Database Application Development

Chapter 6

Overview

Concepts covered in this lecture:

- SQL in application code
- Embedded SQL
- Cursors
- Dynamic SQL
- JDBC
- SQLJ
- Stored procedures

SQL in Application Code

- SQL commands can be called from within a host language (e.g., C++ or Java) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
- Three main integration approaches:
  - Embedded SQL: write SQL in the host language (e.g., SQLJ)
  - CLI: Create special API to call SQL commands (e.g., JDBC)
  - SQL/PL: SQL extended with programming constructs

SQL in Application Code (cont.)

Impedance mismatch:

- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure exist traditionally in procedural programming languages. Nowadays:
  - C++ with the STL
  - Java with utils (vector, etc.)
- SQL supports a mechanism called a *cursor* to handle this.
  - This is like an *iterator* in Java.

Embedded SQL

- Approach: Embed SQL in the host language.
  - A preprocessor converts the SQL statements into special API calls.
  - Then a regular compiler is used to compile the code.

- Language constructs:
  - Connecting to a database:

    `EXEC SQL CONNECT`
  - Declaring variables:

    `EXEC SQL BEGIN (END) DECLARE SECTION`
  - Statements:

    `EXEC SQL Statement;`

Embedded SQL: Variables

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

- Two special “error” variables:
  - `SQLCODE` (long, is negative if an error has occurred)
  - `SQLSTATE` (char[6], predefined codes for common errors)

Cursors

- Can declare a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.
  - Can use a special clause, called ORDER BY, in queries that are accessed through a cursor, to control the order in which tuples are returned.
    - Fields in ORDER BY clause must also appear in SELECT clause.
  - The ORDER BY clause, which orders answer tuples, is *only* allowed in the context of a cursor.
- Can also modify/delete tuple pointed to by a cursor.

Cursor that gets names of sailors who've reserved a red boat, in alphabetical order

```
EXEC SQL DECLARE sinfo CURSOR FOR
        SELECT S.sname
        FROM Sailors S, Boats B, Reserves R
        WHERE S.sid=R.sid AND R.bid=B.bid AND B.color=‘red’
        ORDER BY S.sname
```

- Note that it is illegal to replace *S.sname* by, say, *S.sid* in the ORDER BY clause!
- Can we add *S.sid* to the SELECT clause and replace *S.sname* by *S.sid* in the ORDER BY clause?

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Dynamic SQL

- SQL query strings are now always known at compile time (e.g., spreadsheet, graphical DBMS frontend): Allow construction of SQL statements on-the-fly
- Example:

```
char c_sqlstring[]=
  "DELETE FROM Sailors WHERE raiting>5";
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

Database APIs: Alternative to embedding

Rather than modify compiler, add library with database calls (API)

- Special standardized interface: procedures/objects
- Pass SQL strings from language, presents result sets in a language-friendly way
- Sun's *JDBC:* Java API
- Supposedly DBMS-neutral
  - a "driver" traps the calls and translates them into DBMS-specific code
  - database can be across a network

JDBC: Architecture

- Four architectural components:
  - Application (initiates and terminates connections, submits SQL statements)
  - Driver manager (load JDBC driver)
  - Driver (connects to data source, transmits requests and returns/translates results and error codes)
  - Data source (processes SQL statements)

JDBC Architecture (cont.)

Four types of drivers:

Bridge:

- Translates SQL commands into non-native API. Example: JDBC-ODBC bridge. Code for ODBC and JDBC driver needs to be available on each client.

Direct translation to native API, non-Java driver:

- Translates SQL commands to native API of data source. Need OS-specific binary on each client.

Network bridge:

- Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.

Direction translation to native API via Java driver:

- Converts JDBC calls directly to network protocol used by DBMS. Needs DBMS-specific Java driver at each client.

JDBC Classes and Interfaces

Steps to submit a database query:

- ❖ Load the JDBC driver
- ❖ Connect to the data source
- ❖ Execute SQL statements

JDBC Driver Management

- All drivers are managed by the DriverManager class
- Loading a JDBC driver:
  - In the Java code:

    ```
    Class.forName("oracle/jdbc.driver.Oracledriver");
    ```

  - When starting the Java application:

    ```
    -Djdbc.drivers=oracle/jdbc.driver
    ```

Connections in JDBC

We interact with a data source through sessions. Each connection identifies a logical session.

- JDBC URL:
  jdbc:<subprotocol>:<otherParameters>

Example:

