Database Application Development

Linda Wu

(CMPT 354 • 2004-2)

Topics

- SQL in application code
- Embedded 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)
  - There must be a statement to connect to the right database
- Two main integration approaches
  - Embed SQL in the host language (Embedded SQL, SQLJ)
  - Create special API to call SQL commands (JDBC)

SQL in Application Code (Cont.)

- Impedance mismatch
  - The result of a SQL query is a (multi-)set of records, with no *a priori* bound on the number of records
  - C and Java have no (multi-)set data type

* SQL supports a mechanism called a cursor to handle this

Embedded SQL

- Approach: embed SQL in the host language
  - A DBMS-specific preprocessor converts the SQL statements into special function calls in the host language
  - Then a regular compiler is used to compile the code
  - 2 types
    - Static SQL: prepared at compile time
    - Dynamic SQL: constructed and prepared at run time

Embedded SQL (Cont.)

- Static SQL: C language constructs

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
...... ......
EXEC SQL CONNECT TO db_name USER user_id
     USING password
...... ......
EXEC SQL
INSERT INTO Sailors VALUES
       (:c_sname, :c_sid, :c_rating, :c_age);
```

Variable declaration

Connecting to a DB

SQL statement

Embedded SQL (Cont.)

- Dynamic SQL
  - SQL query strings are not always known at compile time (e.g., spreadsheet, graphical DBMS frontend)
  - Dynamic SQL allows construction of SQL statements on-the-fly
  - Example in C language

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


Embedded SQL (Cont.)

- Two special variables for error reporting
  - SQLCODE
    - Type: long (C), int (Java)
    - Negative if an error has occurred
    - Specific to each driver
  - SQLSTATE
    - Type: char[6] (C), String (Java)
    - Defined in X/Open and ANSI SQL92 that identifies common errors and exceptions
    - Examples

      08001 -- No suitable driver
      HY011 -- Operation invalid at this time
      02000 -- No data

Embedded SQL (Cont.)

- Cursors
  - Declared on a relation or query statement
  - Examines a set of records returned by an embedded SQL statement in the host language
    - *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved
  - Rows pointed to by a cursor can be modified or deleted

Embedded SQL (Cont.)

- Cursor declaration

```
DECLARE    cursorname
FOR        some query
ORDER BY   order-item-list
```

  - ORDER BY clause (optional)
    - To control the order in which tuples are returned (ASC, DESC)
    - Fields in ORDER BY clause must also appear in SELECT clause
    - This clause is only allowed in the context of a cursor
    - ORDER BY is the last step in evaluating the query

Embedded SQL (Cont.)

- Cursor that gets names of sailors who have 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
```


Embedded SQL (Cont.)

Example in C

```
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 ASC, S.age DESC;
EXEC SQL OPEN sinfo;
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;
```

Database APIs

- Rather than modify compiler, add library with database calls (API)
  - ODBC (Open DataBase Connectivity)
  - JDBC (Java DataBase Connectivity)
  - Standardized interface: procedures/objects
  - Pass SQL strings from the host language, presents result sets in a language friendly way
  - Application using ODBC or JDBC is DBMS independent
    - A DBMS-specific driver translates ODBC / JDBC calls into DBMS-specific code
    - Database (data source) can be across a network

JDBC

- Architecture: four components
  - Application
    - Initiate and terminate connections, submit SQL statements, retrieve results
  - Driver manager
    - Load JDBC drivers
  - Driver
    - Establish connection with data source, transmit requests, return results and error codes
  - Data source
    - Process commands from drivers

JDBC (Cont.)

JDBC drivers: four types

1. Bridge
   - Translate JDBC function calls into API that is not native to the DBMS; e.g., JDBC-ODBC bridge
2. Direct translation to native API via non-Java driver
   - Translate JDBC calls to API of a specific data source; driver is written in a combination of C++ and Java
3. Network bridge
   - Send JDBC calls over the network to a middleware server that talks to the data source
4. Direct translation to native API via Java driver
   - Communicate with DBMS via Java sockets, and convert JDBC calls directly to native API of data source; driver on the client is written in Java

JDBC (Cont.)

