Database Application Development

Chapter 6

Why Is This Important?

- So far, accessed DBMS "directly" through client tools
  - Great for interactive use
- How can we access the DBMS from a program?
- Need an interface between programming language and DBMS
- Many different options
- Our focus: JDBC

Overview

- SQL in application code
- Embedded SQL
- Cursors
- JDBC
- 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.
- 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 (Contd.)

- Impedance mismatch:
  - SQL relations are (multi-) sets of records, with no a priori bound on the number of records. No such data structure existed traditionally in procedural programming languages such as C.
  - SQL supports a mechanism called a cursor to handle this.
    - Cursor essentially is a more powerful iterator

Embedded SQL

- Approach: Embed SQL in the host language.
  - A preprocessor converts 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 in C: 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.
- Can also modify/delete tuple pointed to by a cursor.

Cursor: Get names of sailors who 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
```

- Cannot replace S.sname by, say, S.sid in the ORDER BY clause above (Why?)
- 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;
```


Database APIs: Alternative to embedding

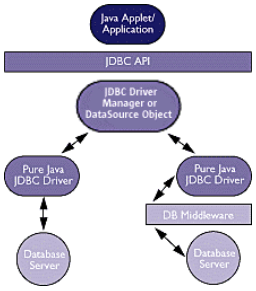
- Rather than modify compiler, add library with database calls (API)
  - Advantage: executable is also DBMS-independent
  - Embedded is SQL DBMS-independent only at source-code level
- Pass SQL strings from language, present result sets in a language-friendly way
  - Sun's JDBC: Java API
- Supposedly DBMS-neutral
  - A driver traps the calls and translates them into DBMS-specific code
  - Driver loaded dynamically and on-demand
  - Database can be across a network

JDBC Architecture Components

- Application
  - Initiates and terminates connections, submits SQL statements
- Driver manager
  - Loads JDBC driver, passes JDBC calls from app to correct driver
- Driver
  - Connects to data source, transmits requests and returns/translates results and error codes
- Data source (DBMS)
  - Processes SQL statements

JDBC Architecture (Pure Java)

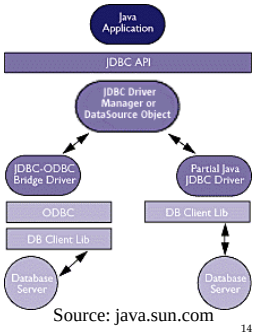
- Left side: type 4 driver
  - Allows direct call from client to DBMS, pure Java
  - Converts JDBC calls into network protocol used by DBMS
- Right side: type 3 driver
  - Translates JDBC calls into middleware protocol
  - Middleware translates this to DBMS protocol
  - Useful when connecting to many different DBMSes



Source: java.sun.com

JDBC Architecture (Not Pure Java)

- Left side: type 1 driver
  - JDBC access via ODBC drivers
  - Each client using the bridge must have ODBC binary code
- Right side: type 2 driver
  - Converts JDBC calls into calls on the DBMS client API
  - Needs binary code on client machine



Source: java.sun.com

JDBC Classes and Interfaces

- Steps to submit a database query:
  - Load the JDBC driver
  - Connect to the data source
  - Execute SQL statements
- Important: make sure you include the driver in the classpath
  - Driver jar file sqljdbc4.jar needs to be in the classpath
  - Should be there by default on Windows lab machines

Connecting to A DBMS

```
private Connection getDBConnection() {
        Connection con = null;
        try {
                // Load the driver
                Class.forName(myDbDriver).newInstance();

        } catch (InstantiationException e) { e.printStackTrace();
        } catch (IllegalAccessException e) { e.printStackTrace();
        } catch (ClassNotFoundException e) { e.printStackTrace(); }

        try {
                String connectionURL = "myURL";
                con = DriverManager.getConnection(connectionURL);

        } catch (SQLException e) { e.printStackTrace(); }

        return con;
}
```


Connection Data

- MSFT JDBC driver for SQL Server
  - dbDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
  - connectionURL = "jdbc:sqlserver://address:1433;" + "databaseName=XYZ;user=YOU;password=SECRET;";
- In the JDBC API 4.0, the DriverManager.getConnection method is enhanced to load JDBC drivers automatically.
  - Do not need to call the Class.forName method to register or load the driver when using the sqljdbc4.jar class library.
- When the getConnection method of the DriverManager class is called, an appropriate driver is located from the set of registered JDBC drivers.
  - sqljdbc4.jar file includes "META-INF/services/java.sql.Driver" file, which contains the com.microsoft.sqlserver.jdbc.SQLServerDriver as a registered driver.

