

Using SQL in RPG Programs: An Introduction

OCEAN Technical Conference
Catch the Wave



Susan M. Gantner
susan.gantner @ partner400.com
www.partner400.com

Partner400
Your partner in AS/400 and iSeries Education

© Copyright Partner400, 2002.

Agenda

Partner400

What is SQL?

SQL language overview

- Accessing data using SQL
- Creating/maintaining databases using SQL

SQL on the AS/400

- Using interactive SQL
- Embedding SQL in programs
- Query Manager

What is SQL on the AS/400?

Partner400

An alternative database interface language

- NOT a database management system

High level, simple statement formats

A language used for:

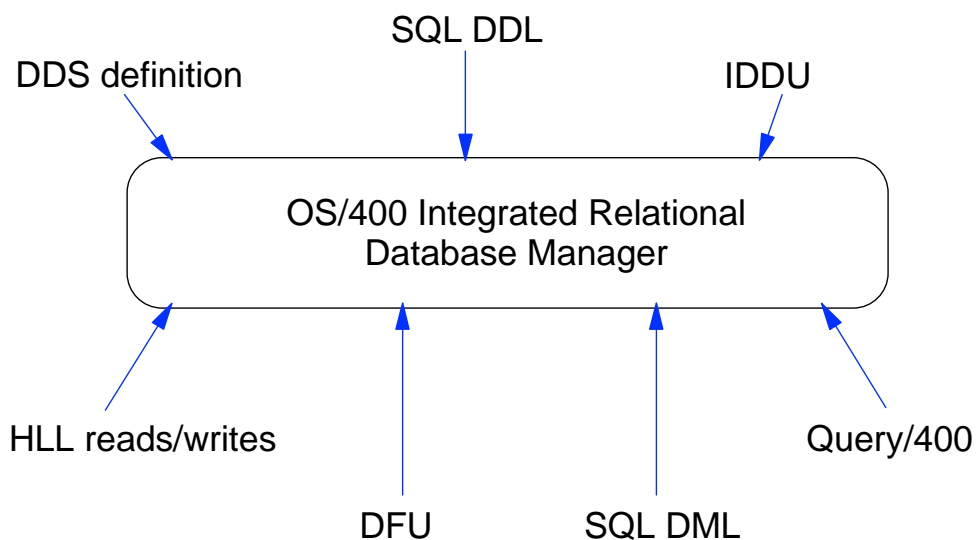
- Data Definition (DDL)
- Data Manipulation (DML)

Completely interchangeable data methods

- SQL tables may be accessed with native language
- DDS created files can be access with SQL

One Database Manager

Partner400



SQL Term	AS/400 Term
Table	File
Row	Record
Column	Field

Table

	Nbr	Name	Pos	Sex	Sal
Row →	10	AMY	2	F	1200
	35	JOE	5	M	1000
	30	JON	7	M	1500
	20	DON	5	M	1150
	25	ANN	8	F	1550

↑
Column

Data Manipulation Language

Basic statements

- SELECT - retrieves data; one row or multiple
- UPDATE - updates one row or multiple
- DELETE - deletes one row or multiple
- INSERT - adds one row or multiple

Environments

- Interactive SQL - Use STRSQL command
- Embedded SQL - Put into High Level Language (HLL)
- Query Manager - Report Formatter

Retrieving Data - the *SELECT* statement *Partner400*

SELECT - some column(s) or * or expression

FROM - some table(s)

WHERE - selection criteria

GROUP BY - some column(s)

HAVING - selection criteria for groups

ORDER BY - Presentation order (sort)

```
SELECT *  
FROM empl
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550

Retrieving Data - the *SELECT* statement *Partner400*

SELECT any number of columns in any order

or *, which means all columns

WHERE clause provides selection criteria

```
SELECT nbr, name  
FROM empl  
WHERE pos = 5
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Nbr	Name
35	JOE
20	DON

SELECT statement

Partner400

Keywords in the WHERE clause:

- Greater than (or =), Less than (or =), Equal
- Not greater, Not less, Not equal
- AND, OR, NOT
- Range - inclusive constant range (BETWEEN)
- Values - list of constant values (IN)
- Pattern matching (LIKE) with wild cards
 - % = any number of characters
 - _ = exactly 1 character

