repetitive tasks
- Both SQL statements and CL commands
- View and save output for SQL queries
  - Save to Options
- Call stored procedures with result sets and/or OUTput parameters and launch iSeries System Debugger
- Do Visual Explain on queries, for performance analysis and tuning
- Syntax error highlighting and font chooser
- SQL state errors in message box, including second level text
- Prompting of CL commands
- Ignore drop error messages
- Support for Global Variables 7.1 (new tab)



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-9. Run SQL Scripts OD477.0

Notes:

In order to prototype an SQL statement prior to embedding it into a high level language (HLL) program, the Run SQL Scripts interface can be used. This interface can also be used to fine tune SQL statements prior to using them in a production environment. Visual Explain is a valuable tool that can help users fine tune their SQL statements.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

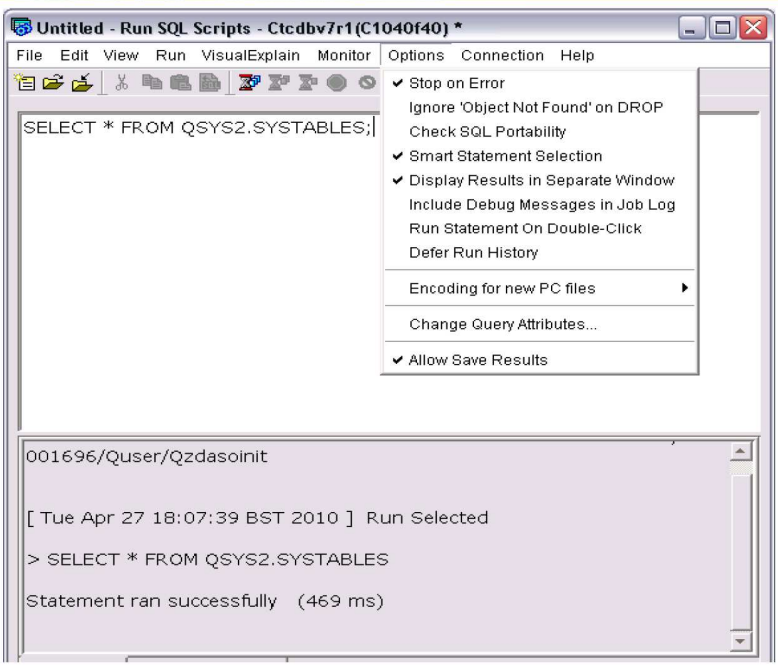
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Save results (1 of 3)

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-10. Save results (1 of 3) OD477.0

Notes:

By choosing the ‘Allow Save Results’ from the menu option pull-down, you can save your query results in a number of versatile formats.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

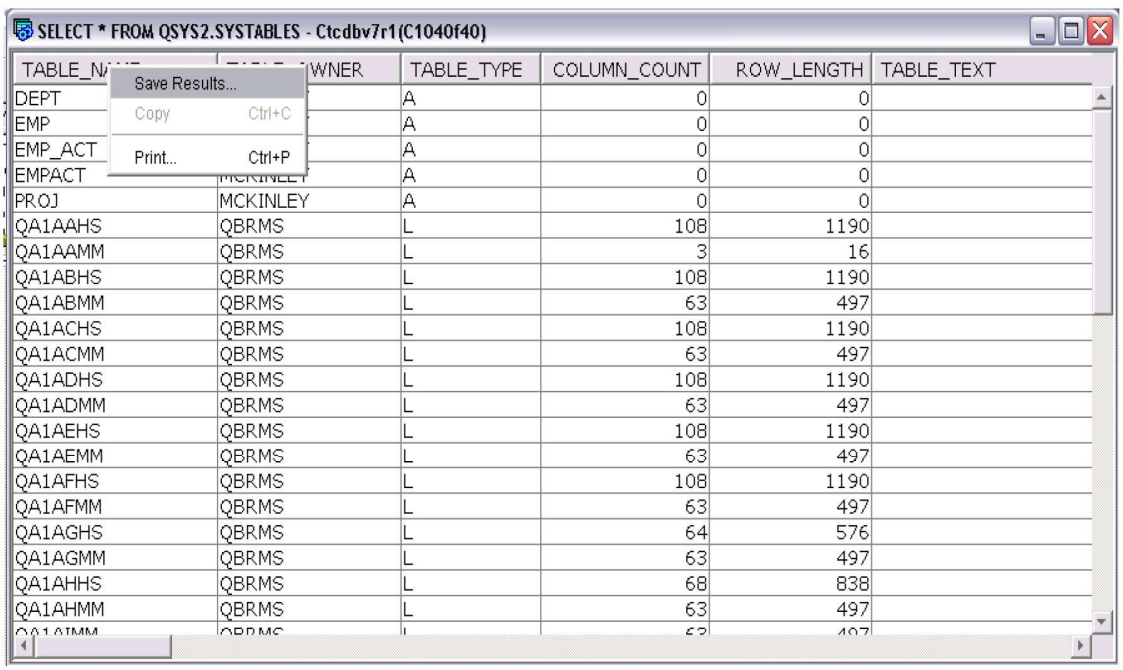
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Save results (2 of 3)

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-11. Save results (2 of 3) OD477.0

Notes:

Once your results are returned, right-click them and select **Save Results**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

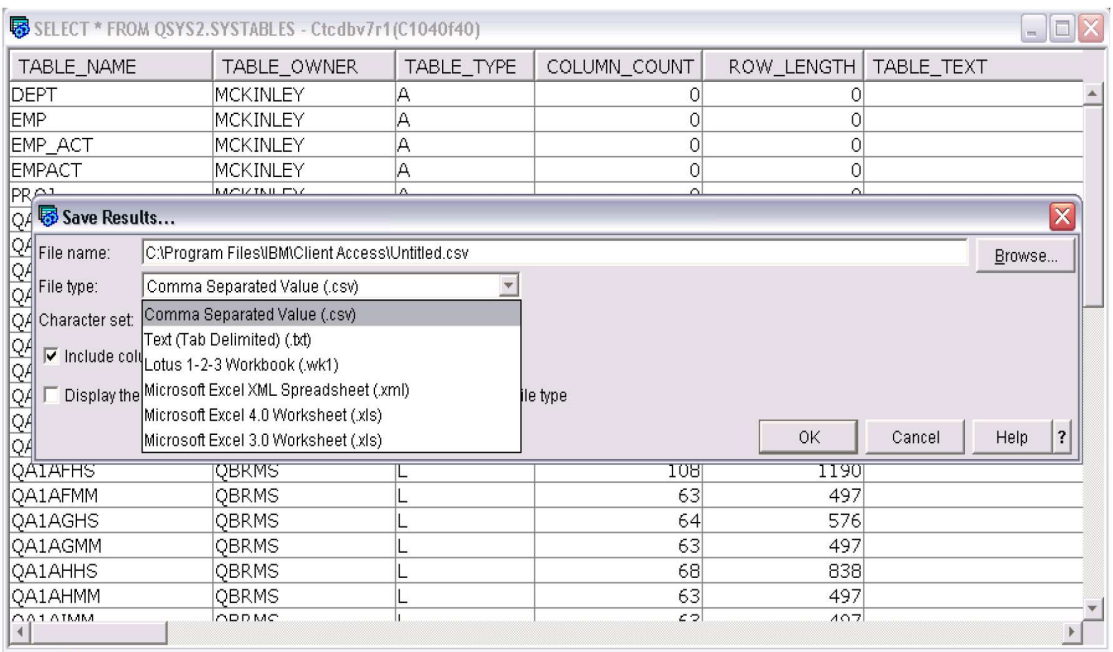
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Save results (3 of 3)

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-12. Save results (3 of 3) OD477.0

Notes:

You can then select a file type from the pull-down.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

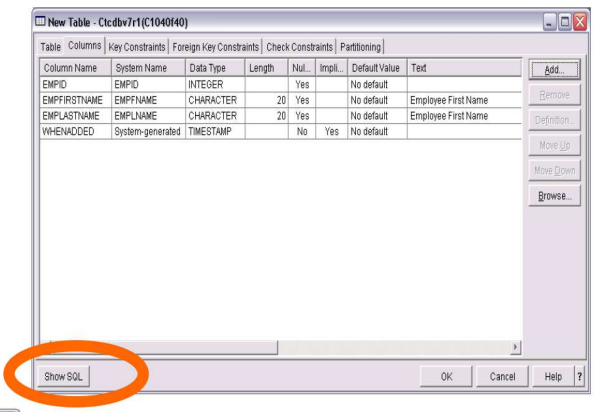
© Copyright IBM Corp. 2010, 2013

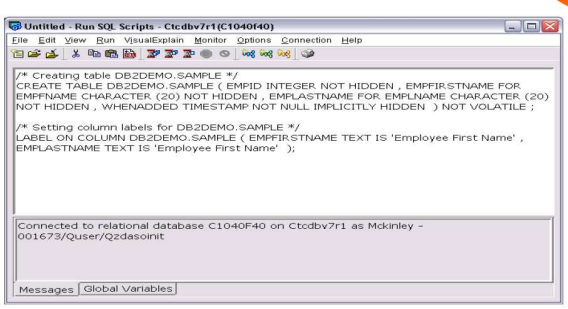
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Show SQL statement from create dialogs

IBM i

- SQL statement is recorded as you fill in dialog
- Option to show the SQL statement being built
- Great for learning SQL syntax
- Allows for statement customization





© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-13. Show SQL statement from create dialogs OD477.0

Notes:

During the create dialogs of System i Navigator, you can show the SQL statements generated by the dialog.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

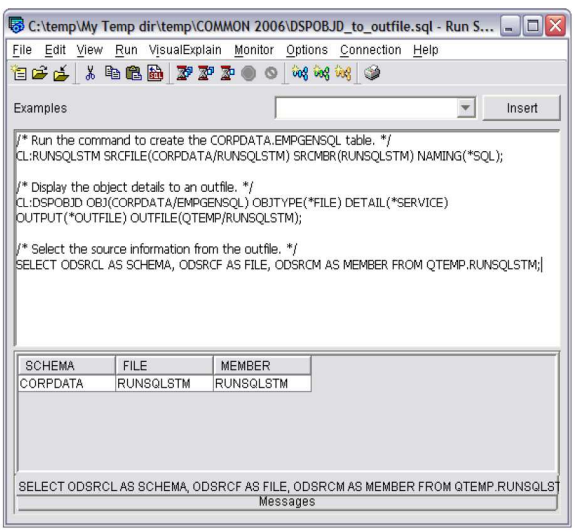
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Keeping schema source

IBM i

- DDS: Source files (object source file info maintained)
- SQL: Run SQL statements PC or IFS files
- SQL: RUNSQLSTM source files (object source file maintained)



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-14. Keeping schema source OD477.0

Notes:

Just as data description specification (DDS) source information used to create objects is kept in source files, SQL statements can also be saved in PC or IFS (integrated file system) files for documentation. The RUNSQLSTM can be used to execute saved statements in order to recreate objects, if needed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details — DSPOBJD can show you where the source came from.

Additional information —

Transition statement —

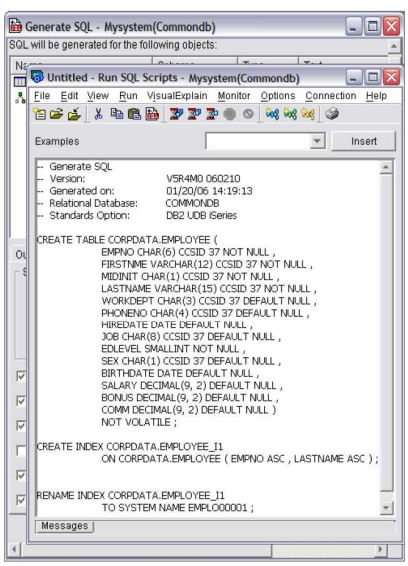
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Generate the SQL

IBM i

- Regenerate SQL CREATE/ALTER statements directly from the existing objects
  - Helpful for when the original statements are no longer available
  - Teach programmers about SQL syntax of objects

    See what the SQL form of my DDS-created files would be like



- Include generating short names of objects
- Does not generate RI and triggers for tables
- Includes SET OPTIONS statement
- **V6R1:** Now supported at folder level versus schema or object
- 7.1: Includes privileges

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-15. Generate the SQL OD477.0

Notes:

An SQL statement can be regenerated (from the existing object) and placed into a run SQL scripts window. Once there, it can be saved to an IBM i source file or an IFS file.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

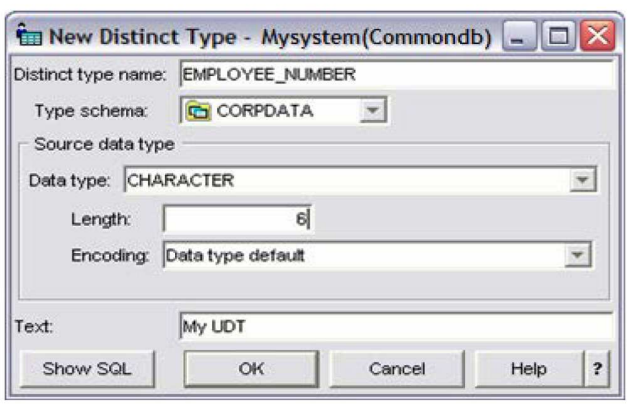
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Data dictionaries

IBM i

- DDS File Field Reference Files
- SQL CREATE TABLE AS (subselect)
  Ex: CREATE TABLE SalesByRegion AS
  (SELECT region, SUM(to_sales) FROM sales GROUP BY region)
  WITH NO DATA
- User-defined types



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-16. Data dictionaries OD477.0

Notes:

With DDS, you can define and create a physical file that has no data, but is used as a reference file for defined fields. Remember that all files have a record format that contains field definitions. I can create the same type of field reference file with SQL. Once created, the field reference file can be used like any DDS created field reference file.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

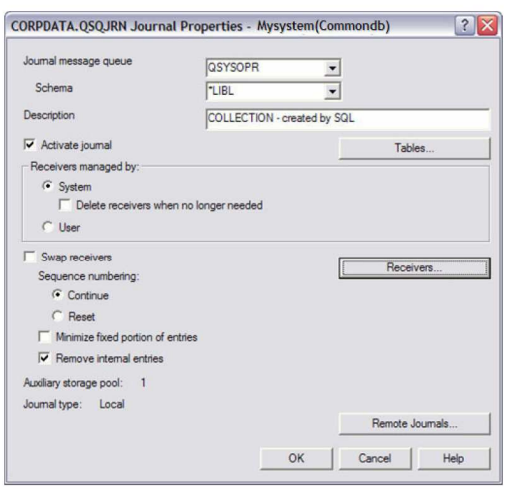
© Copyright IBM Corp. 2010, 2013

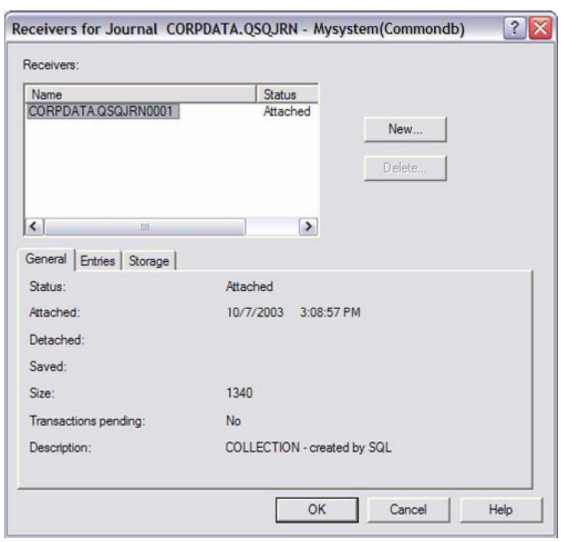
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Journaling

IBM i

- Default journal automatically created and started when creating a schema
- Journal receivers contain journal entries for recovery





© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-17. Journaling OD477.0

Notes:

System i Navigator allows management access to the journal and receivers that were automatically created in your schema.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

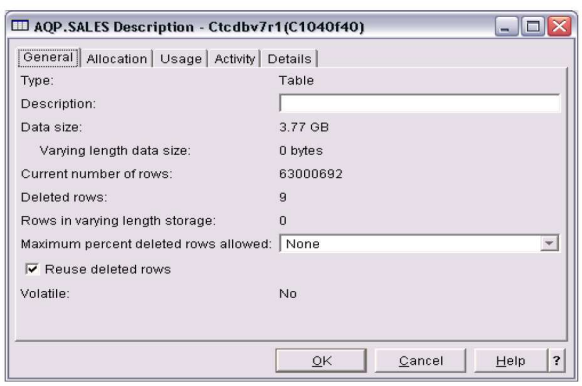
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

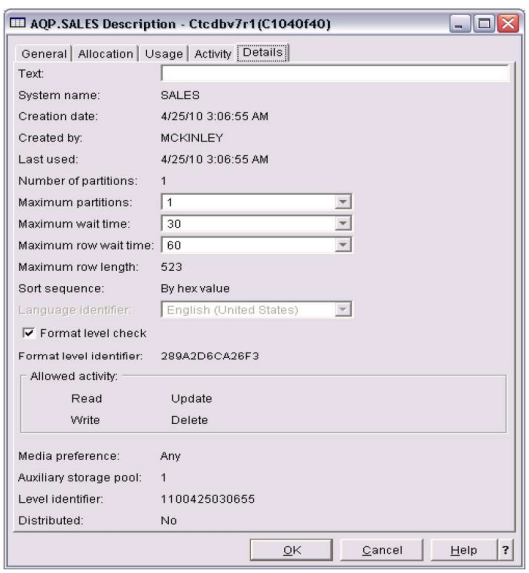
Table description

IBM i

Answers questions like: How big is this table/file getting anyway?

