

Embedded SQL

Inserting SQL into HLL programs

Objectives

- ▶ Embed SQL commands in COBOL statements
- ▶ Retrieve single rows using embedded SQL
- ▶ Use cursors to retrieve multiple rows in embedded SQL
- ▶ Learn how to handle errors in programs containing embedded SQL commands

Introduction

- ▶ Use a procedural language to use embedded SQL commands to accomplish tasks beyond the capabilities of SQL
- ▶ Use EXEC SQL and END-SQL in COBOL programming to distinguish embedded SQL commands from standard COBOL statements

SQL Communication Area

- ▶ Communications area includes items that allow SQL to communicate various aspects of processing to the program
- ▶ Includes the SQLState variable
- ▶ 5 characters – used to communicate what happened after an SQL statement executes
 - '00000' – no warning or exception
 - '02000' – no row found
 - '01xxx' – warning
 - Yyxxx – error
 - Where yy is not 00, 01 or 02

Define the SQL Communication Area

- ▶ In the working-storage section

Exec SQL

 Include SQLCA

End-exec.

Using the SQL Communication Area

Exec SQL

Update Customer

set Discount = :NewDisc

where ShipCity = :SlcCusCity

End-exec.

Evaluate True

when SQLState = '00000'

perform ProcessRow

When SQLState = '01000'

perform SQLNoRow

When SQLState (1:2) = '01'

perform SQLWarning

When Other

perform SQLError

End-evaluate.

*Discount is a column
in Customer
NewDisc is a variable*

SQL Data Types in Cobol

SQL	Cobol Definition
Char(length)	Pic X(length).
VarChar(max-length)	01 variable-name. 49 string-length pic S9(4) Binary. 49 string-identifier pic X(max-length).
Dec(precision, scale)	Pic 9(precision-scale)V(scale) packed-decimal.
Numeric(precision, scale)	Pic 9(precision-scale)V(scale)
SmallInt	Pic S9(4) Binary
Int	Pic S9(9) Binary

SQL Data Types in Cobol

SQL	Cobol Definition
SmallInt	Pic S9(4) Binary
Int	Pic S9(9) Binary
Date	Format of date
Time	Format of time
Timestamp	Format of timestamp

Create Table in SQL

Create Table SALES_REP

(SLSREP_NUMBER

LAST

FIRST

STREET

CITY

STATE

ZIP_CODE

TOTAL_COMMISSION

COMMISSION_RATE

DECIMAL (2),

CHAR (10),

CHAR (8),

CHAR (15),

CHAR (15),

CHAR (2),

CHAR (5),

DECIMAL (7,2),

DECIMAL (3,2));

COBOL Variables

01 WS-SALES-REP.

03 WS-SLSREP_NUMBER	PIC S9(2)	COMP-3.
03 WS-LAST	PIC X(10).	
03 WS-FIRST	PIC X(8).	
03 WS-STREET	PIC X(15).	
03 WS-CITY	PIC X(15).	
03 WS-STATE	PIC X(2).	
03 WS-ZIP-CODE	PIC X(5).	
03 WS-TOTAL-COMMISSION	PIC S9(5)V9(2)	COMP-3.
03 WS-COMMISSION-RATE	PIC S9V9(2)	COMP-3.

PROCEDURE DIVISION

- ▶ Use Host variables
- ▶ Precede the variable with a colon
- ▶ Replace results of SQL queries in host variables in the INTO CLAUSE

```
SELECT LAST
```

```
  INTO :WS-SALES-REP
```

```
  FROM SALES_REP
```

```
  WHERE SLSREP_NUMBER = "03"
```

- ▶ Make provisions for exceptional conditions

Select Into

Exec SQL

Select

CustID,
Name,
ShipCity,
Discount

CustID, Name,
ShipCity, Discount
Are columns in the table

into:

:CsCustID,
:CsName,
:CsShipCity,
:CsDiscount

CSCustid, CSName
CSShipCity, CSDiscount
Are host variables in the
Program

from

Customer

where

CustId = :SlcCustId

End-exec.

: indicates a host variable in the program

Program Examples



Example 1

- ▶ Obtain the last name of sales rep number 03 and place it in W-LAST

Retrieve a Single Row and Column

- ▶ In SQL:

```
SELECT LAST  
  FROM SALES_REP  
  WHERE SLSREP_NUMBER = "03";
```

- ▶ In COBOL:

```
EXEC SQL  
  SELECT LAST  
    INTO :W-LAST  
  FROM SALES_REP  
  WHERE SLSREP_NUMBER = "03"  
END-EXEC.
```

Example 2

- ▶ Obtain all information about the sales rep whose number is stored in the host variable `W-SLSREP-NUMBER`

Retrieve a Single Row and All Columns

EXEC SQL

```
SELECT LAST, FIRST, STREET, CITY, STATE, ZIP_CODE,  
       TOTAL_COMMISSION, COMMISSION RATE  
INTO :W-LAST, :W-FIRST, :W-STREEET, :W-CITY,  
     :W-STATE, :W-ZIP-CODE, :W-TOTAL-COMMISSION,  
     :W-COMMISSION-RATE  
FROM SALES_REP  
WHERE SLSREP_NUMBER = :W-SLSREP-NUMBER  
END-EXEC.
```