SELECT statement - Examples

Partner400

BETWEEN is inclusive of values listed

```
SELECT name, pos  
FROM empl  
WHERE pos BETWEEN 5 and 7
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Name	Pos
JOE	5
JON	7
DON	5

SELECT statement - Examples

Partner400

Note that you can resequence the column (field) names

```
SELECT name, nbr  
FROM empl  
WHERE name LIKE 'A%'
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Name	Nbr
AMY	10
ANN	25

SELECT statement - Examples

Partner400

More complex conditions, including calculations
Note that selections can be made on columns not selected

```
SELECT name, nbr, pos  
FROM empl  
WHERE sex = 'M' and (sal * 12) > 12000
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Name	Nbr	Pos
JON	30	7
DON	20	5

SELECT statement - Examples

Partner400

ORDER BY specifies row order

If not specified, order is unpredictable!

Not always physical order

```
SELECT name, nbr, pos
FROM empl
WHERE sex = 'M' and (sal * 12) > 12000
ORDER BY name
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Name	Nbr	Pos
DON	20	5
JON	30	7

SELECT statement - Examples

Partner400

Derived Columns can be created

If used for ordering, use relative position number

```
SELECT name, sal * 12
FROM empl
ORDER BY 2
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Name	Sal
JOE	12000
DON	13800
AMY	14400
JON	18000
ANN	18600

SELECT statement - Examples

Partner400

GROUP BY provides row summary
Built-in functions for grouping:
AVG, SUM, MAX, MIN COUNT

```
SELECT pos, AVG(sal)
FROM empl
GROUP BY pos
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Pos	AVG(Sal)
2	1200
5	1075
7	1500
8	1550

SELECT statement - Examples

Partner400

Selecting on Groups: HAVING

```
SELECT pos, AVG(sal)
FROM empl
GROUP BY pos
HAVING AVG(sal) > 1200
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Pos	AVG(Sal)
7	1500
8	1550

Retrieving Data from Multiple Tables

Partner400

Join: dynamic connection of selected columns from more than one table

JOB Table

Pos	Desc
2	Operator
5	Programmer
7	Manager
8	Analyst

EMPL Table

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



```
SELECT name, empl.pos, desc
FROM empl, job
WHERE empl.pos = job.pos
```

Name	Pos	Desc
AMY	2	Operator
JOE	5	Programmer
JON	7	Manager
DON	5	Programmer
ANN	8	Analyst

Changing Data in a Table

Partner400

SQL Statements

- UPDATE
- INSERT INTO (add a record)
- DELETE

Each can handle either

- One row at a time
- Multiple rows at a time

UPDATE Statement

Partner400

Give all programmers (pos = 5) a 10% raise!

```
UPDATE empl  
SET sal = sal + (sal * .10)  
WHERE pos = 5
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1100
30	JON	7	M	1500
20	DON	5	M	1265
25	ANN	8	F	1550

INSERT Statement

Partner400

Add new rows using INSERT

Column names and values in one-to-one correspondence

One row at a time using the VALUES clause:

```
INSERT INTO empl  
(name, nbr, pos, sal, sex)  
VALUES ('AMY', 10, 2, 1200, 'F')
```

Or multiple rows at a time using a SELECT statement:

```
INSERT INTO empl  
SELECT nbr, name, pos, sex, sal  
FROM emplnew  
WHERE pos = 9
```

DELETE Statement

Partner400

Rows can be deleted individually or by sets as well

```
DELETE
FROM empl
WHERE nbr = 10
```

Nbr	Name	Pos	Sex	Sal
10	AMY	2	F	1200
35	JOE	5	M	1000
30	JON	7	M	1500
20	DON	5	M	1150
25	ANN	8	F	1550



Nbr	Name	Pos	Sex	Sal
35	JOE	5	M	1100
30	JON	7	M	1500
20	DON	5	M	1265
25	ANN	8	F	1550

Database Management with SQL

Partner400

DDL - Data Definition Language

SQL database objects

- COLLECTION (AS/400 library object)
- TABLE (Physical file)
- VIEW (Logical file)
- INDEX (Logical file)