- Works for indexes and views too!





© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-18. Table description OD477.0

Notes:

Right-click a table, index or view object and select the table description data for viewing. Various tabs allow you to see dynamic statistics on file activity, usage, allocation, and general information.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

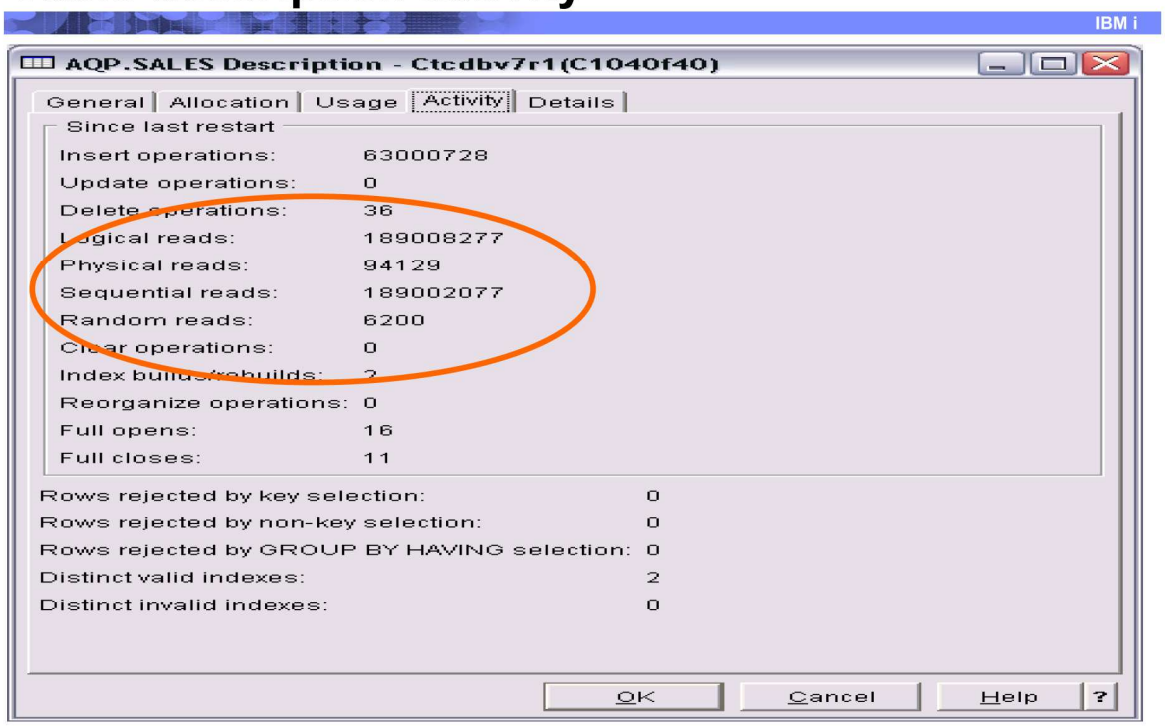
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Table description activity



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-19. Table description activity OD477.0

Notes:

Basic performance data is available. The number of logical, physical, sequential and random reads can be easily seen.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

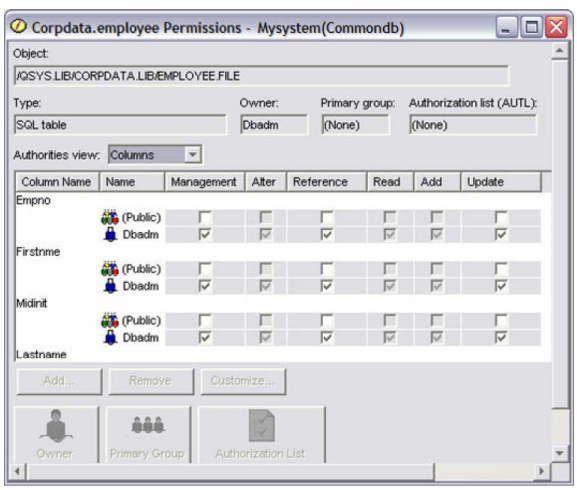
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

How do I synch up my db and user IDs?

IBM i

It is simple! Group and user authorizations are managed by operating system.

- SQL syntax maps to IBM i security types
- Column level security
- Program versus object level security
- Views
- Exits
- Auditing



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-20. How do I synch up my db and user IDs? OD477.0

Notes:

The IBM i manages security as usual. It is viewable from System i Navigator by selecting Permissions on the objects. Users and their authorities are displayed in similar fashion to the command `DSPOBJAUT`.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

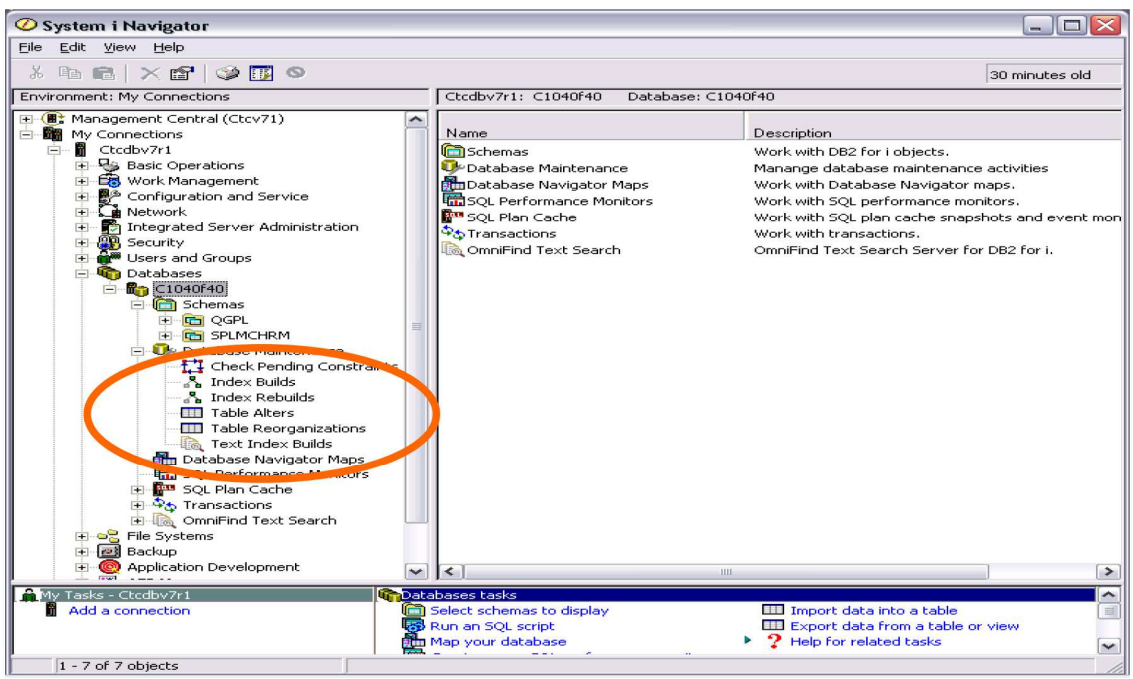
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

New in 7.1: Database Maintenance folder



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-21. New in 7.1: Database Maintenance folder OD477.0

Notes:

If your i server is pre-V7.1, you will only see the check pending constraints and index builds under the database maintenance folder.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

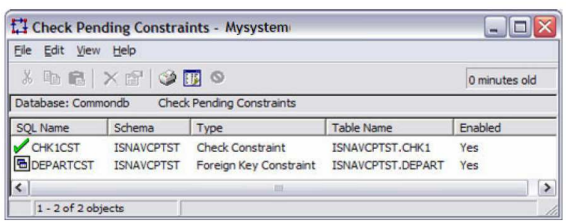
© Copyright IBM Corp. 2010, 2013

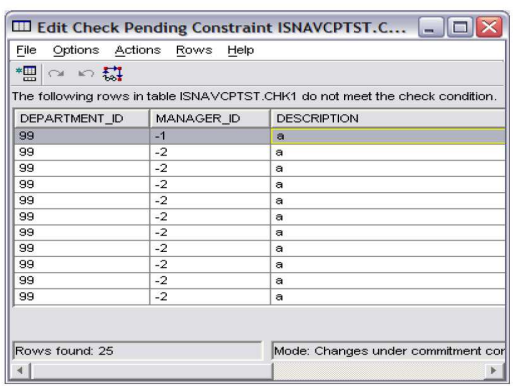
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Constraint repair

IBM i

- Works for both check constraints and referential integrity constraints
- Show problem rows in table
- Able to update column values or delete entire rows
- Quick access to check constraint definition for check constraint repair
- Quick access to parent table rows for RI constraint repair





© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-22. Constraint repair OD477.0

Notes:

Database constraint repair is available under the Database Maintenance folder of System i Navigator.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

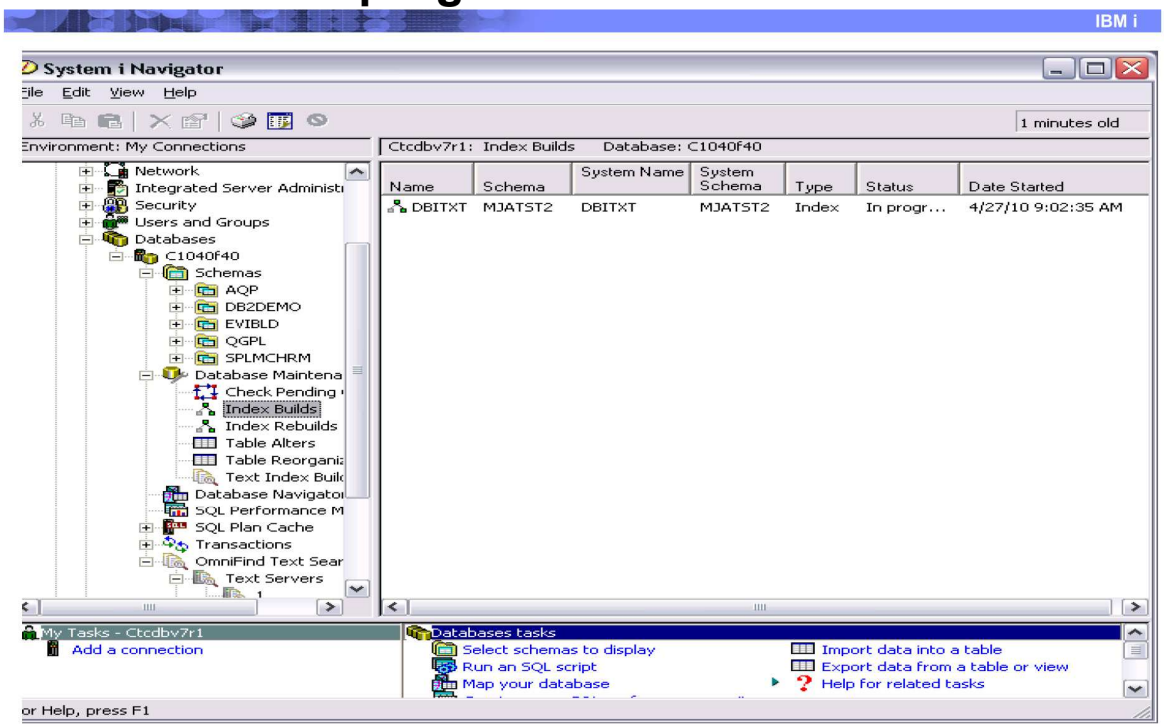
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Index builds in progress



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-23. Index builds in progress OD477.0

Notes:

You can readily check on index build activity for the system under the **Database Maintenance** folder by clicking **Index Builds**. The next visual illustrates the detail available when selecting one of the displayed index builds.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

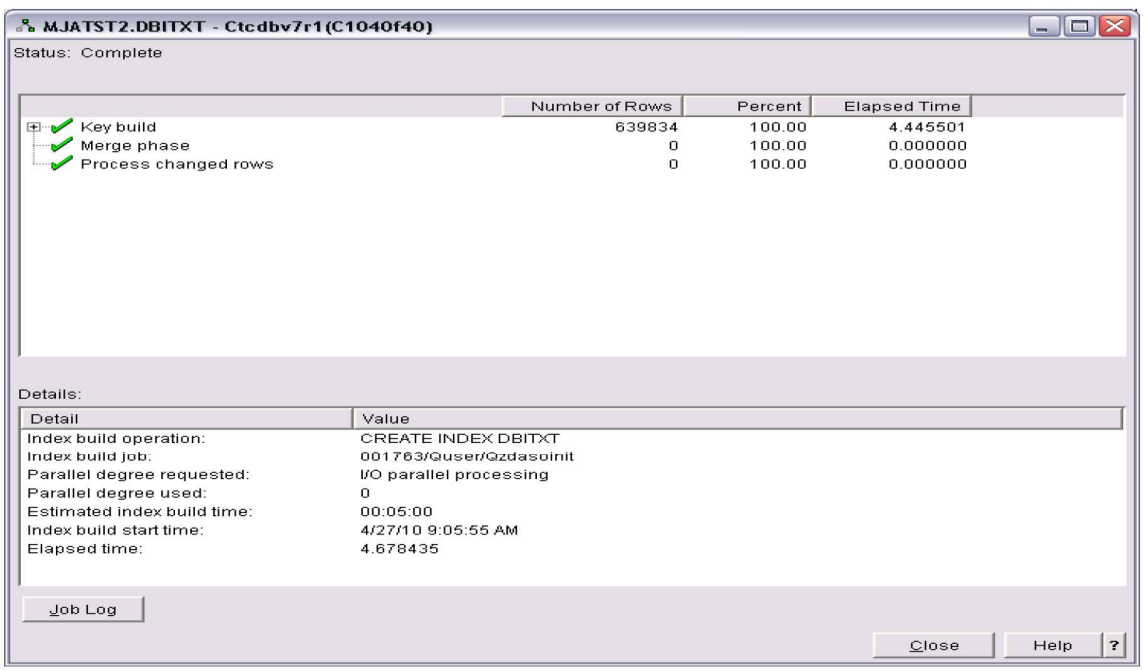
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Right-click build in progress