```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
   con = DriverManager.getConnection(url,usedId,password);
} catch SQLException excpt { … }
```

Connection Class Interface

- `public int getTransactionIsolation()` and `void setTransactionIsolation(int level)` Sets isolation level for the current connection.
- `public boolean getReadOnly()` and `void setReadOnly(boolean b)` Specifies whether transactions in this connection are read-only
- `public boolean getAutoCommit()` and `void setAutoCommit(boolean b)` If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using `commit()`, or aborted using `rollback()`.
- `public boolean isClosed()` Checks whether connection is still open.

Executing SQL Statements

- Three different ways of executing SQL statements:
  - `Statement` (both static and dynamic SQL statements)
  - `PreparedStatement` (semi-static SQL statements)
  - `CallableStatment` (stored procedures)

- `PreparedStatement` class: Precompiled, parametrized SQL statements:
  - Structure is fixed
  - Values of parameters are determined at run-time

Executing SQL Statements (Contd.)

```
String sql=“INSERT INTO Sailors VALUES(?,?,?,?)”;
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);

// we know that no rows are returned, thus we use
   executeUpdate()
int numRows = pstmt.executeUpdate();
```

ResultSets

- ❖ `PreparedStatement.executeUpdate` only returns the number of affected records
- ❖ `PreparedStatement.executeQuery` returns data, encapsulated in a `ResultSet` object (a cursor)

```
ResultSet rs=pstmt.executeQuery(sql);
// rs is now a cursor
While (rs.next()) {
 // process the data
}
```

ResultSets (cont.)

A ResultSet is a powerful iterator (cursor):

- previous(): moves one row back
- absolute(int num): moves to the row with the specified number
- relative (int num): moves forward or backward
- first() and last()

Matching Java and SQL Data Types

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC: Exceptions and Warnings

- Most of java.sql can throw an SQLException if an error occurs.
- SQLWarning is a subclass of EQLException; not as severe (they are not thrown and their existence has to be explicitly tested)

Warning and Exceptions (Contd.)

```
try {
  stmt=con.createStatement();
  warning=con.getWarnings();
  while(warning != null) {
    //handle SQLWarnings;
    warning = warning.getNextWarning():
  }
  con.clearWarnings();
  stmt.executeUpdate(queryString);
  warning = con.getWarnings();
  ...
} //end try
catch( SQLException SQLe) {
  //handle the exception
}
```

Examining Database Metadata

DatabaseMetaData object gives information about the database system and the catalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
   “Name:” + md.getDriverName() +
   “version: ” + md.getDriverVersion());
```

Database Metadata (Contd.)

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
   tableName = trs.getString(“TABLE_NAME”);
   System.out.println(“Table: “ + tableName);
   //print all attributes
   ResultSet crs = md.getColumns(null,null,tableName, null);
   while (crs.next()) {
      System.out.println(crs.getString(“COLUMN_NAME” + “, “);
   }
}
```

A (Semi-)Complete Example

```
Connection con = // connect
  DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
   // loop through result tuples
   while (rs.next()) {
      String s = rs.getString(“name");
      Int n = rs.getFloat(“rating");
      System.out.println(s + "   " + n);
   }
} catch(SQLException ex) {
   System.out.println(ex.getMessage ()
      + ex.getSQLState () + ex.getErrorCode ());
}
```

SQLJ

Complements JDBC with a (semi-)static query model: Compiler can perform syntax checks, strong type checks, consistency of the query with the schema

- All arguments always bound to the same variable:
  ```
  #sql = {
     SELECT name, rating INTO :name, :rating
     FROM Books WHERE sid = :sid
  };
  ```
- Compare to JDBC:
  ```
  sid=rs.getInt(1);
  if (sid==1) {sname=rs.getString(2);}
  else { sname2=rs.getString(2);}
  ```

- SQLJ (part of the SQL standard) versus embedded SQL (vendor-specific)

SQLJ Code