To create SQL database objects

- CREATE object_type object_name

To delete SQL database objects

- DROP object_type object_name

Creating Tables

Partner400

Tables are created as physical files

- Can be accessed same as any other PF
- With or without SQL

If created into an SQL collection, automatically journaled

Columns are null-capable by default

- Specify NOT NULL to mimic DDS behavior
- WITH DEFAULT supplies default value in new rows

```
CREATE TABLE empl  
  (nbr DEC(5,0) NOT NULL,  
   name CHAR(25) NOT NULL,  
   pos DEC(1,0) NOT NULL,  
   sex CHAR(1) NOT NULL,  
   sal DEC(7,2) NOT NULL WITH DEFAULT)
```

SQL Views

Partner400

Contain a selection of columns and/or rows from base table

- May be a subset of columns and/or rows
- May be a join view

Created as a logical file with NO key fields

Views of views are allowed

Views may be summaries (using GROUP BY)

```
CREATE VIEW richmen AS  
  SELECT name, sex, sal  
  FROM empl  
  WHERE sex = 'M' and (sal * 12) > 17000
```

SQL Indexes

Partner400

Creates a keyed logical file over table(s)

Used primarily to enhance performance

- Note: ALL keyed logical files may be used to improve performance
 - Even if not created as SQL indexes

Must contain key field(s)

- May be ascending or descending

May be specified as UNIQUE

```
CREATE INDEX empnbr  
ON empl (nbr)  
  
CREATE INDEX empindx  
ON empl (pos DESC, nbr)
```

Using SQL on the AS/400

Partner400

DB2/400 Query Manager and SQL Development Kit (or QM & SDK)

- Interactive SQL interface
- Pre-compilers for embedding SQL in programs
- Query Manager for generating reports

OS/400 contains:

- Run-time support for SQL and Query Manager
- QM & SDK not required to run SQL applications or pre-created QM queries

A tool for programmers and database administrators

Interactive functions

- Quickly maintain database
- Test SQL code before embedding
- Create test data scenarios

STRSQL to begin

Enter SQL Statements

Type SQL statement, press enter.

==>SELECT _____

F3=Exit **F4=Prompt**

F10=Copy F13=Service

F24=More keys

Specify SELECT Statement

Type info for SELECT. F4 for list

FROM table(s) _____

SELECT column(s) _____

WHERE conditions. _____

GROUP BY column(s). _____

HAVING condition(s) _____

ORDER BY column(s). _____

FOR UPDATE OF column(s) _____

F3=Exit F4=Prompt F5=Refresh F12=Cancel

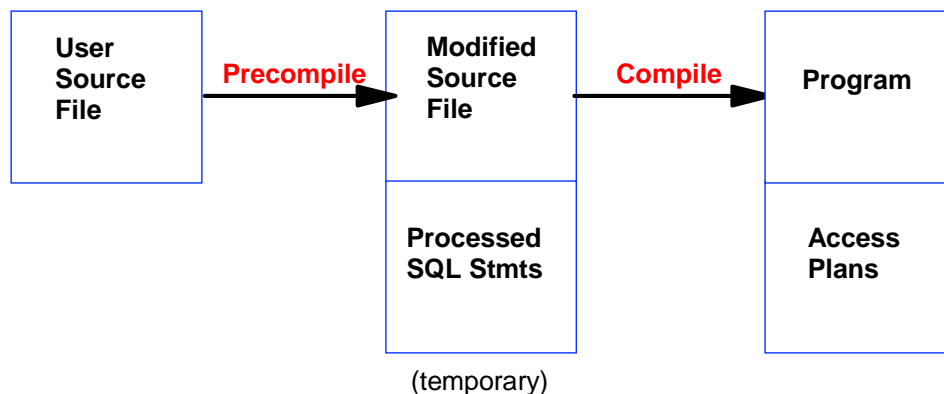
Why embed SQL in programs?

- Perform dynamic selection functions
 - ala OPNQRYF, except more flexible
- Perform set-at-a-time functions under program control
- Even to replace HLL I/O operations
 - e.g., READ, WRITE, UPDATE, CHAIN

What can be embedded?

