09. Programming Databases

API based access to databases

Another approach for accessing remote databases in host program is API based access. API based access are more canonical, powerful and popular way of manipulating remote databases.

Over the period of time, various API standards have evolved in most popular programming environments. Open Database Connectivity (ODBC) was first initiative by Microsoft towards standardization. Current popular standards are JDBC, Python DB-API, ADO.Net.

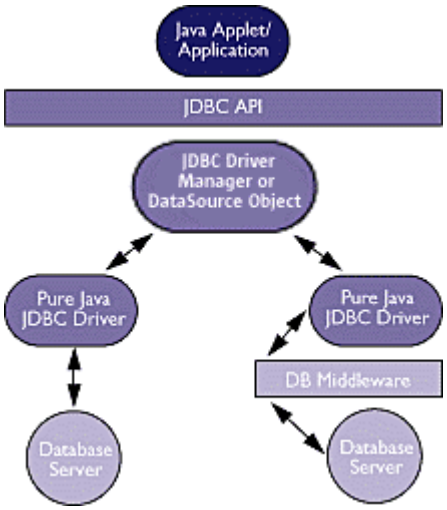
Like in case of embedded SQL, in this approach too most of the time we perform one or more of following operations using API-

1. Connect to remote Database
2. Submit SQL statements
3. If executed query returns a row-set, iterate through the result-set and do some processing for each row.

Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is Application Programming Interface (API) for accessing relational databases. API is basically, often, a *set of interface specifications* for certain tasks.

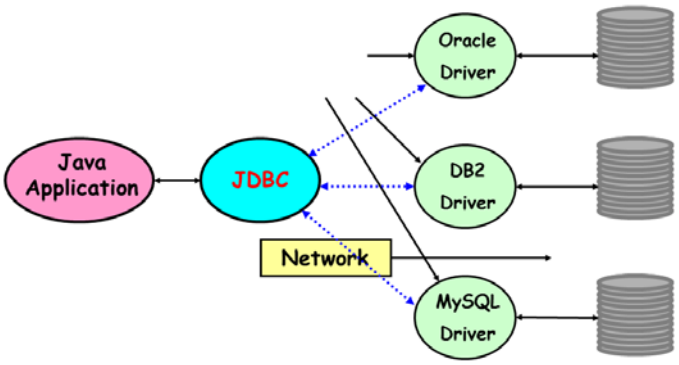
There are following things done in a java application accessing databases-



While API classes are often abstract, concrete object classes are available as part of JDBC Driver.

JDBC API is the industry standard for RDBMS-Independent database connectivity from a java program. Diagram here depicts the JDBC architecture.

In the java applications are programmed for JDBC API rather than any specific RDBMS. This approach makes the programs portable across the RDBMS. It only requires changing of JDBC driver for new RDBMS. Diagram below sketches out logical



You can interpret role of JDBC Driver as a wrapper as per JDBC specifications over the RDBMS specific calls to databases.

Common objects in JDBC

1. Connection
2. Statement
3. ResultSet
4. DatabaseMetaData

Let us have a quick introduction to these objects-

Connection object

Connection Object: main contact reference to remote database. An instance of connection object is received by sending getConnection request to DriverManager class. Before doing any work on database, getting connection object reference is the first task. Database URL, user-name and password are parameters to getConnection request.

Primary responsibilities of Connection object are-

- To control transaction behavior
- To create statements for executing SQL statements
- To finish interaction with a database by closing the connection

For example, consider code fragment given below, lines 13 through 17 opens a connection with test database.

Statement Object

Statement Objects are used to forward SQL queries to DBMS. Steps of usage are following-

1. Create a Statement Object by sending createStatement message to connection object. (line #19 in code below)
2. Call its execute method
3. If the SQL statement is one of CREATE, INSERT, DELETE, UPDATE: apply executeUpdate method onto statement object otherwise call executeQuery method.

Line no 21 in example code below executes a SELECT query over the company database.

```
13  Class.forName( "org.postgresql.Driver" );
14  DB_URL = "jdbc:postgresql://10.100.71.21/test";
15  user = "test";
16  passwd = "test";
17  Connection con =  DriverManager.getConnection(DB_URL, user, passwd);
18
19  Statement stmt = con.createStatement();
20
21  ResultSet rs = stmt.executeQuery("SELECT * FROM company.Employee AS e "
22          + "JOIN company.Employee AS s ON (e.superssn=s.ssn) WHERE e.dno=4");
23
24  while ( rs. next() ) {
25      String fname, ss, sname="";
26      fname = rs.getString("fname");
27      ss = rs.getString("superssn");
28      System.out.println( fname +" " +  ss + " " + sname);
29  }
```

ResultSet Object

Primarily a iterator on row-set returned by a SELECT query. It has following iterating methods. Most should be self-explanatory-

boolean absolute(int row) -- moves the cursor to the specified row
void afterLast()
void beforeFirst()
void close() -- releases the database resources
boolean first()
XType getXType(int columnIndex)
XType getXType(String column-name)
boolean next()
boolean previous()

Most important set of methods for fetching data from a row is getXXX; where XXX denotes a type. Parameter to method is column name. Methods getString, getDouble, getDate are for getting values

from String, Double, and Date data type columns. Lines 26 and 27 in code shows fetching of fname and superssn attributes.