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-24. Right-click build in progress OD477.0

Notes:

By right-clicking a build in progress, you can get details of the index build operation. The number of rows involved as well as the time taken for the rebuild are reported.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

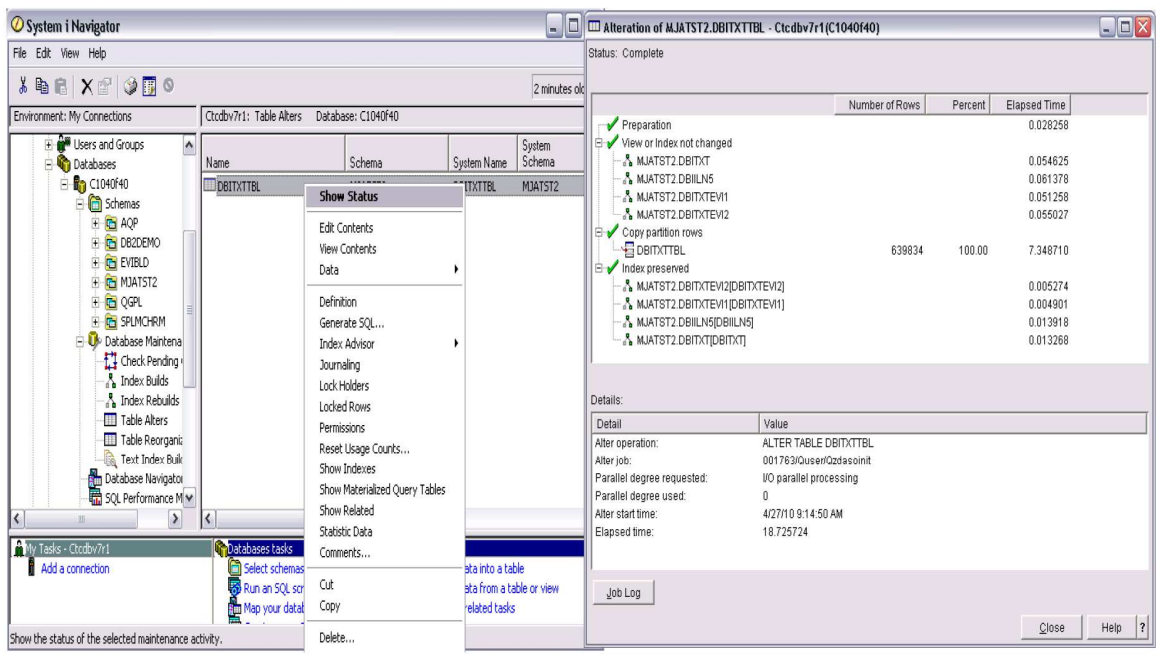
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Alter table status



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-25. Alter table status OD477.0

Notes:

You can check on the status of a table that has been altered by first clicking **Table Alters** under the Database Maintenance folder. Then by right-clicking the table of choice and selecting **Show Status**, you can check on the details.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

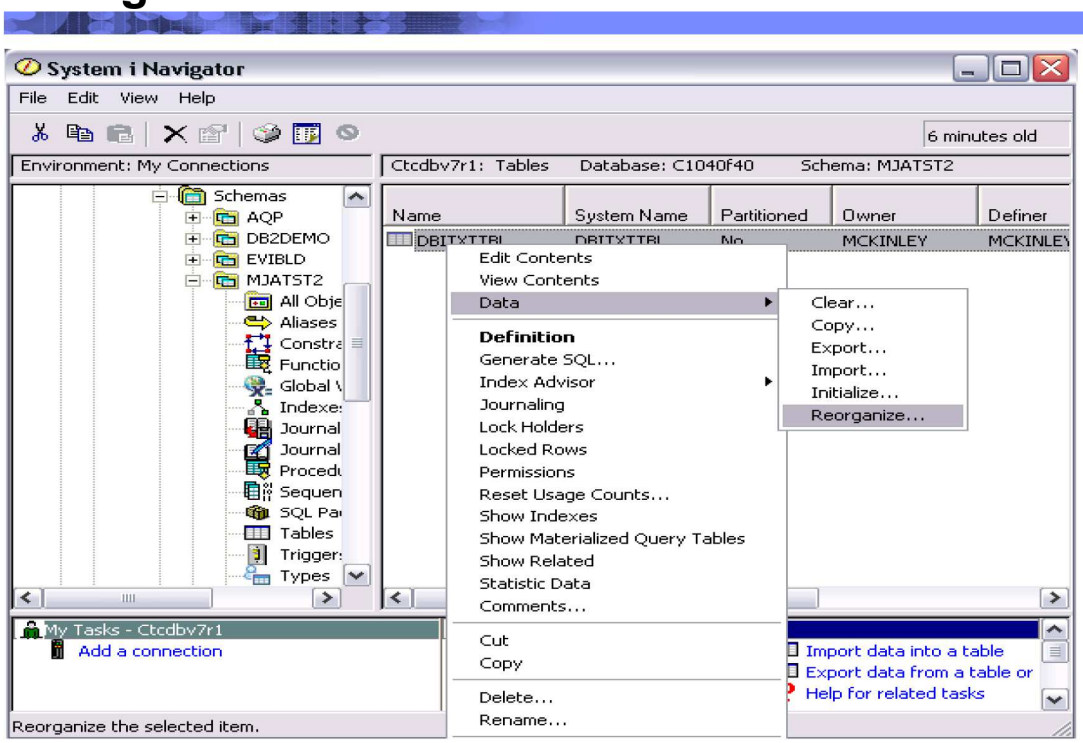
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Reorganize table



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-26. Reorganize table OD477.0

Notes:

Right-click the object, and select **Data** followed by **Reorganize** to reorganize the table's rows.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Reorganize table progress monitoring

IBM i

- New reorganize table options:
  - Allow suspend (cancel)
  - Allow online access
  - Allow changes
  - Influence when to rebuild access paths



- Allow reorganize to be suspended and resumed later

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-27. Reorganize table progress monitoring OD477.0

Notes:

The same versatile options available on the `RGZPFM` command are available for the reorganize process done with System i Navigator. The resulting command parameters can also be viewed or altered.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

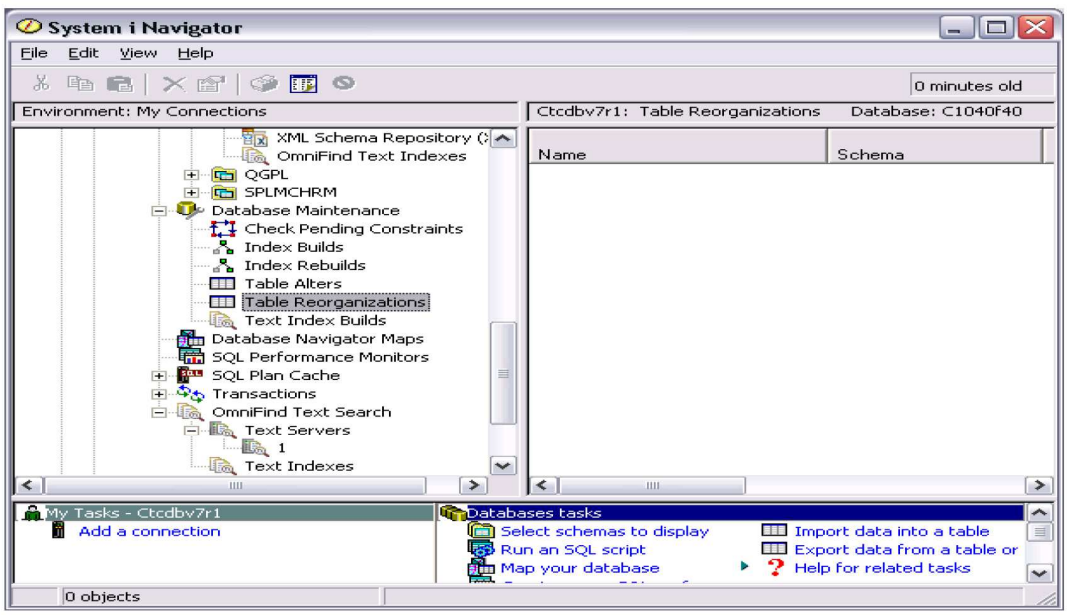
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

From the maintenance folder

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-28. From the maintenance folder OD477.0

Notes:

A status display will report upon a requested file reorganization. You can also check on the status of reorganizations by selecting Table Reorganizations from the Database Maintenance folder.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

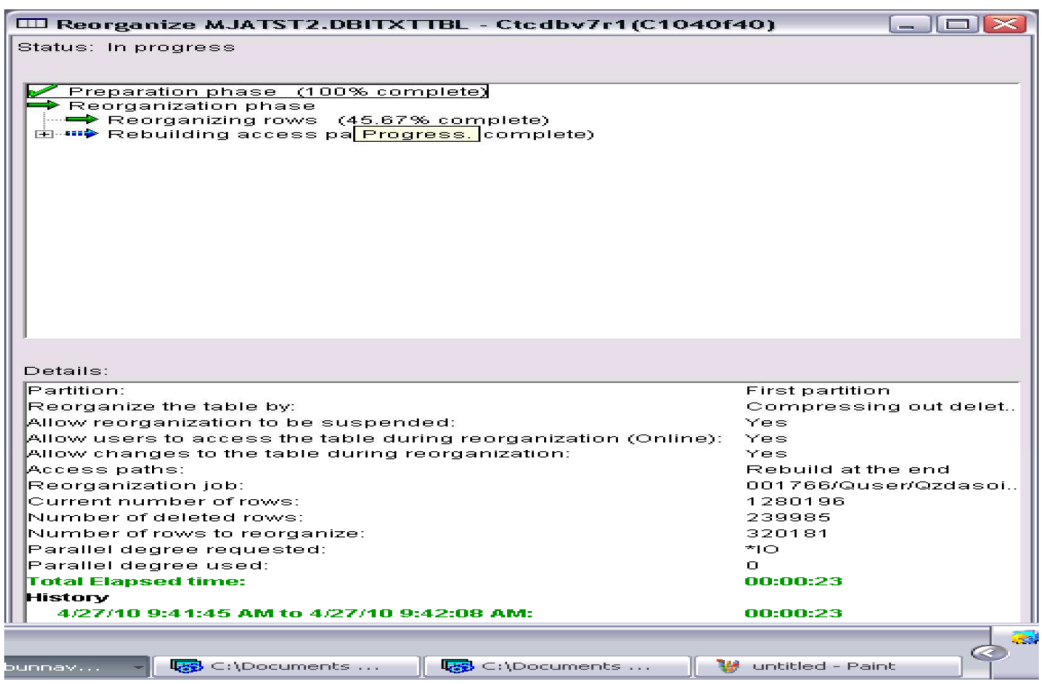
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Reorganization status display

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-29. Reorganization status display OD477.0

Notes:

Here is an example of the reorganization status display. This will report upon percentage of completion, total elapsed time, number of current rows, number of deleted rows, number of rows to reorganize and access path maintenance strategy.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

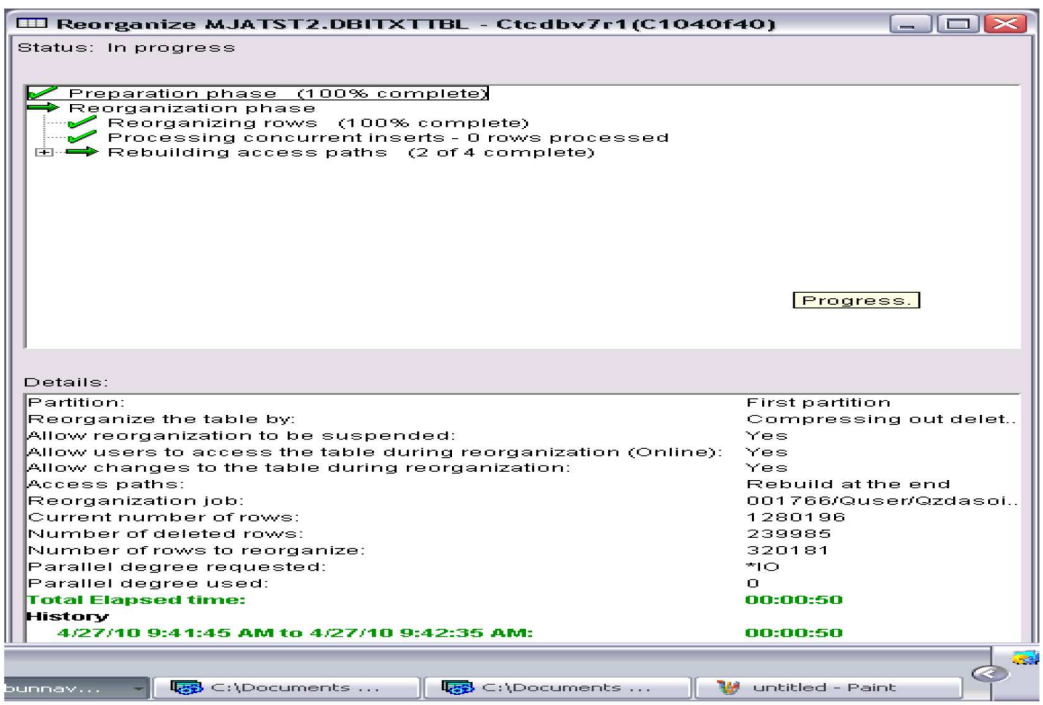
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Reorganize index build phase

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-30. Reorganize index build phase OD477.0

Notes:

The access paths (however many of them) are reported upon at the top of the display as they are rebuilt.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

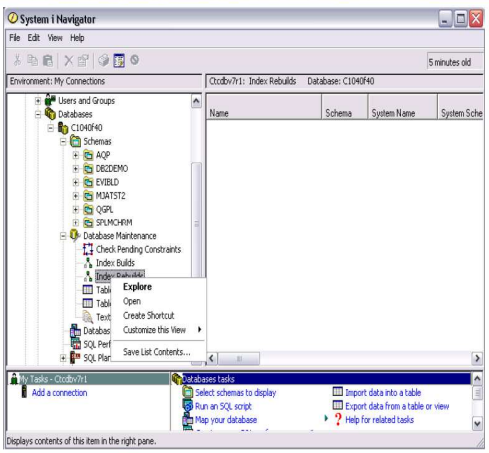
Transition statement —

© Copyright IBM Corp. 2010, 2013

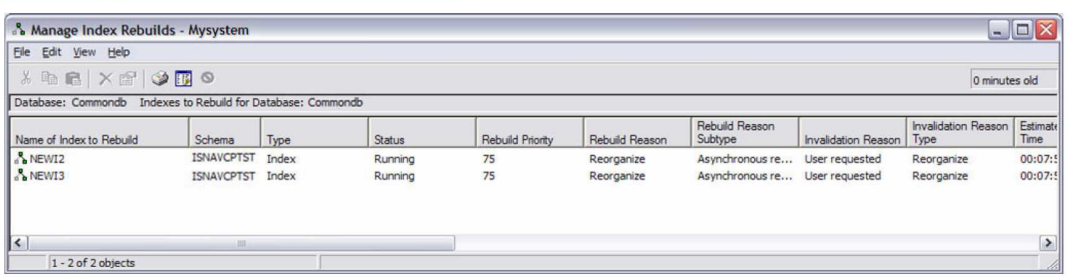
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Manage index rebuilds