- The SQL statements we have seen so far
 - e.g., SELECT, UPDATE, INSERT, CREATE TABLE, etc.
- Program control statements
 - e.g., DECLARE CURSOR, OPEN, CLOSE, FETCH, COMMIT, ROLLBACK

SQL Precompiler for Embedded SQL



Retrieve column/field values into program variables

One-to-one correspondence between SELECT list and INTO list

SELECT....INTO expects only a SINGLE row/record

- multiple rows require the use of cursor operations

```
* No F spec needed !  
  
D  EmpNbr      S          5 0  
D  Name        S          25  
D  Job         S          1  
  
C/EXEC SQL  
C+      SELECT  NAME, POS  
C+      INTO    :Name, :Job  
C+      FROM    EMPL  
C+      WHERE   NBR = :EmpNbr  
C/END-EXEC
```

Rules: Embedding SQL in RPG Code

All SQL statements must be coded on a C spec

SQL statements begin with /EXEC SQL in positions 7-15

- with the slash in position 7

and end with /END-EXEC in positions 7-15

You can enter SQL statements on the same line as /EXEC SQL

- However, /END-EXEC must be on a separate line

Between beginning and ending delimiters, all SQL statements must have + in position 7

SQL statements cannot go past position 80

SQL statements cannot be included via a /COPY statement

Retrieve column/field values into program variables

One-to-one correspondence between SELECT list and INTO list

SELECT....INTO expects only a SINGLE row/record

- multiple rows require the use of cursor operations

```
WORKING-STORAGE SECTION.  
  77 EMPNBR          PIC S9(5)  COMP-3.  
  77 DEPT             PIC S9(3)  COMP-3.  
  77 JOB              PIC X(25).  
  
PROCEDURE DIVISION.  
  EXEC SQL  
    SELECT name, pos  
      INTO :nam, :job  
    FROM empl  
    WHERE nbr = :EmpNbr  
  END-EXEC.
```

Using Structures in SQL

Host structures are groups of variables

- Data structures in RPG
- Group items in COBOL

Structures can be used in SQL statements

- Replaces list of variables

```
D EMP          DS  
D Job          5 0  
D Name        25  
D Sal         7 2  
D EmpNbr      S 1  
  
C/EXEC SQL  
C+ SELECT POS, NAME, SAL  
C+ INTO :EMP  
C+ FROM EMPL WHERE NBR = :EmpNbr  
C/END-EXEC
```

Steps to access multiple rows:

1. Declare cursor
2. Open cursor
3. Fetch a row (record)
4. Process row (UPDATE, INSERT, etc)
5. IF last row: go to Step 6,
 – ELSE go to Step 3
6. Close cursor

DECLARE CURSOR statement

Similar in function to HLL file declarations (F-specs or FD's)

- No processing actually takes place - just definition

Host variables may be included in the statement

Created using an embedded SELECT command

- most SELECT clauses may be used - ORDER BY, GROUP BY, etc

Must be declared before being referenced

```
C/EXEC SQL
C+   DECLARE empcsr CURSOR FOR
C+       SELECT nbr, nam, sal
C+       FROM emp
C+       WHERE dpt = :dept
C+
C/END-EXEC
```

DECLARE CURSOR - more clauses

Partner400

By default, all columns may be updated or deleted

- FOR UPDATE OF - lists the columns that are to be updated
 - columns listed in an ORDER BY clause may not be listed in FOR UPDATE OF clause also
- FOR READ ONLY - specifies no updating/deleting allowed

Considerations:

- FOR READ ONLY - may improve performance; better documentation
- FOR UPDATE OF - security; may improve performance

```
C/EXEC SQL
C+   DECLARE empcsr CURSOR FOR
C+       SELECT nbr, nam, sal
C+       FROM emp
C+       WHERE dpt = :dept
C+       FOR UPDATE OF sal
C/END-EXEC
```