Scrollable and Updatable Result Set

There meaning is as expected and same as what we have understood in other contexts. Code below is sample code and should be self-explanatory-

```
3  con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
4      ResultSet.TYPE_FORWARD_ONLY,
5      ResultSet.CONCUR_UPDATABLE);
6
7  String query = "select SSN, SALARY from EMPLOYEE where dno=4";
8  ResultSet rs = stmt.executeQuery( query );
9  …
10 while ( rs.next() )
11 {
12     double salary = rs.getInt("SALARY");
13     rs.updateDouble("SALARY", salary*1.1);
14     rs.updateRow();
15 }
```

There are two special types of Statement objects: Prepared Statement and Callable Statement

Prepared Statements

DBMS prepares a query execution plan before executing a query. A query plan is basically a sequence of physical operations that are to be performed for getting the result of the query. Query plan also includes algorithms that are to be used. Preparing a query primarily implies preparation of query execution plan.

Sometimes a query statements needs to be executed repeatedly, typically in a loop. In such case we can tell DBMS, a statement to be prepared once, and keep it ready for further requests. Below is an example; a java method that updates from an ArrayList object to the Database.

```java
public void gradeUpload(String acad_yr, String sem, String course_no,
        ArrayList<Grade> marks) throws SQLException {

    PreparedStatement updateMarks = null;

    String updateString = "UPDATE registers SET grade = ? " +
            "WHERE AcadYr = ? AND Semester = ? AND StudentID = ?;";

    try {
        Connection conn = DBConnection.getInstance().getConnection();
        conn.setAutoCommit(false);
        updateMarks = conn.prepareStatement( updateString );
        for (Grade grade : marks ) {
            updateMarks.setString(1, grade.getGrade());
            updateMarks.setString(2, acad_yr);
            updateMarks.setString(3, sem);
            updateMarks.setString(4, grade.getID());
            updateMarks.executeUpdate();
            conn.commit();
        }
    }
    catch (SQLException e ) { }
    catch (Exception e) {}
    finally {
        updateMarks.close();
```

It should be noted that queries are often parametric in case of prepared statements. Parameters are specifying by "?" as shown in example above and below.

Advantages of prepared statements -

- Faster execution – as queries are pre-compiled (and prepared) queries.
- Avoids many run-time errors that may occur if queries are created by string concatenation
- One of the main technique of avoiding "SQL injections"

Below is yet another example that inserts a number of rows in works on relations –

```java
String strSQL = "INSERT INTO company.works_on VALUES(?,?,?)";
PreparedStatement pst = conn.preparedStatement( strSQL );
while ( true ) {
    mssn = ... ; mpno = ... ; mhrs = ... ;
    pst.setInt(1, mssn);
    pst.setInt(2, mpno);
    pst.setFloat(3, mhrs);
    int nrows = pst.executeUpdate();
    System.out.println("" + nrows + " row(s) added");
}
```

Callable Statements

Callable Statements are used to execute stored procedures. Here is a sample code executing stored procedure, should be self-explanatory-

```
12  Connection con = ... ;
13  CallableStatement cstmt = con.prepareCall("{call test12(?,?)}");
14  cstmt.setInt(1, 5);
15  cstmt.setInt(2, 15);
16  cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
17  cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
18  cstmt.executeUpdate();
19  int x = cstmt.getInt(1);
20  int y = cstmt.getInt(2);
21  System.out.println("" + x + "," + y);
```

Database Metadata

Database Metadata object allows us exploring *database schema*. It has following common operations –

- getTables
- getColumns for specified table
- get Primary Key
- get Foreign Key
- get other Constraints

etc. etc.

More constraints using stored procedures

Some constraints are complex to be defined in SQL-DDL, and sometimes it is not possible to be defined in standard DDL. Code below should be self-explanatory. It adds a constraint that an employee can work project controlled by its home department-

```
CREATE TABLE  works_on (
    essn decimal(9,0),
    pno  integer,
    hours  integer,
    PRIMARY KEY  (essn, pno),
    CONSTRAINT work_proj_check
        CHECK(valid_work_dept(pno,essn))
)
```

CREATE ASSERTION command

CREATE ASSERTION is another command that is used for creating complex constraints. Following is an example.

```
CREATE ASSERTION salary_constraint
    CHECK ( NOT EXISTS ( SELECT * FROM employee e NATURAL JOIN
        department d JOIN employee m ON m.ssn = d.mgrssn
        WHERE e.salary > m.salary)
);
```

Unfortunately, many major commercial DBMS do not support assertions or queries in CHECK. However the same functionality can be accomplished by triggers and stored procedures.

PostgreSQL too, does not support assertions and queries in CHECK. It, however provides CREATE CONSTRAINT TRIGGER, instead, and serves the same purpose.