IBM i



- Manage index re-creation after reorgs, database restores, and so on
  - Priority adjusting
  - Monitor progress
- GUI version of EDTRBDAP system command



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-31. Manage index rebuilds OD477.0

Notes:

System i Navigator provides a graphical user interface equivalent to the **EDTRBDAP** command. After database restores or reorganizations, you can use the Database Maintenance folder to manage your index rebuilds according to your schedule.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

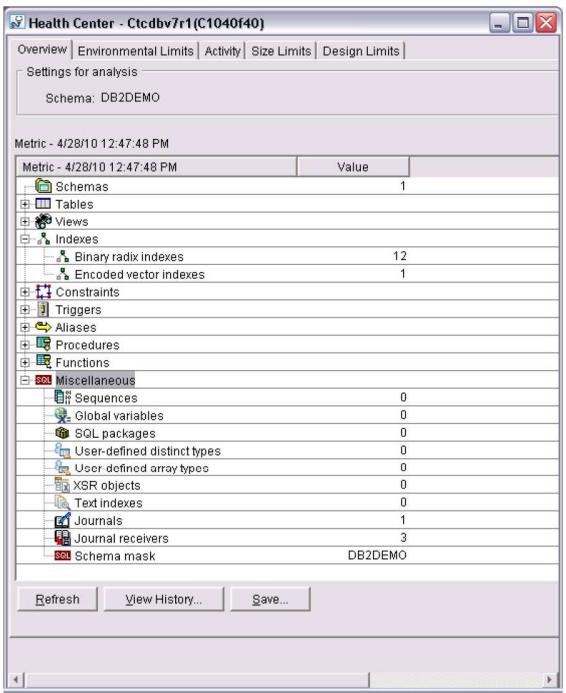
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Health Center

IBM i

- Overview to see database object counts by category
- View database wide or schema specific
  - % wildcards can be used
- Capture to history file
- View history of counts for trending
- Environmental limits
- V6 and V7 i servers



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-32. Health Center OD477.0

Notes:

The Health Center allows you to get an overview of your entire database or selected schema. You can see the types and number of objects that your database contains. In addition to overview information, objects approaching their environmental limits (for example their size, the number of open SQL cursors, maximum call level interface handles) can be displayed. Historical data collections can be captured and viewed over time to get an idea of how your database trends.

Right-click the database of choice and select Health Center to initiate.

Environmental limits are only available when connecting to a V6R1 system or higher.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

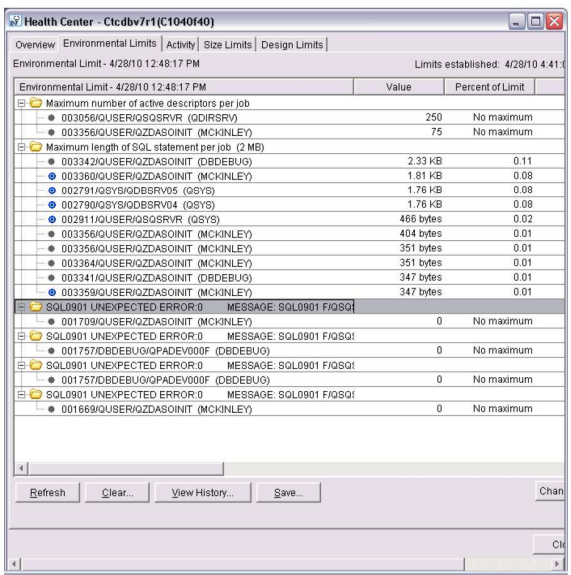
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Environmental limits

IBM i

- DB limits within jobs
  - Lob Locators
  - Active Descriptors
  - Max SQL STMT Size
  - SQL 901 category



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-33. Environmental limits OD477.0

Notes:

Large objects (LOBs) can be as large as 2,147,483,640 bytes or 2 GB long.

Unexpected SQL errors (SQL901) can be investigated.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

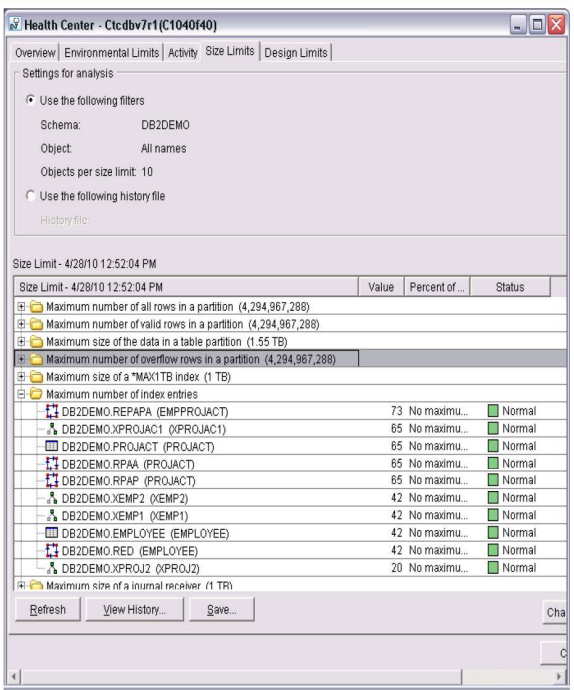
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Health Center

IBM i

- Size and Design Limits views
  - Size limits are things that change by normal operations
  - Design limits are things that change by application design
- View database wide, schema specific, or by object name
  - % wildcards can be used
- Categorized by limit type, includes system limit!
- Status indicator for *within percent of limit*
  - Changeable threshold
- Capture to history file
- View history of limits for trending



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-34. Health Center OD477.0

Notes:

Size limits include:

- Maximum number of all rows in a partition
- Maximum number of valid rows in a partition
- Maximum size of the data in a table partition
- Maximum size of a journal receiver

Design limits include:

- Maximum number of columns in a table or view
- Maximum length of a row
- Maximum row length with LOBs
- Maximum number of columns in an index key
- Maximum length of a key
- Maximum number of constraints on a table

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

- Maximum length of a CHECK constraint
- Maximum number of tables referenced in a view or logical file
- Maximum number of members
- Maximum number of parameters in a procedure

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

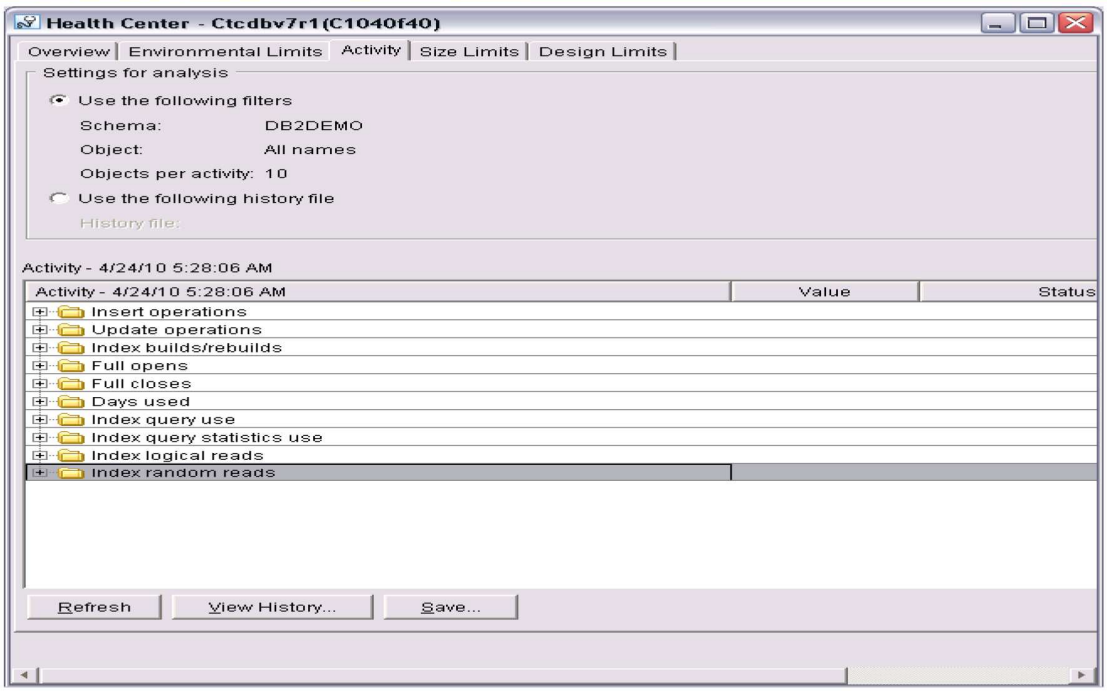
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Additional activity counters in i 7.1



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-35. Additional activity counters in i 7.1 OD477.0

Notes:

The **Activity** tab can be used to determine which objects in your database were used most often and how they were used.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

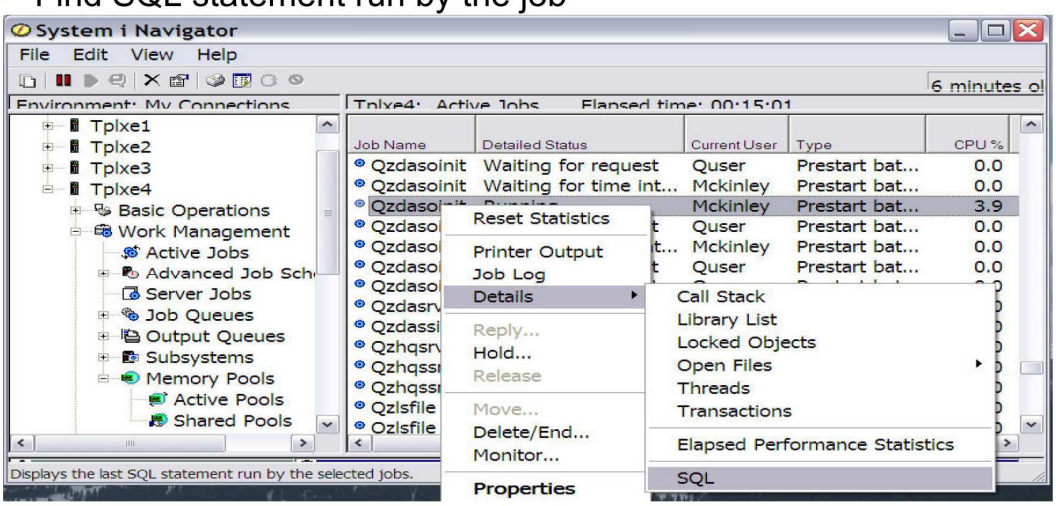
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Work management for database jobs

IBM i

- Work with jobs
  - Find a job for which someone is reporting problems
  - Customize list by user, job information, and column filtering
  - Find SQL statement run by the job



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-36. Work management for database jobs OD477.0

Notes:

Right-click a job, select **Details,** then select **SQL** to take you to the same window as current SQL for a given job.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

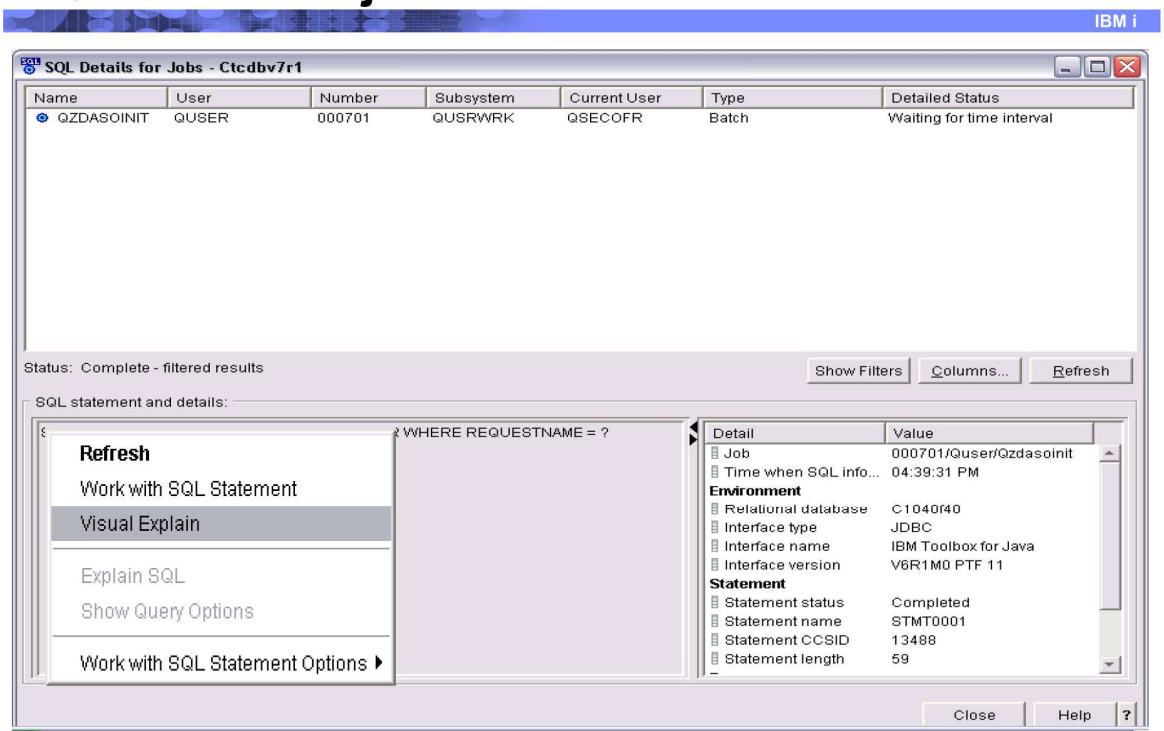
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL details for jobs



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-37. SQL details for jobs OD477.0

Notes:

After highlighting the SQL statement, right-click it and select Visual Explain. This allows you to work with the SQL of a given job.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

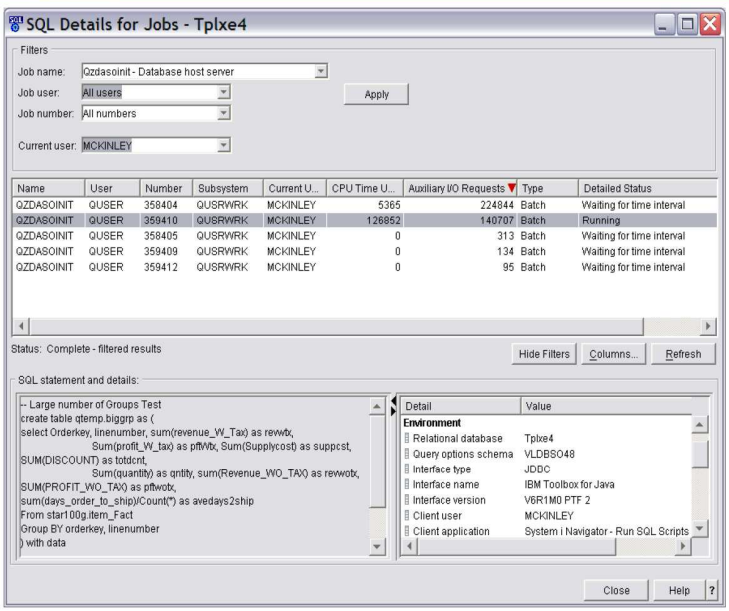
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL Details for Jobs

IBM i

- Launched from Databases folder
- Find a job for which someone is reporting problems
- All columns can be sorted
- Shows schema name, program name, service program or package name, and system IP address
- Can view job log, Show SQL Details, or end job
- Edit SQL launches SQL statement into Run SQL Scripts window, or can go directly to Visual Explain
- Can see environment, statement, and resource details
- Refresh allows for monitoring
- **Allows monitor start on a job in the list (new in 7.1)**



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-38. SQL Details for Jobs OD477.0

Notes:

SQL details can also be retrieved by launching from the Databases folder (right-click **Databases** > **SQL Details for Jobs** and finding the user job of interest.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

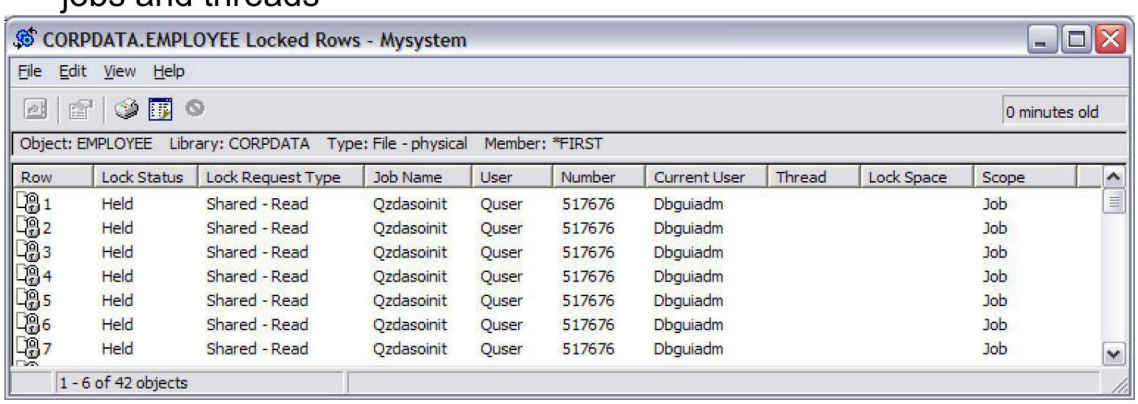
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Who has those table rows locked?