DECLARE CURSOR - more clauses

Partner400

With Hold clause useful with Commitment Control

- By default, cursors are closed when Commit/Rollback commands execute
- With Hold - keeps cursor open
- With Hold also an optional clause on the Commit/Rollback commands

```
C/EXEC SQL
C+
C+   DECLARE empcsr CURSOR FOR
C+       WITH HOLD
C+       SELECT nbr, nam, sal
C+       FROM emp
C+       WHERE dpt = :dept
C+       FOR UPDATE OF sal
C+
C/END-EXEC
```

OPEN statement

Partner400

Actually executes the SQL Select statement

Builds the access path if necessary

Successful Open places the file cursor before the first row of the result table

Cursor must be closed before it can be opened

Syntax: OPEN cursor-name

```
C/EXEC SQL
C+
C+      OPEN      empcsr
C+
C/END-EXEC
```

FETCH statement:

Partner400

Two functions

- position the cursor for the next operation

```
C/EXEC SQL
C+
C+      FETCH NEXT FROM empcsr
C+
C/END-EXEC
```

- bring rows into the program

```
C/EXEC SQL
C+
C+      FETCH NEXT FROM empcsr
C+      INTO :number, :name, :salary
C+
C/END-EXEC
```

FETCH statement

Partner400

Alternatives to Next processing:

- must define the cursor as a scrollable cursor in the declare statement

```
C/EXEC SQL
C+
C+   DECLARE empcsr SCROLL CURSOR FOR
C+       SELECT nbr, nam, sal
C+           FROM emp
C+           ORDER BY empid
C+
C/END-EXEC

C/EXEC SQL
C+
C+   FETCH PRIOR FROM empcsr
C+   INTO :number, :name, :salary
C+
C/END-EXEC
```

FETCH statement

Partner400

- Alternatives to Next processing:

Keyword	Positions Cursor
Next	On the next row after the current row
Prior	On the row before the current row
First	On the first row
Last	On the last row
Before	Before the first row - must not use INTO
After	After the last row - must not use INTO
Current	On the current row (no change in position)
Relative <i>n</i>	n < -1 Positions to <i>n</i> th row before current n = -1 Same as Prior keyword n = 0 Same as Current keyword n = 1 Same as Next keyword n > 1 Positions to <i>n</i> th row after current

Positioned Update and Delete Stmts

Partner400

Update or delete the current row of an updatable cursor

- Can only be done after successful Fetch operation
- Add a "Where Current of" clause to the Update and Delete statements

```
C/EXEC SQL
C+   DECLARE empcsr CURSOR FOR
C+       SELECT nbr, nam, sal
C+       FROM emp
C+       ORDER BY empid
C+       FOR UPDATE OF sal
C/END-EXEC

C/EXEC SQL
C+   FETCH NEXT FROM empcsr
C+   INTO :number, :name, :salary
C/END-EXEC

C/EXEC SQL
C+   UPDATE emp
C+       SET sal = sal + :raise
C+       WHERE CURRENT OF empcsr
C/END-EXEC
```

Close Statement

Partner400

Close the cursor

- Cursor must be opened in order to be closed

DB2/400 may close cursors for other reasons also:

- job end
- activation group ends
- program ends
- modules ends
- commit or rollback without a 'with hold' clause
- error handling.....

```
C/EXEC SQL
C+
C+   CLOSE empcsr
C+
C/END-EXEC
```

Error Detection and Handling

Partner400

Status always returned in the code

- both successful and unsuccessful statements

Programmer must check return codes within program

SQL Communications Area (SQLCA)

- contains feedback information
- must be included in all SQL programs
- RPG includes SQLCA automatically
- other languages must have specific include:

```
/EXEC SQL  
  
INCLUDE SQLCA  
  
/END-EXEC
```

Error Detection and Handling

Partner400

SQL Communications Area (SQLCA)

