UNIT-4 [Around 5 marks]

Database Connectivity

JDBC introduction imp

⊛JDBC:

JDBC stands for Java Database Connectivity, which is standard Java API for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC library includes API's for each of tasks commonly associated with database usage:

→ Making a connection to a database
→ Creating SQL or MySQL statements
→ Executing SQL or MySQL queries in the database
→ Viewing and Modifying the resulting records.

JDBC Architecture:

JDBC Architecture consists of two layers:

JDBC API: This provides the application-to-JDBC Manager connection.

JDBC Driver API: This supports the JDBC Manager-to-Driver connection.

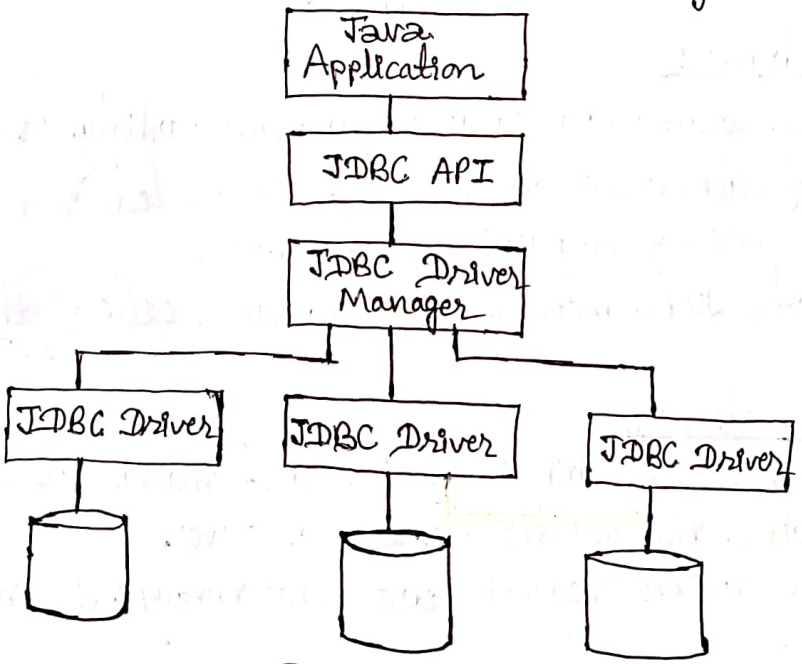

Fig: JDBC Architecture

Common JDBC Components:

The JDBC API provides the following interfaces and classes:

Driver Manager: This class manages a list of database drivers. Matches connection requests from java application with proper database driver.

Driver: This interface handles the communications with the database server.

Connection: This interface consists all methods for contacting a database. All communication with database is done through connection object only.

Statement: We use objects created from interface to submit the SQL statements to the database.

ResultSet: These objects hold data retrived from database after we execute an SQL query.

SQLException: This handles any errors that occur in database application.

JDBC Driver Types: [Imp]

There are 4 types of JDBC drivers:

Type1: JDBC-ODBC Bridge:

→ To use a Type 1 driver in a client machine, an ODBC driver should be installed and configured correctly.
→ This driver does not directly interact with the database. To interact, it needs ODBC driver. It converts JDBC method class into ODBC method class.
→ It can be used to connect to any type of databases.

Type2: Native-API driver:

→ Type 2 drivers are written partially in Java and partially in native code.
→ The Native-API of each database should be installed in the client system before accessing a particular database.
→ This driver converts JDBC method calls into native calls of the database API.

Type3: Net Pure Java driver:

→ In type3 driver, the JDBC driver on the client machine uses the socket to communicate with the middleware at the server.
→ The client database access requests are sent through the network to the middleware.
→ Type-3 drivers are fully written in Java, hence they are portable.

Type4: Pure Java driver:

→ This driver interact directly with database.
→ No client-side or server-side installation.
→ It is fully written in Java, hence they are portable drivers.

#Benefits of JDBC:

→ It enables enterprise applications to continue using existing data even if the data is stored on different database management systems.
→ The combination of the java API and the JDBC API makes the database transferable from one vendor to another without modifications in the application code.
→ JDBC is cross-platform or platform independent.
→ With JDBC it is easy to deploy and economical to maintain.

⊛ JDBC Configuration:

Steps to connect to database if asked write in short about ~~JDBC~~ JDBC configuration, connection, statement & Result set.

Let we are using mysql database, now to connect Java application with MySQL database, we need to follow following steps:

1. Driver class: The driver class for the mysql database is **com.mysql.jdbc.Driver.**

2. Connection URL: The connection URL for mysql database is **jdbc:mysql://localhost:3306/roshan** where jdbc is API, mysql is database, localhost is server name on which mysql is running, 3306 is port number and roshan is database name.

3. Username: The default username for the mysql database is **root.**

4. Password: It is given by user at the time of installing mysql database.

⊛ Managing Connections:

Create the connection object with getConnection() method of DriverManager class.

```
Connection con = DriverManager
getConnection(url:"jdbc:mysql://localhost:3306/roshan", user:"root",
password:"");
```

(i.e, connection url //dburl/dbname; username; password)

⊛ Statement:

Create the Statement object with createStatement() method of Connection interface.

```
Statement stmnt = con.createStatement();
```

⊛ Result Set:

The executeQuery() method of Statement interface is used to execute queries to database.

```
String select = "SELECT * FROM roshan";
ResultSet rs = stmnt.executeQuery(select);
```

Note:

- Use executeQuery(String) method to retrive data from database. It returns ResultSet object.
- Use executeUpdate(String) method to insert, update, delete data in database. It returns int for the number of affected rows.