IBM i

- See what jobs have which rows locked for a given table
  - Shows the row number locked
  - Current User column shows *real* user ID if from a client
  - Job information provided
  - Viewing of row locks also available in Work Management folder for jobs and threads



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-39. Who has those table rows locked? OD477.0

Notes:

Row lock support has been added for jobs and threads. You can now see, to the row level, locks that are held on a file/table while viewing through the Work Management folder of System i Navigator. Row locks were already viewable through the Database folder in System i Navigator in previous releases.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

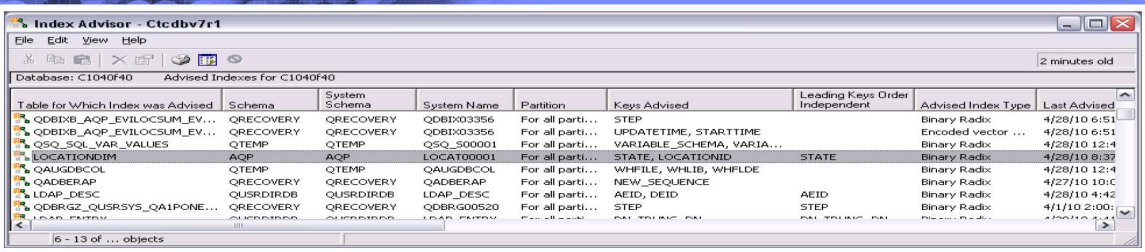
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Live analysis: Index Advisor

IBM i



DB2 for i can help with your indexing strategy

- System-wide repository of DB2 Advised Indexes
  - System view makes it easier to create index with biggest impact
  - Gauge benefits of index creation based on frequency, last occurrence, common keys
  - Available at the database, schema, or individual table level
- Enhanced Index Advice (SQE only)
- Ability to view SQL statements that caused index advice (SQE only)
- Index Advice Condenser to ease DB2 Performance tuning
  - http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0701forstie/

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-40. Live analysis: Index Advisor OD477.0

Notes:

The Index Advisor can analyze your entire database or selected schema in order to recommend indexes that should be built over the tables. Be sure to observe the Maintained Temporary Index (MTI) columns. These columns could certainly help make the case for permanent index creation.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

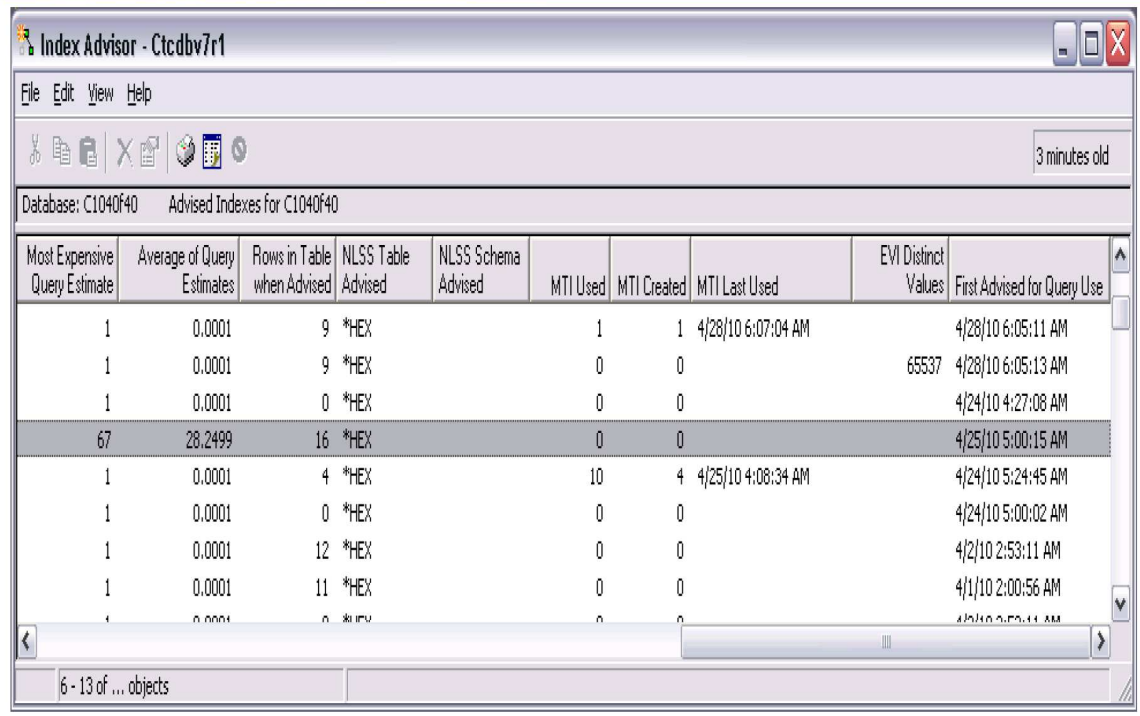
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Index Advisor: Scroll to the right



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-41. Index Advisor: Scroll to the right OD477.0

Notes:

MTI Used: The number of times that this specific Maintained Temporary Index has been used by the optimizer.

MTI Created: The number of times that this specific Maintained Temporary Index has been created by the optimizer. MTIs do not persist across system IPLs.

MTI Last Used: The timestamp representing the last time this specific Maintained Temporary Index (MTI) was used by the optimizer to improve the performance of a query. The MTI Last Used field can be blank, which indicates that an MTI which exactly matches this advice has never been used by the queries which generated this index advice.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

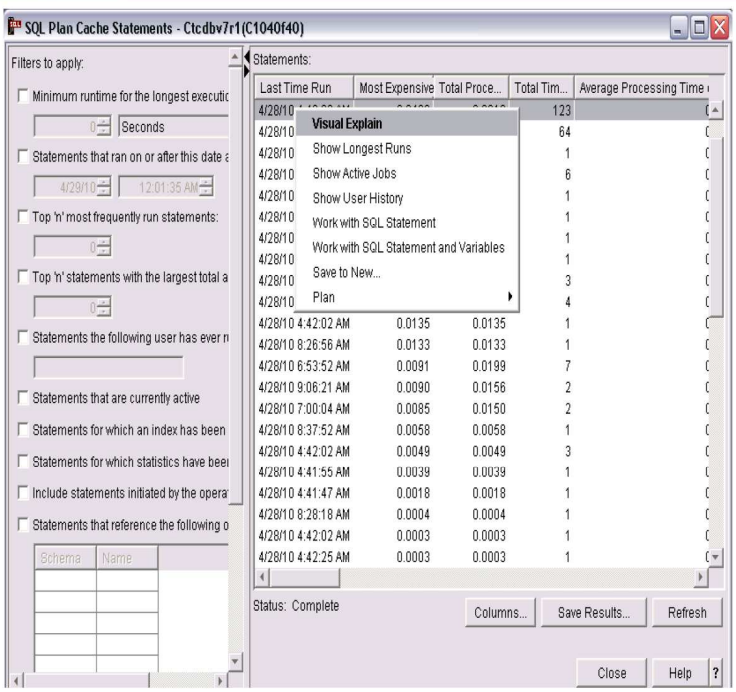
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Live analysis: SQL Plan Cache Statements

IBM i

Easy analysis of current SQL workloads …

- Without overhead of database monitor
- Advanced filtering to quickly narrow focus to the most interesting SQL
  - By number of executions
  - By DB2 object
- Easy access to Visual Explain for detailed analysis
- Archive capability into snapshot for future comparisons
- SQE only



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-42. Live analysis: SQL Plan Cache Statements OD477.0

Notes:

Right-click the database and select SQL Plan Cache to work with database requests made of the SQL query engine (SQE). The SQL performance monitor does not need to be started in order to view the SQL Plan Cache. This makes ad hoc analysis of your SQL queries easier.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

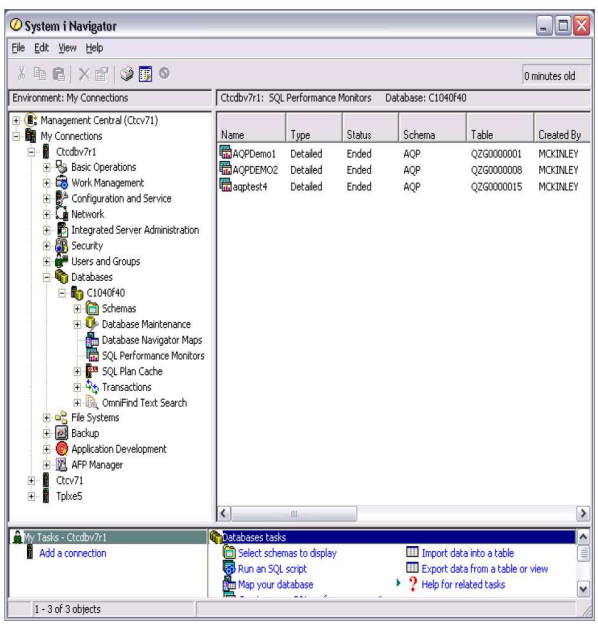
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL Performance Monitor toolset

IBM i

Have an existing application and want to capture the SQL activity and analyze it?

- Start/stop database monitor
  - Pre and post-filtering
  - Summary or detailed
  - Import data from other systems
- Areas to watch:
  - Index advised
  - Full opens vs. pseudo opens
  - Access plan rebuilds
  - Index creates
- Recommended during application tuning and problem determination
- Keep a baseline monitor run for comparison



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-43. SQL Performance Monitor toolset OD477.0

Notes:

To study a specific SQL application, the SQL Performance Monitor can be started prior to the application being executed.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

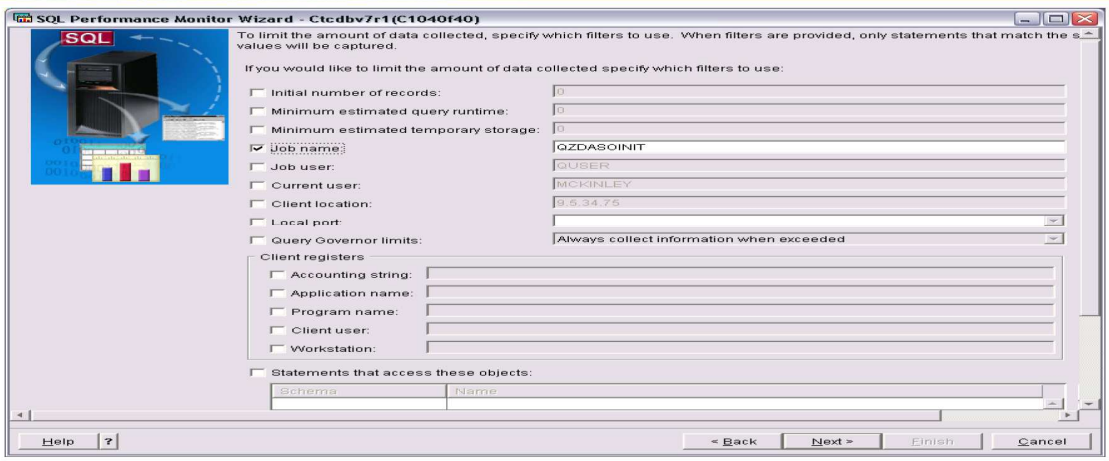
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL Performance Monitor filtering



- Minimize system-impact by reducing amount of data collected
  - Collect data for a specific user
  - Collect data for a collection of jobs (QZDASO*)
  - Collect data for longer-running SQL requests
- Enables quicker analysis and identification of SQL performance bottlenecks
- Post-filtering available for Visual Explain interface

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-44. SQL Performance Monitor filtering OD477.0

Notes:

Database monitors on DB2 for i have existed for many releases, but the usability of these monitors was significantly enhanced in V5R4 with the addition of both pre- and post-filters. These filters are similar to the filters that are available with the SQL Plan Cache viewer. Database monitor filters are important for several reasons. First, the filters can minimize the overhead and disk space consumed by a database monitor collection. For example, applying one of the filters such as only collecting monitor data for a certain user or only for SQL statements referencing a particular table could significantly reduce the amount of data collected by DB2. These filters are made available when you start a new SQL Performance Monitor collection using the graphical interface or the `STRDBMON` CL command.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

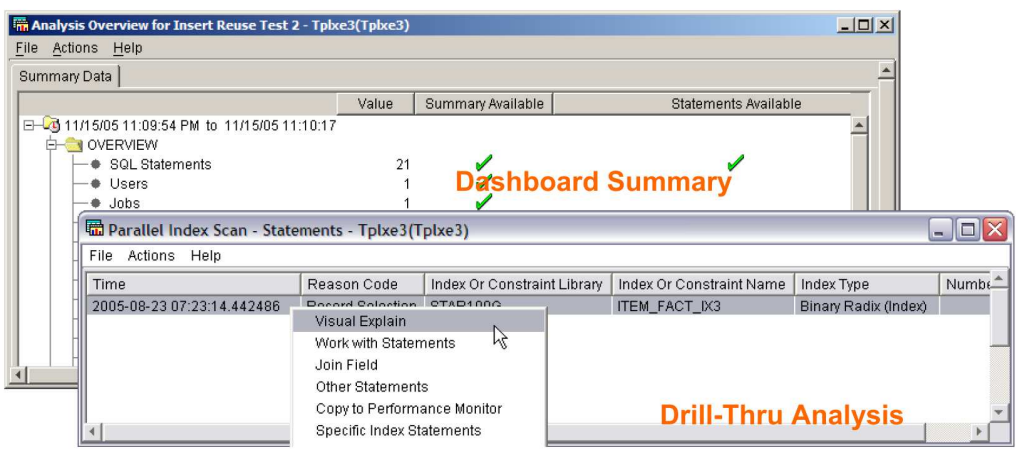
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

SQL Performance Monitor analysis

IBM i

- Summary view to more simply decide, if the monitor has attributes that require deeper analysis
- Integrated analysis with ability to launch other performance tools from current analysis view



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-45. SQL Performance Monitor analysis OD477.0

Notes:

The filters discussed on the previous chart eliminate the number of rows in a database monitor collection, but there are still hundreds of fields in a single database monitor row and many monitor row types that need to be sorted through to identify and fix performance problems. This maze of data made it a challenge to quickly look at a database monitor collection and determine if there were any issues: Unless you were a DB2 performance expert.