SQLCAID	Char(8)	Structure identifying literal: "SQLCA"
SQLCABC	Integer	Length of SQLCA
SQLCode	Integer	Return code
SQLErrML	SmallInt	Length of SQLErrMC
SQLErrMC	Char(70)	Message Replacement text
SQLErrP	Char(8)	Product ID literal: "QSQ" for DB2/400
SQLErrD	Array of Integers	SQLErrD(1) - treated as Char(4); last 4 characters of CPF or other escape message SQLErrD(2) - treated as Char(4); last 4 characters of CPF or other diagnostic message SQLErrD(3) - for Fetch, Insert, Update or Delete, number of rows retrieved or updated SQLErrD(4) - for Prepare, relative number indicating resources required for execution SQLErrD(5) - for multiple-row Fetch, contains 100 if last available row is fetched; for Delete, number of rows affected by referential constraints; for Connect or Set Connection, contains t-1 if unconnected, 0 if local and 1 if connection is remote SQLErrD(6) - when SQLCode is 0, contains SQL completion message id

SQL Communications Area (SQLCA) continued

SQLWarn	Char(11)	Set of 11 warning indicators; each is blank, W, or N
SQLWarn0	Char(1)	Blank if all other SQLWARNx warning indicators are blank W if any warning indicator contains W or N
SQLWarn1	Char(1)	W if a string column was truncated when assigned to host variable
SQLWarn2	Char(1)	W if null values were eliminated from a function
SQLWarn3	Char(1)	W if number of columns is larger than number of host variables
SQLWarn4	Char(1)	W if prepared Update or Delete statement has no a Where clause
SQLWarn5	Char(1)	Reserved
SQLWarn6	Char(1)	W if date arithmetic results in end-of-month adjustment
SQLWarn7	Char(1)	Reserved
SQLWarn8	Char(1)	W if result of character conversion contains the substitution character
SQLWarn9	Char(1)	Reserved
SQLWarnA	Char(1)	Reserved
SQLState	Char(5)	Return code; '00000' if no error or warning

SQLCODE Error Handling

SQLCODE (SQLCOD) contains return code

- = 0 Successful statement execution
- > 0 Successful, with warning condition
- < 0 Unsuccessful - statement failed

SQLCODE value indicates exact error or condition

- e.g.. 100 = Row not found (or end of file)
- e.g.. -552 = Not authorized to object

SQLCODE values have corresponding messages

- e.g.. SQL0100 = Row not found
- e.g.. SQL0552 = Not authorized to &1.

Error Checking Within a HLL Program

Partner400

```
C/EXEC SQL
C+  SELECT name INTO :nam
C+    WHERE emp = :number
C/END-EXEC
C
C          If      SQLCod < 0
C          ExSr    Error
C          EndIf
C
C          If      SQLCod = 100
C          ExSr    NotFound
C          EndIf
```

RPG

```
EXEC SQL
  SELECT name INTO :lastname
  WHERE emp = Employee-Number
END-EXEC.
IF SQLCODE < 0
  PERFORM ERROR-ROUTINE.
IF SQLCODE = 100
  PERFORM NOT-FOUND-ROUTINE.
```

COBOL

WHENEVER Error Handling

Partner400

WHENEVER statement checks SQLCA

- can branch to a location based on condition

Three conditions:

- SQLWARNING (SQLCODE > 0 except 100)
– OR (SQLWARN0 = 'W')
- SQLERROR (SQLCODE < 0)
- NOT FOUND (SQLCODE = 100)

Two possible actions - neither very good!

- CONTINUE
- GO TO label

```
C/EXEC SQL
C+
C+  WHENEVER SQLERROR GO TO err
C+
C/END-EXEC
```

What is Dynamic SQL?

Partner400

A different way to use SQL

SQL statements are not predefined in program

- Dynamically created on the fly as part of program logic

SQL Precompiler cannot fully process dynamically created SQL statements

- PREPARE statement is used in program logic to compile dynamically created SQL statements at run time

Simple dynamic SQL statement process:

- Build SQL statement in a character variable
- PREPARE the SQL statement
- EXECUTE the SQL statement

Special considerations exist for SELECT statements

Where to use Dynamic SQL

Partner400

Report programs with user run time selection

- Files
- Fields
- Record selection criteria
- Sorting
- SQL built in functions

Whenever the exact syntax of an SQL statement cannot be determined beforehand

Dynamic SQL can be resource intensive

- A dynamic SQL statement has to be parsed (interpreted) and executed at run time
 - Negative performance impact
 - Use dynamic SQL only when necessary
-