Connections in JDBC

- Notice: We interact with a data source through sessions.
- Each connection identifies a logical session.
  - JDBC URL: jdbc:<subprotocol>:<otherParameters>
- Multiple users: each has his/her own session(s)

Important Imports For JDBC

- import java.sql.Connection;
- import java.sql.DriverManager;
- import java.sql.ResultSet;
- import java.sql.SQLException;
- import java.sql.Statement;

Running A Simple SQL Query


```
public List getSpeciesNames() {
        Connection con = getDBConnection();
        List species = new ArrayList();

        try {
                Statement S = con.createStatement();
                // Get query results
                ResultSet rs = S.executeQuery(
                                "SELECT DISTINCT " + speciesColName
                                + " FROM " + scoresTableName);
                // Copy results into list
                while (rs.next()) {
                        String speciesName = rs.getString(speciesColName);
                        species.add(speciesName);
                }
                rs.close();
                con.close();
        } catch (SQLException e) { e.printStackTrace(); }
        return species;
}
```


Connection Interface

- Can set auto-commit mode
  - Auto-commit on: each statement considered its own transaction, no need for explicit commit()
  - boolean **getAutoCommit**(),
    void **setAutoCommit**(boolean autoCommit)
- Can set transaction isolation level
  - Connection.TRANSACTION_READ_UNCOMMITTED, Connection.TRANSACTION_READ_COMMITTED, Connection.TRANSACTION_REPEATABLE_READ, or Connection.TRANSACTION_SERIALIZABLE
  - int **getTransactionIsolation**(),
    void **setTransactionIsolation**(int level)
- Isolation, auto-commit covered later, for now use default

Connection Interface (Contd.)

- Better performance possible for read-only access
  - boolean **isReadOnly**(),
    void **setReadOnly**(boolean readOnly)
- Check whether connection is still open
  - boolean **isClosed**(),
    void **close**()
- Commit or abort transaction
  - Use only when autoCommit is false
  - void **commit**(),
    void **rollback**()

Statement Interface

- Used to execute SQL statement and return its results
  - execute(String sql) to execute any SQL statement
  - executeQuery(String sql) to obtain single ResultSet object
  - executeUpdate(String sql) for INSERT, UPDATE, or DELETE
- Sub-interface PreparedStatement
  - Precompiled SQL statement for efficiently executing a statement multiple times.
  - Structure fixed, parameters determined at runtime
    - PreparedStatement pstmt = connection.prepareStatement("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");
    - pstmt.setBigDecimal(1, 153833.00); pstmt.setInt(2, 110592);
  - Sub-interface CallableStatement
    - For calling SQL stored procedures through standard way for all RDBMSes

SQL 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
    - Only final result is returned to Java app

Example Stored Procedure


```
CREATE PROCEDURE getReservations
        @Name varchar(50),
AS
        SET NOCOUNT ON;
        SELECT bid, date
        FROM Reserves R, Sailors S
        WHERE R.sid = S.sid
                AND S.name = @Name;
GO
```

Syntax for SQL Server, will be different for other DBMS

Calling A Stored Procedure


```
EXECUTE getReservations 'Joe';
-- Or
EXEC getReservations @Name = 'Joe';
GO
-- Or, if this procedure is the first statement within a batch:
getReservations 'Joe';
```


Stored Procedure with Output Parameters


```
CREATE PROCEDURE getReservationCnt
        @SailorID int,
        @ResCnt int OUT
AS
        SET NOCOUNT ON;
        SET @ResCnt = (SELECT COUNT(*)
                FROM Reserves R
                WHERE R.sid = @SailorID);
GO
```


Calling Stored Procedures with Output


```
DECLARE @ResCnt int
EXECUTE getReservationCnt 101, @ResCnt OUT
PRINT 'The sailor made '
        + RTRIM(CAST(@ResCnt AS varchar(20)))
        + ' reservations.'
```


Calling Stored Procedures from JDBC


```
CallableStatement cs = null;
try {
        // Procedure without parameters
        cs = con.prepareCall("{call myStoredProcName}");
        cs.execute();
        // Procedure with input parameters only
        cs = connection.prepareCall("{call getReservations(?)}");
        cs.setString(1, "Joe");
        cs.execute();
        // Procedure with input and output parameters
        cs = connection.prepareCall("{call getReservationCnt(?, ?)}");
        cs.setInt(1, 101);
        cs.registerOutParameter(2, Types.INT);
        // For parameters that are used for both input and output,
        // have both the set and registerOutParameter statement
        cs.execute();
        int result = cs.getInt(2);
} catch (SQLException e) { }
```


ResultSet Interface

- Maintains a cursor, initially positioned before first row
- Next() method advances cursor, returns false if no more rows
- Default: not updateable, cursor can only move forward
  - Can be changed, of course
  - Can update database by updating ResultSet
    - Update column values of current row or delete entire row
    - Insert new row by setting values of special "insert row"
- GetString(), getBoolean() etc. to retrieve values in columns
  - Access through column number or name

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()

Exceptions and Warnings

- Most of java.sql can throw an SQLException if an error occurs.
- SQLWarning is a subclass of SQLException
  - Not as severe
  - Not thrown and their existence has to be explicitly tested

Exceptions and Warnings (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" + ",
    ");
    }
}
```


Summary

- Embedded SQL allows execution of parameterized static 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
- Stored procedures execute application logic directly at the server