Example 3

- ▶ Obtain the last name, first name, and address of the customer whose customer number is stored in the host variable, W-CUSTOMER-NUMBER, as well as the number, last name, and first name of the sales rep who represents this customer

Retrieve a Single Row from a Join

EXEC SQL

```
SELECT CUSTOMER.LAST, CUSTOMER.FIRST, CUSTOMER.STREET,  
CUSTOMER.CITY, CUSTOMER.STATE, CUSTOMER.ZIP_CODE,  
CUSTOMER.SLSREP_NUMBER, SALES_REP.LAST,  
SALES_REP.FIRST
```

```
INTO :W-LAST OF :W-CUSTOMER,  
:W-FIRST OF :W-CUSTOMER, :W-STREET OF  
:W-CUSTOMER, :W-CITY OF :W-CUSTOMER,  
:W-STATE OF :W-CUSTOMER, :W-ZIP-CODE OF  
:W-CUSTOMER, :W-SLSREP-NUMBER OF  
:W-CUSTOMER, :W-LAST OF :W-SALES-REP,  
:W-FIRST OF :W-SALES-REP  
FROM SALES-REP, CUSTOMER  
WHERE SALES-REP.SLSREP_NUMBER =  
CUSTOMER.SLSREP_NUMBER  
AND CUSTOMER.CUSTOMER_NUMBER =  
:W-CUSTOMER-NUMBER
```

END-EXEC.

Multiple-Row SELECT

- ▶ Example of SELECT statement producing multiple rows

EXEC SQL

```
SELECT CUSTOMER_NUMBER, LAST, FIRST  
      INTO :W-CUSTOMER-NUMBER, :W-LAST, :W-FIRST  
      FROM CUSTOMER  
      WHERE SLSREP_NUMBER = :W-SLSREP-NUMBER
```

END-EXEC

- ▶ Problem:
 - COBOL can process only one record at a time, where this SQL command produces multiple rows (records)

Cursors

▶ Cursor

- A pointer to a row in the collection of rows retrieved by a SQL statement
- Advances one row at a time to provide sequential, record-at-a-time access to the retrieved rows so COBOL can process the rows
- Using a cursor, COBOL can process the set of retrieved rows as though they were records in a sequential file.

Example 4

- ▶ Retrieve the number, last name, and first name of every customer represented by the sales rep whose number is stored in the host variable W-SLSREP-NUMBER

Using a Cursor

EXEC SQL

DECLARE CUSTGROUP CUROSUR FOR

SELECT CUSTOMER_NUMER, LAST, FIRST

FROM CUSTOMER

WHERE SLSREP_NUMBER = :W-SLSREP-NUMBER

END-EXEC.

OPEN, FETCH, CLOSE

- ▶ OPEN, FETCH, and CLOSE commands are used in processing a cursor
- ▶ Analogous to the OPEN, READ, and CLOSE commands used in processing a sequential file

Opening a Cursor

```
EXEC SQL  
    OPEN CUSTGROUP  
END-EXEC.
```

Fetching Rows from a Cursor

```
EXEC SQL  
  FETCH CUSTGROUP  
    INTO :W-CUSTOMER-NUMBER, :W-LAST, :W-FIRST  
END-EXEC.
```

Closing a Cursor

```
EXEC SQL  
    CLOSE CUSTGROUP  
END-EXEC.
```

Example 5

- ▶ For every order that contains an order line for the part whose part number is stored in W-PART-NUMBER, retrieve the order number, order date, last name, and first name of the customer who placed the order, and the number, last name, and first name of the sales rep who represent the customer
- ▶ Sort the results by customer number

More Complex Cursors

```
EXEC SQL
  DECLARE ORDGROUP CURSOR FOR
  SELECT ORDERS.ORDER_NUMBER, ORDERS.ORDER_DATE,
         CUSTOMER.CUSTOMER_NUMBER, CUSTOMER.LAST, FIRST,
         SALES_REP.SLSREP_NUMBER, SALES_REP.LAST, FIRST
  FROM ORDER_LINE, ORDERS, CUSTOMER, SALES_REP
  WHERE ORDER_LINE.PART_NUMBER = :W-PART-NUMBER
  AND ORDER_LINE.ORDER_NUMBER =
     ORDERS.ORDER_NUMBER
  AND ORDERS.CUSTOMER_NUMBER =
     CUSTOMER.CUSTOMER_NUMBER
  AND CUSTOMER.SLSREP_NUMBER =
     SALES_REP.SLSREP_NUMBER
  ORDER BY CUSTOMER.CUSTOMER_NUMBER
END-EXEC.
```

Advantages of Cursors

- ▶ The coding the in the program is greatly simplified
- ▶ A special component of the database management system called the optimizer determines the best way to access the data
- ▶ If the database structure changes in such a way that the necessary information is still obtainable using a different query, the only change required in the program is the cursor definition in WORKING-STORAGE (PRODECURE DIVISION code is not affected)