Parameter Markers in Dynamic SQL

Partner400

Dynamic SQL statements cannot contain host variables

- e.g., :CUSTNO

Parameter markers are placed in embedded SQL statements

- Indicated by ?
- Used to dynamically insert host variable data for predicate values and/or column assignments
- Values are assigned to markers when the statement is executed
 - Example on next chart

```
C          Eval      SQLStmtStr = 'Delete From Customer Where -  
C          CUSTNO = ?'
```

Dynamic SQL - Example

Partner400

```
C          If          DeleteCorp  
C          Eval        Condition = 'Corp = ?'  
C          Else  
C          Eval        Condition = 'CustNo = ?'  
C          EndIf  
  
C          Eval        SQLStmtStr = 'Delete From Customer Where '  
C          + Condition  
  
C/EXEC SQL  
C+    PREPARE  DynSQLStmt  
C+    FROM    :SQLStmt  
  
C/END-EXEC  
C          If          (SQLCod = 0) And (SQLWn0 = *Blank)  
C/EXEC SQL  
C+    EXECUTE  DynSQLStmt  
C+    Using   :Cust  
  
C/END-EXEC  
C          EndIf
```

Use SQL source member types

- e.g., SQLRPG, SQLRPGLE, SQLCBL, SQLCBLLE
- Prompting won't work without SQL member type

You can prompt SQL statements in SEU

- You MUST key both EXEC SQL and END-EXEC statements first
 - Then you can prompt (F4) for statements in between
 - Same prompter as interactive SQL

Compile commands

- Pre-ILE compilers
 - CRTSQLRPG, CRTSQLCBL
- ILE compilers
 - CRTSQLRPGI, CRTSQLCBLI
 - Creates either *PGM, *SRVPGM or *MODULE depending on parameter value specified for "Compile type" or OBJTYPE

Test statements in Interactive SQL before embedding them

- When exiting Interactive SQL session
 - You can save session statements to a source member
 - Copy from this member into your program source

Default for SQL is to use Commitment Control

- Requires journaling
 - Program execution fails if updated files are not journaled
- To request no commitment control
 - COMMIT(*NONE) on compile

SQL precompile step happens before RPG/COBOL compile

- Therefore, if SQL syntax or semantic error occurs, no "typical" compile source listing available
- Can be very difficult to work through problems at this stage
- Try removing "SQL" from member type and compile "normally"
 - Compile will fail, but you can see results of externally described structures, COPYs, etc.

To help diagnose run-time problems

- Look in your job log after running the SQL program
 - Messages there often help diagnose problems
- Put your job in debug mode before running SQL program, then look in your joblog
 - Additional messages are put in joblog when in debug mode which can be helpful in diagnosing SQL performance problems

When using ILE source view debugger

- ILE SQL program compiles automatically generate 3 different views with DBGVIEW(*SOURCE):
 - Use F15 to switch between:
 - SQL Root Source view
 - SQL Output view (output from SQL precompiler)
 - Listing view

Performance Tips

SQL uses two basic ways to retrieve data

- Dataspace scan or arrival sequence
 - Generally used when MORE than 20% of records will be selected
- Index based or keyed access
 - Generally used when LESS than 20% of records will be selected

If SQL can use an index, performance usually improves significantly!

Create indexes for columns frequently referenced in

- WHERE clause
- GROUP BY clause
- ORDER BY clause

Create indexes for fields that are frequently used to join files

Use PRTSQLINF command and/or job log debug messages to see if indexes are being used by SQL optimizer

Query Manager

Partner400

With Query Manager users can:

- Create, run and manage queries and report forms
- Create, manage and query database files
 - QM Table support allows creation and data entry facilities

With Query Manager programmers can:

- Do all the above user functions
 - Embed queries into applications
 - Pass parameter data into queries at run time
 - Any part of the query (SQL statement) can be supplied
 - including the entire SQL statement itself!
-

Summary

Partner400

Practical, effective solution for applications requiring complex data retrieval

Very flexible, functional application development tool

- embedded in a HLL program
- or entered interactively

Portability to other relational databases

- Easy report writing with programmable flexibility

Similarity across many relational databases