Thus, a dashboard summary was added in V5R4 to address this issue. On the first analysis of a database monitor collection, DB2 for i will automatically summarize some of the key performance indicators in the collection and then present the results in the summary window. This summary will allow an analyst to quickly scroll through some high-level DB2 performance indicators to determine if more detailed analysis is needed.

V5R4 also addresses another issue with the database monitor analysis reports by providing drill-through analysis. The reports could be used to find an SQL statement with performance issues. However, if more detailed research of this statement was needed with a tool such as Visual Explain, the user had to exit out of the tool and then manually copy

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

that SQL statement for Visual Explain analysis. As you can see with the bottom figure, an analyst can easily just right-click to drill into more detail with Visual Explain. This is a great enhancement for improving the efficiency of your database analysts and administrators.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

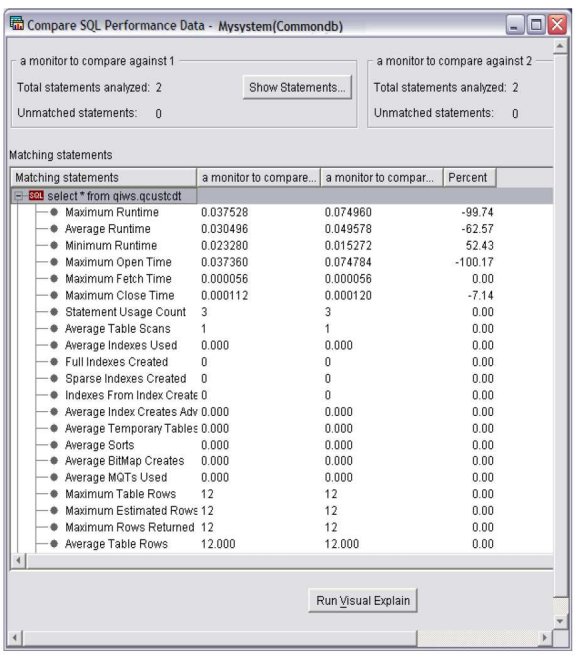
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

SQL Performance Monitor comparison

IBM i

- Want to know if an application change, system change, or PTF altered your DB2 performance?
  - Compare current SQL performance *profile* with archived monitor data
  - See if the optimizer's implementation of an SQL statement has changed
  - Can be long-running operation for large database monitor collections
  - Can do high level compare of two or more monitors



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-46. SQL Performance Monitor comparison OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

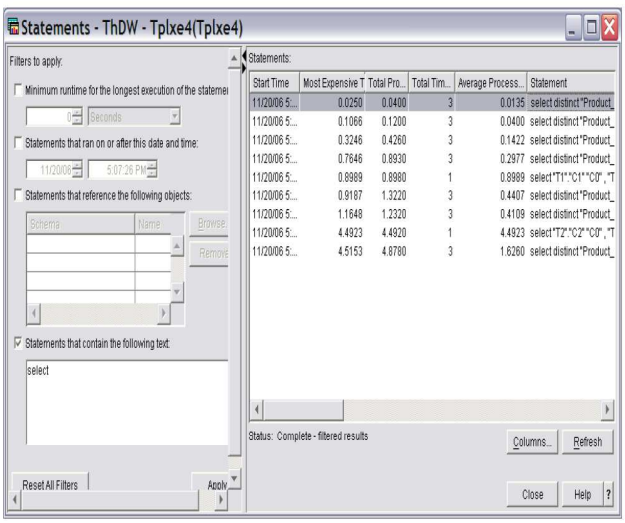
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Analyze SQL Performance Monitors with Visual Explain

IBM i

- Select existing detailed SQL Performance Monitor
- Show list of SQL statements captured by the monitor
- Can import monitor data from other systems
- Can narrow down analysis with post-filters



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-47. Analyze SQL Performance Monitors with Visual Explain OD477.0

Notes:

You can even analyze with Visual Explain the SQL samples captured with the Performance Monitors.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

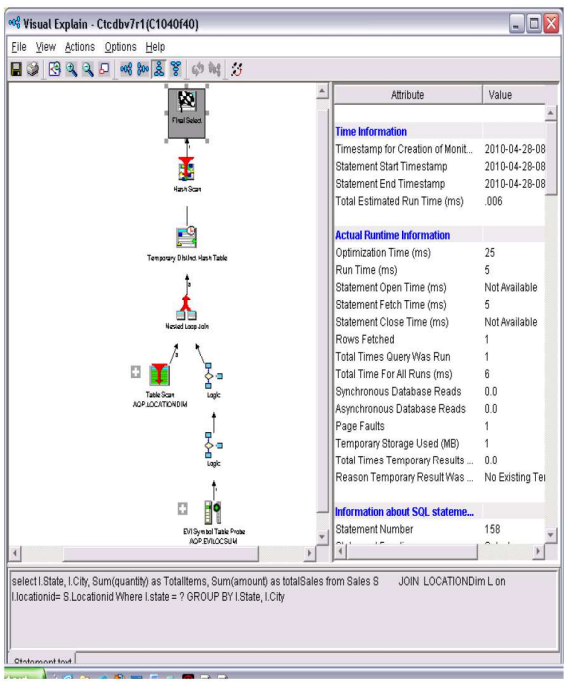
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Visual Explain

IBM i

- Active window: Look at DB objects (indexes, constraints, and so on) involved
- Attributes show all the settings that impact query optimizer
  - Labels include optimizer's estimates
  - SQE plans show decomposed SQL statement for each operation
- Icon highlighting to speed analysis
  - Expensive nodes
  - Index Advised nodes
- Debug messages help bridge to picture



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-48. Visual Explain OD477.0

Notes:

Visual Explain's active window shows the database objects (indexes, constraints, and so forth.) that are involved as well as their key attributes and settings that might impact the query optimizer.

The SQL for each statement is analyzed and reported upon, including highlighted Advisor suggestions that might speed the process.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

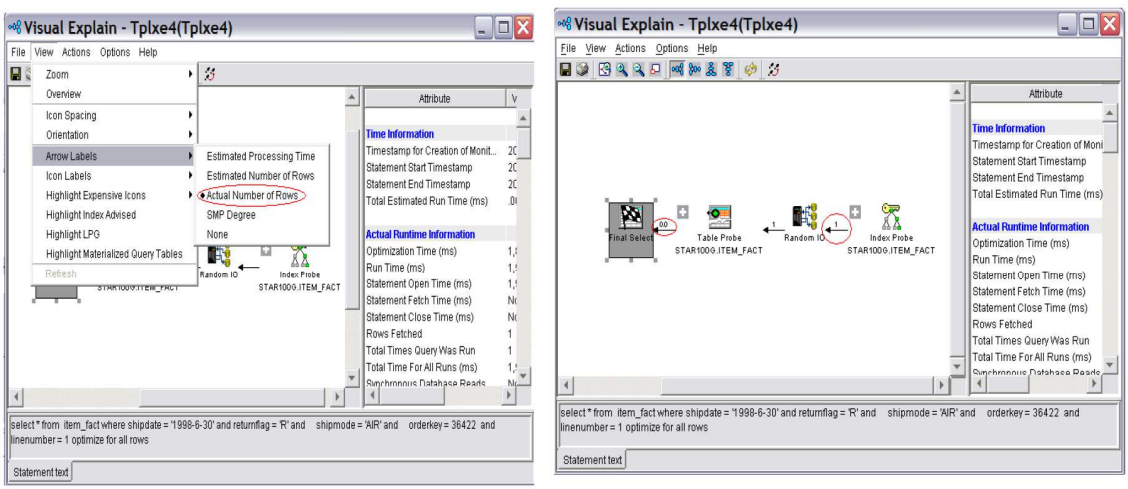
Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Visual Explain while running

IBM i

- Allows you to see actual runtime feedback in Visual Explain (VE) diagram

THIS IS VERY COOL!



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-49. Visual Explain while running OD477.0

Notes:

Visual Explain even works while jobs are running.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

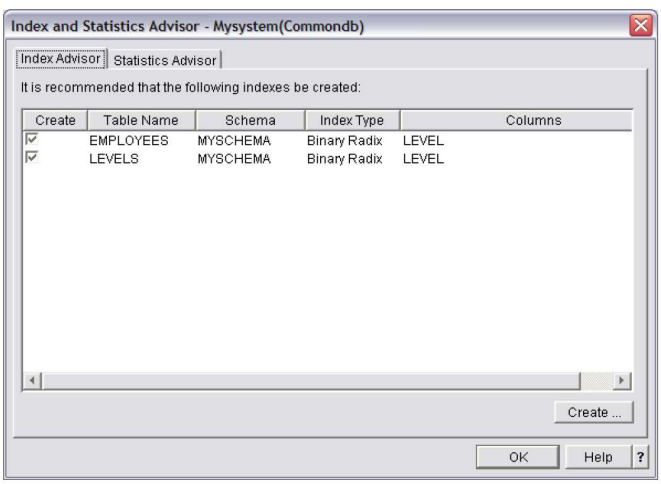
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Index Advisor

IBM i

- Advisor shows indexes that would help specific query (if any)
- Use advice along with Index Analyzer for the table to determine if index would be helpful
- The Create button prompts the Create Index dialog with recommended columns prefilled



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-50. Index Advisor OD477.0

Notes:

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

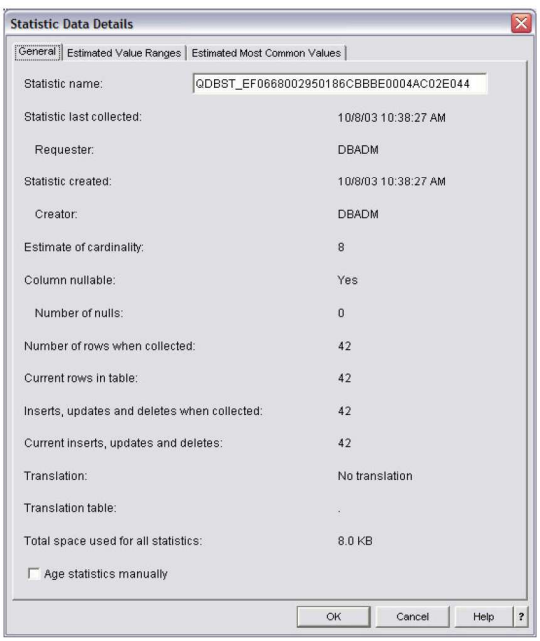
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Statistics

IBM i

- Advisor in Visual Explain shows statistics that would help specific query
- How do I run statistics?
  - It is simple! Statistics are automatically created/maintained by database



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-51. Statistics OD477.0

Notes:

The Advisor uses statistics that are automatically collected by the database manager to make its recommendations.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

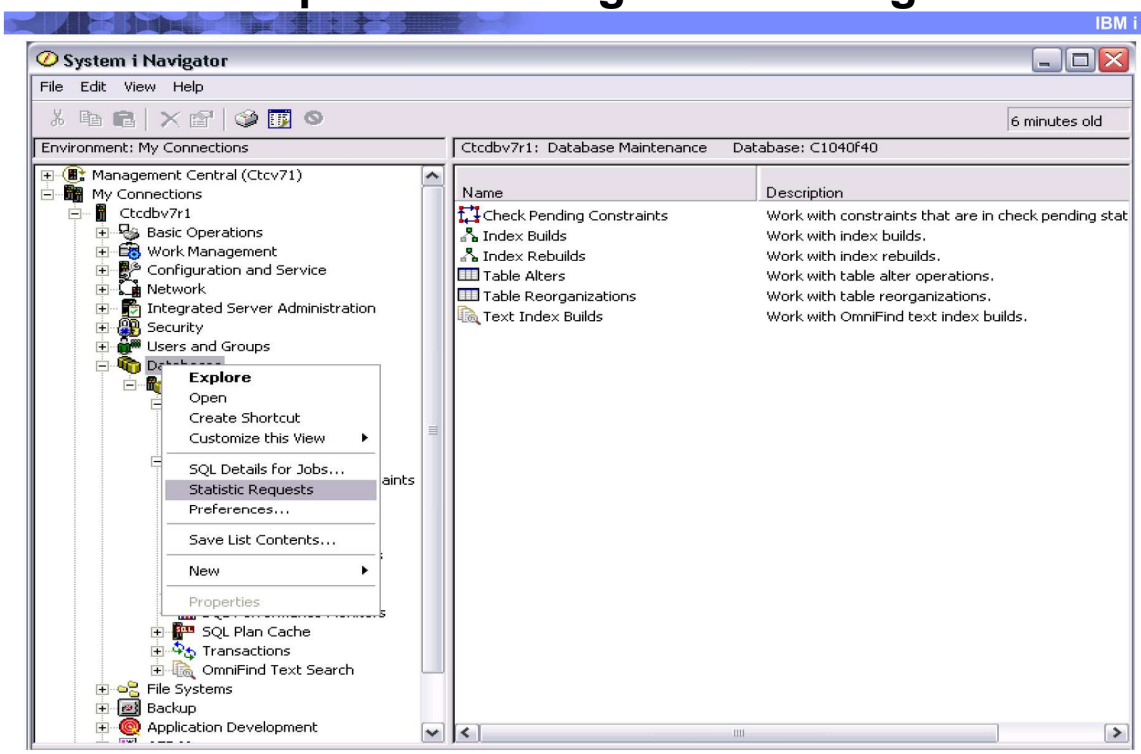
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Statistics requests running in the background



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-52. Statistics requests running in the background OD477.0

Notes:

A right-click of the Database folder and the selection of Statistic Requests will display a screen of statistic requests (next visual) that can be explored in more detail.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

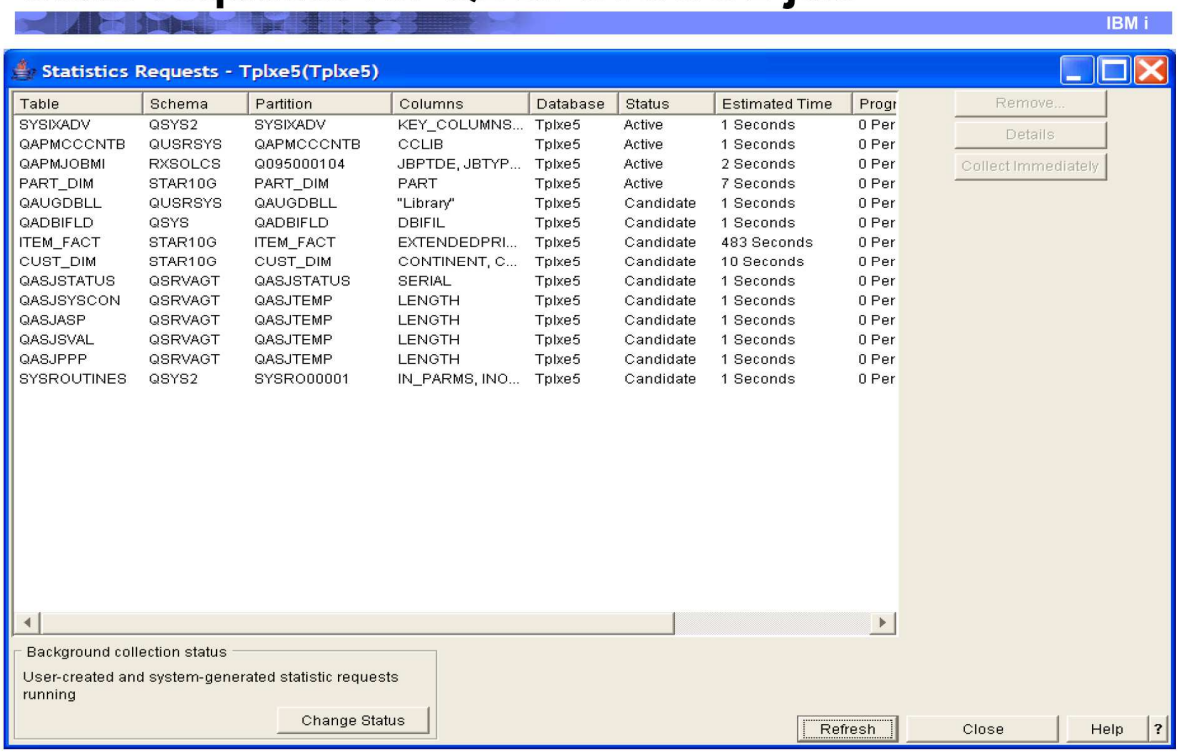
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Stats requests for QDBFSTCCOL job