- JDBC classes and interfaces
  - java.sql package
    - Data source connection, SQL execution, result retrieval, transaction management, exception handling
  - javax.sql package
    - Connection pool, RowSet interface
- How to submit a database query
  - Load the JDBC driver
  - Connect to the data source
  - Execute SQL statements

JDBC (Cont.)

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc:dbname";
String myQuery = "SELECT sname FROM Sailors";
try {
    Connection con =
            DriverManager.getConnection(
                    url, "myid", "mypwd");
    stmt = con.createStatement();
    stmt.executeQuery(myQuery);
    stmt.close();
    con.close();
} catch(SQLException ex) {
    ……
}
```


JDBC (Cont.)

- All drivers are managed by *DriverManager* class
- Connection is specified via a JDBC URL
  jdbc:<subprotocol>:<otherParameters>
- Three different ways of executing SQL statements
  - Statement (static and dynamic SQL statements)
  - PreparedStatement (semi-static SQL statements)
  - CallableStatment (stored procedures)
  - * Statement is the base class of both PreparedStatement and CallableStatment
- Remember to close connections
- Most of the methods in java.sql package can throw an SQLException if an error occurs

JDBC (Cont.)

```
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);

int numRows = pstmt.executeUpdate();
```

"?": placeholder; can be replaced with a value

JDBC (Cont.)

- SQL query submission
  - executeQuery()
    - Used when SQL statement returns records (SELECT)
    - Returns a *ResultSet* object
  - executeUpdate()
    - Used when SQL statement returns no records (DELETE, UPDATE, INSERT, ALTER)
    - Returns the number of modified rows
  - execute() and executeBatch()

JDBC (Cont.)

- Result retrieval using ResultSet
  - ResultSet is a powerful cursor
  - previous(), absolute(int), relative (int), first(), last(), ...

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


JDBC (Cont.)

- Matching Java and SQL Data Types

SQL Type	Java class	ResultSet methods
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()

SQLJ (SQL-Java)

- Complement JDBC with a static model
- Compiler can check syntax, type, and consistency of the query with the schema
- Host language variables are always bound to the same arguments

SQLJ

```
#sql sailors = {
    SELECT sname, rating INTO :name, :rating
    FROM Sailors WHERE sid = :sid};
```

JDBC

```
// rs is a ResultSet cursor on Sailors
sid = rs.getInt(1);
if (sid == 1) { sname  = rs.getString(2); }
else          { sname2 = rs.getString(2); }
```


SQLJ (Cont.)

- Writing SQLJ application

```
String name; Double age; int rating;
#sql iterator Sailors(String name, Double age);
Sailors sailors;

#sql sailors = {
   SELECT sname, age  INTO :name, :age
   FROM   Sailors  WHERE rating = :rating };

while ( sailors.next() ) {
   System.out.println( sailors.name() + " " +
                       sailors.age() );
}
sailors.close();
```

Declare Iterator class

Instantiate an object

Initialize Iterator

Read result

Close Iterator object

SQLJ (Cont.)

Two types of iterators (“cursors”)

- Named iterator
  - Both the variable type and the name of each column of the iterator are specified
  - Allow retrieval of columns by name
- Positional iterator
  - Specify only the variable type of column

```
#sql iterator Sailors(String, Double);
Sailors sailors;
#sql sailors = { … };
while (true) {
    #sql {FETCH :sailors INTO :name, :age};
    if ( sailors.endFetch() ) { break; }
}
```


Stored Procedures

- Stored procedure
  - A program executed through a single SQL statement
  - Executed in the process space of database server
- Advantages
  - Encapsulate application logic without knowing database schema
  - Minimize the amount of data transferred between server and client; avoid tuple-at-a-time return of records through cursors
  - Reused by different users

Stored Procedures (Cont.)

* Stored procedure must have a name

```
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
```

* Parameter 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 (Cont.)

- Calling stored procedures

Embedded SQL

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

EXEC SQL CALL IncreaseRating(:sid,:rating);
```

JDBC

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

Stored Procedures (Cont.)

SQLJ

```
#sql iterator ShowResv(...);
ShowResv showResv;
#sql showResv = {CALL showNumReservations};
while (showResv.next() )  ...
```

- Stored procedures do not have to be written in SQL

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


Summary

- Embedded SQL allows execution of parameterized 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
- SQLJ: static model, queries checked at compile time
- Stored procedures execute application logic directly at the server side