Introduction to SQL programming Techniques- Embedded SQL

P. Mirunalini

Outline

- Database Programming
- Embedded SQL

Objectives

- Most database access in practical applications is accomplished through software programs that implement database applications
- Various techniques for accessing and manipulating a database via programs

Database Programming

- Most DBMS system has an interactive interface which is convenient but not sufficient.
- Majority of the database interaction are excuted through programs, generally called as application programs or database application.
  - Eg: Canned transactions by end users
- Another common use of database programming is to access a database through an application program that implements a Web interface.
  - Eg: Airline reservations or online purchases

Database Programming Approaches

- Different approaches for database programming:
  - Embedded commands in general purpose programming language
  - Using library of database functions or classes
  - Designing a brand new, full-fledged language

Database Programming Approaches

Embedded commands:

- Database commands are embedded in a general-purpose programming language
- Programming language is called as host language.
- Database statements are identified by prefix EXEC SQL which precedes all SQL commands in the host language programs
- A pre-compiler scans the source program to identify database statements for processing by DBMS.
- They are replaced in the program by function calls to the DBMS-generated code
- This technique is generally referred to as embedded SQL

Database Programming Approaches

Using Library of database functions:

- Library functions available to the host language for database calls;
- Functions to connect database, prepare a query, execute a query loop over the query record.
- Actual DB query and update commands and other info included as parameters to function calls.
- Approach known as an API(Application Program Interface)
- Eg: JDBC class library as connection objects, query objects and query result objects.

Database Programming Approaches

Designing brand new language:

- A database programming language deigned from scratch to be compatible with the database model and query language.
- Programming structures such as loops and conditional statements are added to the database language to convert it into a full-fledged programming language.
- Eg: Oracle's PL/SQL, SQL/PSM

Impedance Mismatch

- Impedance match: Incompatibilities between a host programming language and the database model, e.g.,
  - The data types of the programming language differ from the attribute data types that are available in the data model.
    - Type mismatch and incompatibilities; requires a new binding for each attribute type with compatible programming language types.
    - A different binding is needed *for each programming language* because different languages have different data types.
    - The data types available in C/C++ and Java are different, and both differ from the SQL data types, which are the standard data types for relational databases.

Impedance Mismatch

- The results of most queries are sets or multisets of tuples (rows), and each tuple is formed of a sequence of attribute values
  - Binding is needed to map the query result of the database model with appropriate data structure of pl model
- A mechanism is needed to loop over the tuples in a **query result** in order to access a single tuple at a time and to extract individual values from the tuple.
  - Need special iterators to loop over query results and manipulate individual values. (cursor or iterator variable is used)

Impedance Mismatch

- Impedance mismatch is less of a problem when a special database programming language is designed that uses the same data model and data types as the database model.
- Example: Oracle's PL/SQL.
- The SQL standard also has a proposal for such a database programming language, known as *SQL/PSM*.
- The object data model is quite similar to the data model of the Java programming language, so the impedance mismatch is greatly reduced when Java is used as the host language for accessing a Java-compatible object database.

Steps in Database Programming

1. Client program *opens a connection* to the database server
2. Client program *submits queries to and/or updates* the database
3. When database access is no longer needed, client program *closes (terminates) the connection*

SQL Commands for Connecting to a Database

- Connection (multiple connections are possible but only one is active)

```
CONNECT TO <server-name> AS <connection-name> AUTHORIZATION <user-account-info>;
```

- Change from an active connection to another one

```
SET CONNECTION connection-name;
```

- Disconnection

```
DISCONNECT connection-name;
```

Embedded SQL

- Most SQL statements can be embedded in a general-purpose *host* programming language such as COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between **EXEC SQL** or **EXEC SQL BEGIN** and a matching **END-EXEC** or **EXEC SQL END** (or semicolon)
  - Syntax may vary with language
  - *Shared variables* (used in both languages) usually prefixed with a colon (:) in SQL
  - Names of the database constructs such as attributes and relations can be used within SQL but shared program variables can be used in C program without (:)

Embedded SQL

- Embedded SQL statements are prefixed by EXEC SQL, and terminated by a special terminator symbol.
- SQL statements can include references to host variables; such references must include a colon prefix.
- All host variables must be declared within an embedded SQL declare section.
- After the execution of any SQL statement, a status code is returned to the program in a host variable called SQLSTATE
- SQLCODE=0 – indicates statement was executed successfully
- SQLCODE<0 – indicates some error has occurred
- SQLCODE>0 – indicates no data found

Embedded SQL

- SQL standard, a communication variable called **SQLSTATE** was added, which is a string of five characters.
- A value of ‘00000’ in SQLSTATE indicates no error or exception;
- A value of ‘02000’ indicates ‘no more data’ when using SQLSTATE.
- Both SQLSTATE and SQLCODE are available in the SQL standard.