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-53. Stats requests for QDBFSTCCOL job OD477.0

Notes:

Here is an example of the statistical requests display produced from the previous visual. Use the Statistics Requests dialog to control the system-wide statistics collection processes. Both user-created and system-generated statistics collections are available. With this dialog, you can:

- Stop statistics from being collected for a table
- Display statistics request details and view the error data for a request
- Request that statistics for a table be collected immediately
- View and change the background collection status for all background requests
- Refresh the list of statistics requests

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

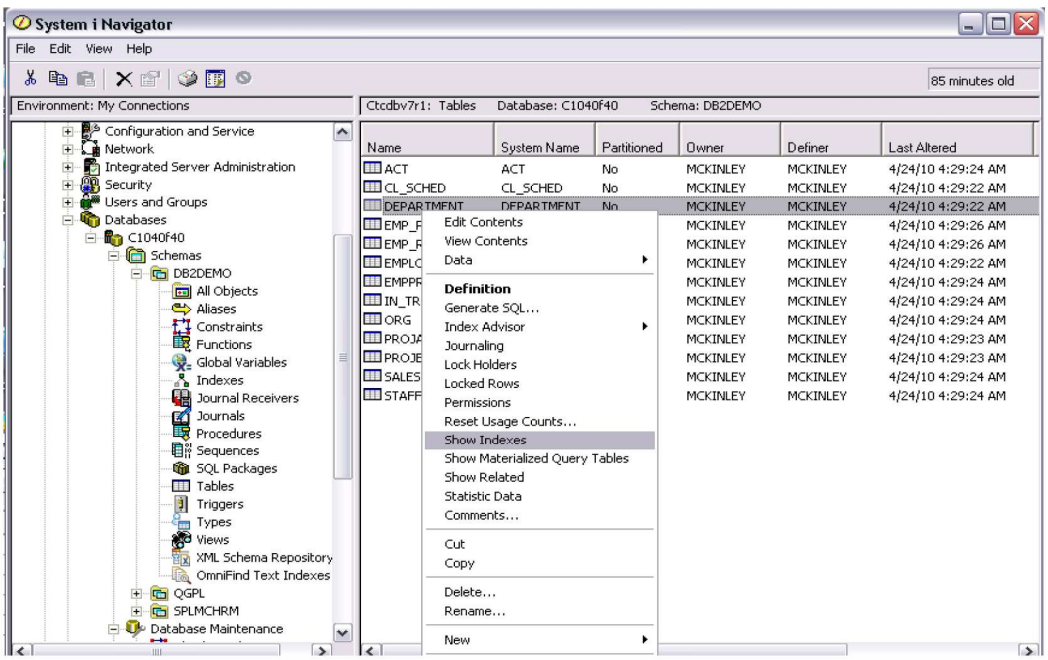
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Show indexes

IBM i



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-54. Show indexes OD477.0

Notes:

To further investigate the index situation for a given object, right-click the object and select **Show Indexes**. This information coupled with Index Advisor information could help you decide whether or not to follow an Advisor recommendation. Note that one of the many choices in the dialog window is **Index Advisor**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

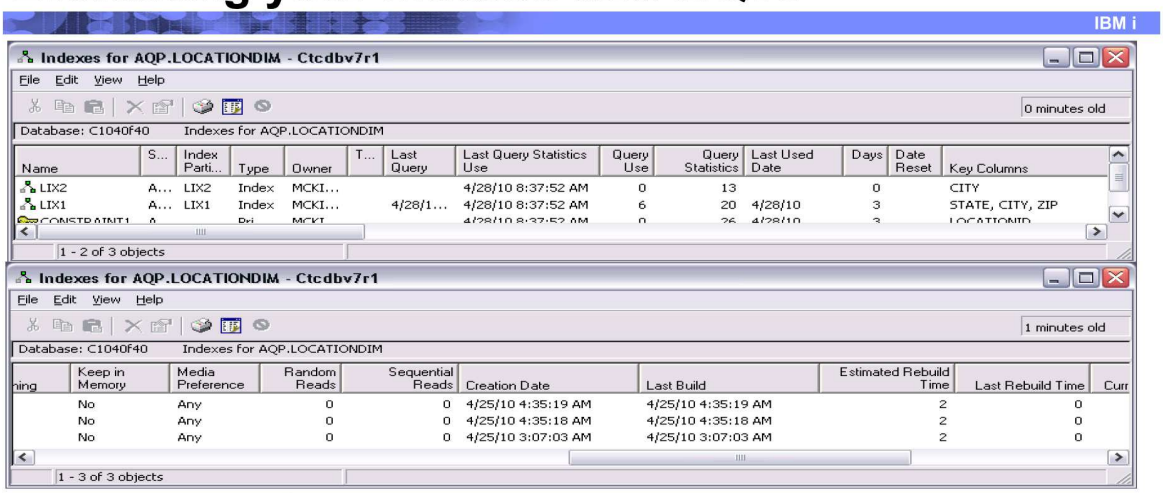
Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Evaluating your indexes and MQTs



Is an index or MQT providing value?

- Launch Evaluators by right-clicking on table and select **Show Index** or **MQT**.
- Scroll right to find the interesting query optimizer usage statistics.
  - Stats updated for all query interfaces: MQTs are SQL-only
  - Includes all objects the optimizer evaluates whether created by DDS or SQL
- Index Evaluator available on V5R2 and V5R3 using PTF: Statistics not updated until the PTFs are applied

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-55. Evaluating your indexes and MQTs OD477.0

Notes:

Prior to V5R3 there was no easy way to determine if there were any unnecessary indexes over a physical file. Also, there was no documentation or statistics given to the query optimizer of when an index was used by a query. Users would delete indexes based on the Last Used Date and erroneously delete indexes that had not been used in a while. The absence of the indexes would cause performance issues because the optimizer no longer had the necessary index statistics to aid it in making a good decision.

At V5R3 there was a new feature added to V5R3 System i Navigator and V5R3 OS/400 to provide statistics on index usage. To activate the feature, PTFs for APAR SE14709 must be applied along with iSeries Access Service Pack 2 for V5R3 as well as an additional PTF, SI14782.

Details on the new statistic fields follow:

1. There are two counters, one for when the index is used in a query implementation and one for when the index is used to gather statistics.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

2. Both counters are set by both query engines, the SQL Query Engine (SQE) and the Classic Query Engine (CQE). So, the statistics fields will be updated regardless of which query interface is used. It makes no difference whether the query is initiated by Query/400, SQL, OPNQRYF, or the QQQQRY API.
3. The statistics are persistent across IPLs.
4. Save/Restore does *not* reset the statistics of an index**,** *even if the index is restored over an existing index. If an index is restored that does not exist on the system, the statistics are reset.*
5. The statistics start accumulating after the PTFs are applied and active.
6. For each counter there is a corresponding timestamp recording the last time the counter was changed.
7. This data is stored internally in the index object. At this time, there is no way to query this, so in order to determine usage of an index over a specific timeframe, you must look at each timestamp on each individual index.

The new statistics are Query statistics only. Native opens from RPG and COBOL, that are NOT queries are not covered by the new statistics fields. However, we have had native open usage statistics for many years, such that a user could find out whether an index was used through these non-query interfaces by looking at existing statistics, such as Object Last Used.

==========

V5R3 and V5R2 Details:

Requires V5R3 Service Pack 2 for iSeries Access for V5R3 i5/OS and Service Pack 4 for V5R2 i5/OS

Requires DB group PTF SF99503 for V5R3 i5/OS and PTF SI16313 for V5R2 i5/OS

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

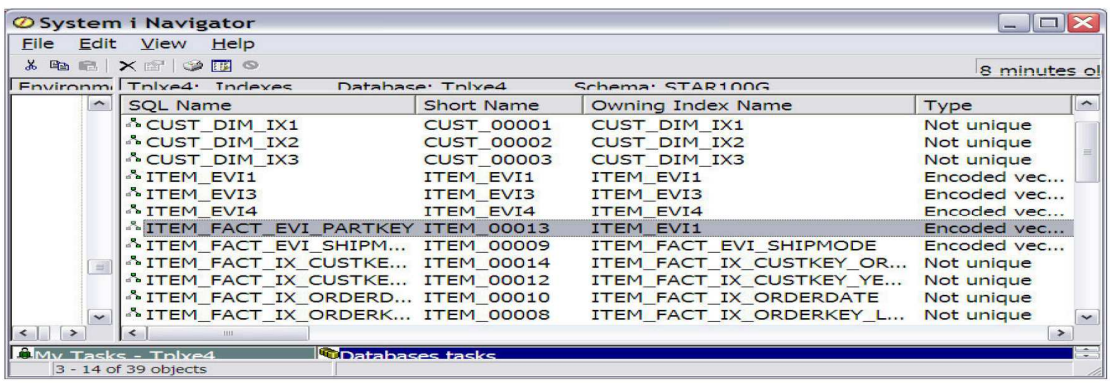
Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Enhanced index management

IBM i



From Index Folder you can now:

- See Owning Index Name
- See/Reset Query Usage Stats
- Show statements from plan cache that use this index

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-56. Enhanced index management OD477.0

Notes:

By clicking the Index folder, usage information and statistics can help you evaluate the benefit of given indexes. Even statements that used the index from the plan cache can be viewed. This could certainly give you an idea of how many queries used a particular index.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

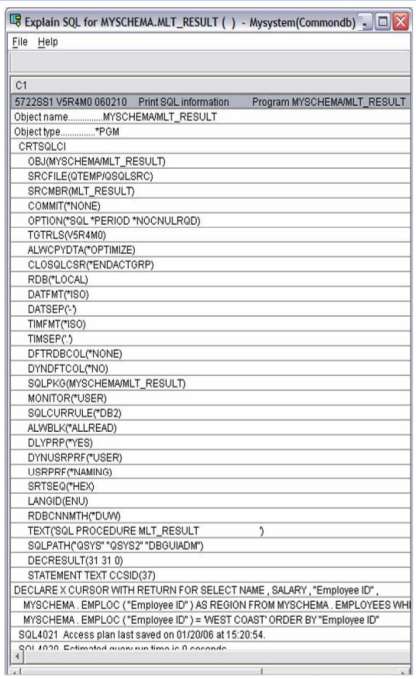
© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Analyze performance of SQL programs and packages

IBM i

- Explain SQL is equivalent to PRTSQLINF CL command
- Can explain SQL for functions, procedures, triggers, and packages
- Can explain SQL for programs by going down the File System folder
- How do I rebind my application packages?
  - It is simple! The database does it for you automatically.



© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-57. Analyze performance of SQL programs and packages OD477.0

Notes:

System i Navigator allows you to probe the details of SQL attributes for functions, procedures, triggers and packages. Click the appropriate folder, that is, **Procedures**. From the listed procedures, right-click the procedure of choice and then select, **Explain SQL**.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

For an ILO (Instructor Lead On-Line) class: In place of this checkpoint visual you should play file **OLD477U11** checkpoint:

a. To access the multimedia library click the **CD** button along the toolbar in Elluminate.

b. Once the multimedia library window is open, select **OD477U11** and click play.

c. Ask the students to indicate using green checkmark when they have finished the file.

For a FTF (Face to Face) class: Play the file **OD477U11** and ask the students to call out the answers to the questions on the screen.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Checkpoint

IBM i

1. True or False: System Director Navigator for i utilizes a web browser to interface with the IBM i.

2. True or False: An SQL statement can be regenerated from both DDS and SQL created objects.

3. True or False: Object authorities can be changed using System i Navigator.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-58. Checkpoint OD477.0

Notes:

Write your answers here:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Checkpoint solutions

IBM i

1. True or False: System Director Navigator for i utilizes a web browser to interface with the IBM i.

   The answer is True.

2. True or False: An SQL statement can be regenerated from both DDS and SQL created objects.

   The answer is True.

3. True or False: Object authorities can be changed using System i Navigator.

   The answer is True.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Additional information —

Transition statement —

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit summary

IBM i

Having completed this unit, you should be able to:

- List the System i Navigator database tools
- Use the System i Navigator database tools

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Figure 11-59. Unit summary OD477.0

Notes:

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Instructor notes:

Purpose —

Details —

Additional information —

Transition statement —

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Appendix A. Checkpoint solutions

Unit 1, "Introduction to relational database management and DB2 for i"

Solutions for Figure 1-35, "Checkpoint," on page 1-72

Checkpoint solutions

IBM i

1. True or False: All data items in a relational database are stored in base tables.

   The answer is True.

2. Which of the following are relational operators?

   a. Selection
   b. Projection
   c. Joins
   d. All of the above

   The answer is All of the above.

3. True or False: The IBM i implementation of a relational database has a performance advantage over the traditional implementation in that it is an integrated part of the operating system.

   The answer is True.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 2, "Introducing Structured Query Language"

Solutions for Figure 2-8, "Checkpoint," on page 2-27

Checkpoint solutions

IBM i

1. True or False: An IBM i logical file is known as a view in SQL terminology.

   The answer is True.

2. True or False: SQL has always been known as Structured Query Language.

   The answer is False. It was initially known as SEQUEL, Structured English Query Language. A copyright conflict was discovered in the late 1970s and the name was changed to SQL at that time.

3. True or False: The interactive SQL facility (invoked by the command STRSQL) is included as part of the base operating system.

   The answer is False. No, you must purchase the DB2 Query Manager for i and SQL Development Kit. It is bundled in certain enterprise packages.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 3, "Extracting data using SELECT"

Solutions for Figure 3-93, "Checkpoint (1 of 2)," on page 3-215

Checkpoint solutions (1 of 2)

IBM i

1. Place the following clauses in the order in which they must be coded with a SELECT statement.

   _4_ HAVING
   _2_ WHERE
   _3_ GROUP BY
   _1_ FROM
   _5_ ORDER BY

   The answers are 1 – FROM, 2 – WHERE, 3 – GROUP BY, 4 – HAVING, 5 – ORDER BY.

2. True or False: In order to SELECT all columns from a table, you can code SELECT * or SELECT ALL.

   The answer is False. You can code SELECT *. Also, you can code each individual field. If you want to resequence the selected columns, you must code them individually.

3. True or False: When ORDER BY specifies multiple columns, you must remember that the result set is sequenced by the last column, within the next to last column, and so on.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Solutions for Figure 3-94, "Checkpoint (2 of 2)," on page 3-217

Checkpoint solutions (2 of 2)

IBM i

4. When coding a condition that includes a character literal, what should you be careful to do when coding the value?

   The answer is Enclose the value in quotes and be aware of case.