```
Int sid; String name; Int rating;
// named iterator
#sql iterator Sailors(Int sid, String name, Int rating);
Sailors sailors;
// assume that the application sets rating
#sailors = {
    SELECT sid, sname INTO :sid, :name
     FROM Sailors WHERE rating = :rating
};
// retrieve results
while (sailors.next()) {
    System.out.println(sailors.sid + “ “ + sailors.sname));
}
sailors.close();
```

SQLJ Iterators

Two types of iterators (“cursors”):

- Named iterator
  - Need both variable type and name, and then allows retrieval of columns by name.
  - See example on previous slide.
- Positional iterator
  - Need only variable type, and then uses FETCH .. INTO construct:

```
#sql iterator Sailors(Int, String, Int);
Sailors sailors;
#sailors = …
while (true) {
   #sql {FETCH :sailors INTO :sid, :name} ;
   if (sailors.endFetch()) { break; }
   // process the sailor
}
```

Stored Procedures

- What is a stored procedure:
  - Program executed through a single SQL statement
  - Executed in the process space of the server
- Advantages:
  - Can encapsulate application logic while staying “close” to the data
  - Reuse of application logic by different users
  - Avoid tuple-at-a-time return of records through cursors

Stored Procedures: Examples

```
CREATE PROCEDURE ShowNumReservations
   SELECT S.sid, S.sname, COUNT(*)
   FROM Sailors S, Reserves R
   WHERE S.sid = R.sid
   GROUP BY S.sid, S.sname
```

Stored procedures can have parameters:

- Three different modes: IN, OUT, INOUT

```
CREATE PROCEDURE IncreaseRating(
   IN sailor_sid INTEGER, IN increase INTEGER)
UPDATE Sailors
   SET rating = rating + increase
   WHERE sid = sailor_sid
```

Stored Procedures: Examples (cont.)

Stored procedure do not have to be written in SQL:

```
CREATE PROCEDURE TopSailors(
   IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME “file:///c:/storedProcs/rank.jar”
```

Calling Stored Procedures

```
EXEC SQL BEGIN DECLARE SECTION
Int sid;
Int rating;
EXEC SQL END DECLARE SECTION

// now increase the rating of this sailor
EXEC CALL IncreaseRating(:sid,:rating);
```

Calling Stored Procedures (cont.)

JDBC:

```
CallableStatement cstmt=
    con.prepareCall(“{call
    ShowSailors});
ResultSet rs =
    cstmt.executeQuery();
while (rs.next()) {
   …
}
```

SQLJ:

```
#sql iterator
    ShowSailors(… );
ShowSailors showsailors;
#sql showsailors={CALL
    ShowSailors};
while (showsailors.next()) {
    …
}
```

SQL/PSM (Persistent Stored Modules)

Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL) → SQL/PSM standard is a representative

Declare a stored procedure:

```
CREATE PROCEDURE name(p1, p2, … , pn)
    local variable declarations
    procedure code;
```

Declare a function:

```
CREATE FUNCTION name (p1, … , pn) RETURNS
    sqlDataType
    local variable declarations
    function code;
```

Main SQL/PSM Constructs

```
CREATE FUNCTION rate Sailor
        (IN sailorId INTEGER)
        RETURNS INTEGER
DECLARE rating INTEGER
DECLARE numRes INTEGER
SET numRes = (SELECT COUNT(*)
                          FROM Reserves R
                           WHERE R.sid = sailorId)
IF (numRes > 10) THEN rating =1;
ELSE rating = 0;
END IF;
RETURN rating;
```

Main SQL/PSM Constructs (cont.)

- Local variables (DECLARE)
- RETURN values for FUNCTION
- Assign variables with SET
- Branches and loops:
  - IF (condition) THEN statements;
    ELSEIF (condition) statements;
    … ELSE statements; END IF;
  - LOOP statements; END LOOP
- Queries can be parts of expressions
- Can use cursors naturally without “EXEC SQL”

Summary

- ❖ Embedded SQL allows execution of parametrized static queries within a host language
- ❖ Dynamic SQL allows execution of completely ad-hoc queries within a host language
- ❖ Cursor mechanism allows retrieval of one record at a time and bridges impedance mismatch between host language and SQL
- ❖ APIs such as JDBC introduce a layer of abstraction between application and DBMS

Summary (cont.)

- ❖ SQLJ: Static model, queries checked a compile-time.
- ❖ Stored procedures execute application logic directly at the server
- ❖ SQL/PSM standard for writing stored procedures