Example: Variable Declaration in Language C

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements
- **SQLCODE, SQL state** is used to communicate errors/exceptions between the database and the program

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
 varchar dname[16], fname[16], …;
 char ssn[10], bdate[11], …;
 int dno, dnumber,
  int  SQLCODE,char SQLSTATE;
EXEC SQL END DECLARE SECTION;
```

Embedded SQL in C Programming Examples

```
loop = 1;
while (loop) {
   prompt ("Enter SSN: ", ssn);
   EXEC SQL
    select FNAME, LNAME, ADDRESS, SALARY
    into :fname, :lname, :address, :salary
    from EMPLOYEE where SSN == :ssn;
    if (SQLCODE == 0) printf(fname, …);
    else printf("SSN does not exist: ", ssn);
    prompt("More SSN? (1=yes, 0=no): ", loop);
   END-EXEC
}
```

Processing Query Results using Cursors

- Retrieval operations requires special treatment since it retrieves many rows, not just one.
- Host languages are generally not equipped to handle the retrieval of more than one row at a time.
- To bridge the gap between set-level retrieval capabilities of SQL and row-level retrieval capabilities of host – cursor are used.
- Cursor – a logical pointer (in application) pointing to each of the rows thereby providing addressability to those rows one at a time.

Embedded SQL in C Programming Examples

- A **cursor** (iterator) is needed to process multiple tuples
- **`FETCH`** commands move the cursor to the *next* tuple
- **`CLOSE CURSOR`** indicates that the processing of query results has been completed

Operations Involving Cursors

Cursor – a mechanism for accessing the rows in the set one by one.

- Cursor is declared by means of DECLARE CURSOR.
- The table expression is evaluated when the cursor is opened.

```
EXEC SQL DECLARE <cursor name> CURSOR
FOR [ table expression ] /*define the cursor */

EXEC SQL DECLARE X CURSOR FOR
SELECT S.SUPPLIER_NUMBER, S.SUPPLIER_NAME, S.STATUS
FROM SUPPLIERS S
WHERE S.CITY = :Y ;
```

Operations Involving Cursors

EXEC SQL DECLARE <cursor name> CURSOR

- opens the specified cursor.

- the table expression associated with the cursor is evaluated.

- a set of rows becomes the current active set for the cursor.

EXEC SQL FETCH <cursor name> INTO <host variable reference>;

- advances the specified cursor to the next row in the active set

- then assigns the *i* th value to the *i* th host variable

EXEC SQL CLOSE <cursor name>;

closes the specified cursor. The cursor now has no current active set.

Operations Involving Cursors

- FETCH appears inside the loop since there will be many rows in the result set.
- INTO clause assigns the retrieved values to host variables.

```
EXEC SQL DECLARE X CURSOR FOR
SELECT S.SUPPLIER_NUMBER, S.SUPPLIER_NAME, S.STATUS
FROM SUPPLIERS S
WHERE S.CITY = :Y ;
EXEC SQL OPEN X;
DO for all SUPPLIERS rows accessible via X;
EXEC SQL FETCH X INTO :SUPPLIER_NUMBER, :SUPPLIER_NAME,
    :STATUS
..... /* fetch next supplier*/
END;
EXEC SQL CLOSE X;
```

Embedded SQL Uses Cursors for Updation

```
0) prompt("Enter the Department Name: ", dname) ;
1) EXEC SQL
2)    SELECT Dnumber INTO :dnumber
3)     FROM DEPARTMENT WHERE Dname = :dname ;
4) EXEC SQL DECLARE EMP CURSOR FOR
5)    SELECT Ssn, Fname, Minit, Lname, Salary
6)     FROM EMPLOYEE WHERE Dno = :dnumber
7) FOR UPDATE OF Salary ;
8) EXEC SQL OPEN EMP ;
```

Embedded SQL in C Programming Examples

```
9) EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname,
     :salary ;
10) while (SQLCODE = = 0) {
11) printf("Employee name is:", Fname, Minit, Lname) ;
12) prompt("Enter the raise amount: ", raise) ;
13) EXEC SQL
14)    UPDATE EMPLOYEE
15)    SET Salary = Salary + :raise
16)    WHERE CURRENT OF EMP ;
17) EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname,
     :salary ;
18) }
19) EXEC SQL CLOSE EMP ;
```