5. Which lines of code below are valid?

   a. WHERE quantity > 1000 and quantity < = 10000
   b. Where (quantity > = 1000) and (season = ‘Christmas’)
   c. WHERE (year < = 2000 and > = 1998)

   The answers are WHERE quantity > 1000 and quantity < = 10000 and Where (quantity > = 1000) and (season = ‘Christmas’).

6. Code the SQL WHERE clause that will limit the selection of rows to those columns of the name 'COUNTRY' that have the string 'Amer' anywhere in the value.

   The answer is WHERE COUNTRY LIKE ‘%Amer%’.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 4, "Functions"

Solutions for Figure 4-36, "Checkpoint (1 of 2)," on page 4-85

Checkpoint solutions (1 of 2)

IBM i

1. True or False. The MIN function is identical to the MINUTE function.
   The answer is False.

2. True or False. A COLUMN function can be an argument of a SCALAR function.
   The answer is True.

3. The EMPLOYEE table includes a column, EDUCLVL, that indicates the employee's level of education. The number of years of education is used in the EMPLOYEE table to give the education level. A CASE expression can be used to group these and to show the level of education. If the value of EDUCLVL is 22, what will be returned by the following CASE expression?

```
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
  CASE
  WHEN EDUCLVL < 15 THEN 'High School'
  WHEN EDUCLVL < 19 THEN 'University Graduate'
  WHEN EDUCLVL < 23 THEN 'Masters Degree'
  ELSE 'PhD'
  END
FROM EMPLOYEE
```

   a. High School
   b. University Graduate
   c. Masters Degree
   d. PhD
   e. All of the above.

   The answer is Masters Degree.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Solutions for Figure 4-37, "Checkpoint (2 of 2)," on page 4-87

Checkpoint solutions (2 of 2)

IBM i

4. True or False: The function NOW returns a timestamp value.
   The answer is True.

5. True or False: The CAST function can perform the same function as many of the other conversion functions.
   The answer is True.

6. True or False: CASE expressions are used in the SELECT statement WHERE, GROUP BY and HAVING clauses.
   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 5, "JOIN and UNION"

Solutions for Figure 5-23, "Checkpoint," on page 5-61

Checkpoint solutions

IBM i

1. Assume you want to list all members and their cars plus you want to list the members without cars. What are you doing?

   The answer is LEFT OUTER JOIN.

2. True or False: When coding a join, coding a condition is optional. If you do not code a condition, all rows will be selected.

   The answer is False. All rows from the first table will be combined with all rows from the second table.

3. We want to make sure that we have been maintaining the CARS table correctly. We want to list all cars in the CARS table that are not associated with a member in the MEMBERS table. Is this possible with a JOIN?

   The answer is Yes, an exception JOIN.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 6, "SubSELECTs"

Solutions for Figure 6-23, "Checkpoint," on page 6-56

Checkpoint solutions

IBM i

1. List three quantified predicates that can be used with subqueries.

   The answers are ALL, ANY, and SOME.

2. You have the requirement to list all employees with the highest salaries in each of their respective work departments. What kind of subquery can you use?

   The answer is Correlated subquery.

3. True or False: A subselect may be presented as a column of the result set.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 7, "Query tools"

Solutions for Figure 7-27, "Checkpoint," on page 7-68

Checkpoint solutions

IBM i

1. True or False: Query Manager is part of IBM i/OS.

   The answer is False.

2. True or False: Query Manager queries are SQL-based.

   The answer is True.

3. True or False: Query Manager output can be directed to the display, printer, or to another database table.

   The answer is True.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 8, "Database maintenance"

Solutions for Figure 8-37, "Checkpoint," on page 8-87

Checkpoint solutions

IBM i

1. True or False: Data Definition Language includes the CREATE, ALTER, and DROP statements of SQL.

   The answer is True. DDL includes CREATE, ALTER, and DROP.

2. A schema:
   a. Is an IBM i library
   b. Includes an SQL catalog and views of that catalog
   c. Includes journals and receivers for tables
   d. All of the above

   The answer is All of the above.

3. True or False: Column names can be no more than 10 characters in length.

   The answer is False. Column names can be up to 30 characters in length.

© Copyright IBM Corporation 1999, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 9, "Using embedded SQL"

Solutions for Figure 9-81, "Checkpoint," on page 9-199

Checkpoint solutions

IBM i

1. True or False: All host variables used in an SQL statements must be declared.
   The answer is True.

2. True or False: To embed SQL into an RPG IV program, the member type must be SQLRPGLE.
   The answer is True.

3. True or False: The SQL Communications Area is required in any program that uses embedded SQL.
   The answer is False.

4. True or False: Embedded SQL in RPG IV programs cannot process multiple records in a single I/O operation.
   The answer is False.

5. True or False: A serial cursor allows for more than sequential reading of rows in a table.
   The answer is False.

6. True or False: Dynamic non-SELECT embedded statements may or may not use parameter markers.
   The answer is True.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 10, "An overview of stored procedures"

Solutions for Figure 10-29, "Checkpoint," on page 10-73

Checkpoint solutions

IBM i

1. True or False: Once a program has been registered as a stored procedure, you can call it from a PC client using ODBC or Java.

   The answer is True.

2. Which of the following are types of stored procedures?
   a. Embedded procedure
   b. External procedure
   c. SQL stored procedure
   d. All of the above

   The answers are External procedure and SQL stored procedure.

3. An SQL CALL command
   a. Targets an external stored procedure
   b. Targets an SQL stored procedure
   c. Targets an RPG IV program with embedded SQL
   d. All of the above

   The answers are Targets an external stored procedure and Targets an SQL stored procedure.

© Copyright IBM Corporation 2000, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Unit 11, " Database tools with System i Navigator"

Solutions for Figure 11-58, "Checkpoint," on page 11-121

Checkpoint solutions

IBM i

1. True or False: System Director Navigator for i utilizes a web browser to interface with the IBM i.

   The answer is True.

2. True or False: An SQL statement can be regenerated from both DDS and SQL created objects.

   The answer is True.

3. True or False: Object authorities can be changed using System i Navigator.

   The answer is True.

© Copyright IBM Corporation 2010, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Appendix B. SQL terms

Access plan An access plan is a set of internal structures and information that tells SQL how to run an embedded SQL statement most effectively.

Catalog The database manager maintains a set of tables containing information about the data in the database. These tables are collectively known as the catalog. The catalog tables contain information about tables, parameters, procedures, packages, views, indexes, and constraints on the i system.

Check constraints A check constraint is a rule that specifies the values allowed in one or more columns of every row of a table. Check constraints are optional and can be defined using the SQL statements CREATE TABLE and ALTER TABLE.

Child table Also called a dependent table.

Collection The objects in a relational database are organized into sets called collections. A collection provides a logical classification of objects in the database. A collection is also called a schema. When a table, view, or index is created, it is assigned to exactly one collection.

Column Part of a table that has a name and a specific datatype. Also referred to as a field.

Column functions An operation on a column. The argument of a column function is a set of values. An argument of a scalar function is a single value. The following column functions are supported on the i: AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE, or VAR.

Composite index An index that is composed of two or more columns.

Composite key A composite key is an ordered set of columns of the same table.

Condition Search criteria in a query's WHERE clause that evaluates to TRUE or FALSE.

Constants A value that does not change.

Constraints Restrictions on data that are enforced at the data level.

Cursor A work area in memory where the current SQL statement is stored.

Datatype Defines data as a specific type such as character, date, or number.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Dataspace	An internal object that contains the data in a table. It is the same thing as table, physical file, or data repository.
DDL	Data Definition Language. Examples of DDL statements are CREATE, ALTER, and DROP.
Distinct	Unique. Used in the SELECT statement to return unique values, no duplicates.
DML	Data Manipulation Language. Examples of DML statements are INSERT, UPDATE, and DELETE.
Dynamic SQL	An SQL statement prepared during the execution of an SQL application. SQL statements embedded in a REXX application are dynamic SQL statements. SQL statements submitted to the interactive SQL facility are also dynamic SQL statements.
Embedded SQL	SQL code embedded within other programs written in languages such as COBOL, RPG, and so forth.
Field	A column in a table designed to maintain specific information about every record in the table.
Foreign key	One or more columns whose values are based on the primary key column values in another table.
Functions	A function is an operation denoted by a function name followed by one or more operands which are enclosed in parentheses. The operands of functions are called arguments. Most functions have a single argument that is specified by an expression. The result of a function is a single value derived by applying the function to the result of the expression. Functions are classified as scalar functions or column functions. The argument of a column function is a set of values. An argument of a scalar function is a single value.
Index	An index is an ordered set of pointers to rows of a base table. Each index is based on the values of data in one or more table columns. An index is an object that is separate from the data in the table. When you request an index, the database manager builds this structure and maintains it automatically. Indexes speed up data retrieval.
Interactive SQL	An interactive SQL facility is associated with every database manager. Essentially, every interactive SQL facility is an SQL application program that reads statements from a terminal, prepares and executes them dynamically, and displays the results to the user. The interactive facilities for DB2 for i are invoked by the STRSQL command or the STRQM command.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Join	Combines data from two or more tables into a result table by linking columns. Used in the WHERE clause of a SQL statement. There are four different types of joins is supported by DB2 for i. An inner join returns only the rows from each table that have matching values in the join columns. Any rows that do not have a match between the tables will not appear in the result table. A left outer join returns values for all of the rows from the first table and the values from the second table for the rows that match. Any rows that do not have a match in the second table will return the null value for all columns from the second table. An exception join returns only the rows from the left table that do not have a match in the right table. Columns in the result table that come from the right table have the null value. A cross join returns a row in the result table for each combination of rows from the tables being joined (a Cartesian Product).
Key	A column or columns that identify rows of a table. See primary key and foreign key for additional details.
Null	A value that is unknown.
Operator	A reserved work or symbol used to perform an operation. Examples would include arithmetic operators (addition, subtraction, and so forth), logical operators (BETWEEN, LIKE, IN, and so forth), or comparison operators (>,<, =, and so forth).
Packages	For distributed SQL programs, a package is an object that contains control structures used to execute SQL statements. Packages are produced during program preparation. The control structures can be thought of as the bound or operational form of SQL statements. All control structures in a package are derived from the SQL statements embedded in a single source program.
Parent/Child	A concept utilized in relational databases. Often also called the master/dependent relationship. The 'parent' is the higher level or master table in the database. A 'child' or dependent is a table that is tied to the master table through the use of a foreign key. Under this concept, a parent table can have multiple child tables, but child tables can only have one parent table. Used to ensure referential integrity.
Predicates	A predicate specifies a condition that is true, false, or unknown about a given row or group. The statements BETWEEN, EXISTS, IN, LIKE, and NULL and symbols such as =, <, <=, >, >=, <> are examples of predicates.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Primary key	A column that makes each row of data in the table unique. A table cannot have more than one primary key.
Query	An SQL statement that is used to retrieve data from a database. Queries are case sensitive.
Referential integrity	Assures that values from one column depend on the values from another column. The state of a database in which all values of all foreign keys are valid.
Row	Set of records in a table.
Scalar function	The argument of a column function is a set of values. An argument of a scalar function is applied to a single value rather than a set of values. Examples include: ABS or ABSVAL, ACOS, ANTILOG, ASIN, ATAN, ATANH, CHAR, CHARACTER_LENGTH or CHAR_LENGTH, COALESCE, CONCAT, COS, COSH, COT, CURDATE, CURTIME, DATE, DAY, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, DAYS, DECIMAL, DEGREES, DIGITS, DOUBLE_PRECISION or DOUBLE, EXP, FLOAT, FLOOR, HASH, HEX, HOUR, IFNULL, INTEGER or INT, LAND, LEFT, LENGTH, LN, LNOT, LOCATE, LOG or LOG10, LOR, LTRIM, MAX, MICROSECOND, MIN, MINUTE, MOD, MONTH, NODENAME, NODENUMBER, NOW, NULLIF, PARTITION, POSITION or POSSTR, POWER, QUARTER, REAL, RRN, RTRIM, SECOND, SIN, SINH, SMALLINT, SQRT, STRIP, SUBSTRING or SUBSTR, TAN, TANH, TIME, TIMESTAMP, TRANSLATE, TRIM, UCASE or UPPER, VALUE, VARCHAR, VARGRAPHIC, WEEK, XOR, YEAR, and ZONED.
Schema	A collection of related objects in a database.
SELECT	The basic statement used to extract information from databases.
Static SQL	An SQL statement embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program. A source program containing static SQL statements must be processed by an SQL precompiler before it is compiled. The precompiler checks the syntax of the SQL statements, turns them into host language comments, and generates host language statements to call the database manager.
Stored procedures	A group of one or more SQL statements or functions that are stored in the database and compiled and ready to be executed.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Subquery	A SELECT statement embedded within another SELECT statement. A nested select.
Subselect	A subselect specifies a result table derived from the tables or views identified in the FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation may be quite different from this description.)
Table	A table is an object that stores user data. Tables are logical structures maintained by the database manager. Tables are made up of columns and rows. At the intersection of every column and row is a specific data item called a value. A column is a set of values of the same type. A row is a sequence of values.
Triggers	A trigger is a set of actions that are executed automatically whenever a specified event occurs to a specified base table. A trigger can be defined to be executed before the event or after the event. The event can be an insert, update, or delete operation.
View	A database object that is created from one or more tables.

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Bibliography

Web URLs:

IBMs Internet Connection website:

http://www.ibm.com

Internet site for IBM Support Portal: Support Home

www.ibm.com/support/entry/portal/

Internet site for ITSO Redbooks:

http://www.redbooks.ibm.com

Internet “Requests for Comments” Editor:

http://www.isi.edu/rfc-editor/

IBM i and System i Information Center:

http://publib.boulder.ibm.com/eserver/ibmi.html

IBM i website:

http://www.ibm.com/systems/i/

Web PDF URLs:

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/db2/rbafz.pdf

DB2 for i SQL Reference

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/db2/rbafo.pdf

Database Programming

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzala/rzala.pdf

SQL Messages and Codes

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzamp/rzamp.pdf

IBM i Maximum Capacities, Database and SQL Limits

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzajp/rzajp.pdf

IBM i Database Embedded SQL Programming

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/cli/rzadp.pdf

SQL Call Level Interface

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

http://www.redbooks.ibm.com/abstracts/sg246503.html

Stored Procedures, Triggers, and User-Defined Functions on DB2 UDB for iSeries

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzakb/rzakbmst.pdf

Programming DDS for Physical and Logical Files

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzajp/rzajp.pdf

Getting to Know System i Navigator

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzamm/rzamm.pdf

IBM i Access for Web

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzatg/rzatg.pdf

System i Navigator Tasks on the Web

http://www.redbooks.ibm.com/abstracts/sg247789.html

IBM Systems Director Navigator for i

http://publib.boulder.ibm.com/infocenter/iseries/v6r1m0/topic/rzatc/sc415212.pdf

Query Manager Use

http://publib.boulder.ibm.com/infocenter/iseries/v6r1m0/topic/rzatc/sc237691.pdf

XML Extender Administration and Programming

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/topic/rzahx/rzahxtbls.pdf

IBM i Performance Reference Information

ITSO Redbooks:

SG24-5402 *Who Knew You Could Do That with RPG IV?*

SG24-4249 *DB2/400 Advanced Database Functions*

SG24-5485 *Developing Cross-Platform DB2 Stored Procedures*

SG24-6503 *Stored Procedures and Triggers on DB2 Universal Database for iSeries*

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

Other useful websites:

Certification:

http://www.ibm.com/certify

MidRange News for the IBM i Community:

http://www.midrangenews.com/

Books:

ISBN 1-55860-482-0 *A Complete Guide to DB2 Universal Database (Don Chamberlin)*

© Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.

 © Copyright IBM Corp. 2010, 2013

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.