⊛. Close Connection:

The close() method of Connection interface is used to close the connection.

```
con.close();
```

⊛. Fetching multiple results:

```
System.out.println("ID \t NAME \t SALARY");
while(rs.next()){
    //rs.next() checks whether next row is available
    System.out.print(rs.getInt(s: "id")+"\t");
    System.out.print(rs.getString(s:"name")+"\t");
    System.out.print(rs.getFloat(s: "salary"));
}
```

rs is object created by us for ResultSet

⊛. SQL Exceptions:

Exception handling allows us to handle exceptional conditions such as program-defined errors in a controlled fashion. to maintain the flow of program and to avoid program from crash. JDBC Exception handling is very similar to Java Exception. but for JDBC, the most common exception we will deal with is **java.sql.SQLException**.

When exception occurs, an object of type SQLException will be passed to catch clause. The passed SQLException object has following methods available for retriving additional information about the exception:

- getErrorCode()
- getMessage()
- getSQLState()
- getNextException()
- printStactTrace()

etc.

Very less chance but If example asked write similar example of exception as we write in unit 1 with try, catch including sql query in try block like: sql="Update Employees SET age=?"

catch block should throw output as: "Invalid SQL Statement".

⊛. Scrollable result sets: & Updatable result sets:

A scrollable updateable result set maintains a cursor which can both scroll and update rows. Use scrollable insensitive result sets. To create a scrollable insensitive result set which is updatable, the statement has to be created with concurrency mode:

ResultSet.CONCUR_UPDATABLE and type
ResultSet.TYPE_SCROLL_INSENSITIVE

Example:

```
Statement stmnt=con.createStatement(ResultSet.TYPE_SCROLL_
  INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
String sql="SELECT *"+"FROM teachers";
ResultSet rs=stmnt.executeQuery(sql);
rs.absolute(i:2); //update the second row
float. newSalary=rs.getFloat(s:"salary")+2000;
rs.updateFloat(s:"salary",newSalary);
rs.updateRow();
rs=stmnt.executeQuery(sql);

//Output multiple results
System.out.println("ID \t NAME\t SALARY");
while(rs.next()){
        System.out.print(rs.getInt(s:"id")+"\t");
        System.out.print(rs.getString(s:"name")+"\t");
        System.out.println(rs.getFloat(s:"salary"));
}
```

⊛ Row Set:

We have ResultSet, which have certain limitations as it explicitly requires Connection and Statement explicitly. All those extra efforts can be reduced by using RowSet, which simplifies our program.

Example:

```
Class.forName("com.mysql.jdbc.Driver");
JdbcRowSet rs=RowSetProvider.newFactory().createJdbcRowSet();
```

⊛ DDL and DML Operations using Java:

Data Definition Language (DDL) and Data Manipulation Language (DML) together forms a Database Language. The basic difference between DDL and DML is that DDL is used to specify the database schema data structure. On the other hand, DML is used to access, modify or retrive the data from the database.

DDL query using java:

```
String createTable = "CREATE TABLE" + "test(" + "id INT(10), " +
    "name VARCHAR(20), " + "price FLOAT(20), )";
Stmnt.execute(createTable);
```

DML query using java:

```
String sql = "insert into test(name, address, price) values" + "
        ('Roshan', 'Mahendranagar', 1000000)";
Stmnt.executeUpdate(sql);
stmnt.executeQuery(sql);
```

⊛ Prepared Statements: [Imp]

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. The performance of the application will be faster if we use PreparedStatement interface because query is compiled only once. To get the instance of PreparedStatement, the prepareStatement() method of connection interface is used.

Syntax:

```
public PreparedStatement prepareStatement(String query) throws SQLException{}
```

Example of PreparedStatement interface that inserts the record:

```
create table emp(id number(10), name varchar(50));
PreparedStatement stmt = con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);
stmt.setString(2,'Roshan');
int i = stmt.executeUpdate();
```

con → connection object

```
rs.setURL("jdbc:mysql://localhost:3306/roshandb");
rs.setUsername("root");
rs.setPassword("root");

rs.setCommand("select * from teachers");
rs.execute();
```

⊛ Cached Row Set:

A CachedRowSet object is a container for rows of data that caches its rows in memory, which makes it possible to operate without keeping the database connection open all the time.

A CachedRowSet object makes use of a connection to the database only briefly: while it is reading data to populate itself with rows, and again while it is committing changes to the underlying database. So the rest of the time, a CachedRowSet object is disconnected, even while its data is being modified. Hence it is called disconnected row set.

⊛ JDBC Transactions:

→ A transaction is a set of actions to be carried out as a single, atomic action. Either all of the actions are carried out, or none of them are.

→ The classic example of when transactions are necessary is the example of bank accounts.

→ Let we need to transfer 10,000 rupees from one account to the other.

→ We do so by subtracting Rs 10,000 from the first account, and adding Rs 10,000 to the second account.

→ If this process fails after we have subtracted Rs 10,000 from the first account, the Rs 10,000 are never added to second account.

→ The money is lost in cyber space. To solve this addition and subtraction are grouped into a transaction.

→ If the subtraction succeds, but the addition fails, we can "rollback" the first subtraction to left database in same state as it was before.

⊛. SQL Escapes:

Escape Sequence	Usage
\n	A newline character
\b	A backspace character
\r	A carriage return character
\\	A backslash character
\%	A percentage "%" character. Useful in LIKE clauses.
_	An underscore character. Useful in LIKE clauses.
\t	A tab character
\'	A single quote
\"	A double quote

Q. Write a Java program using JDBC to extract name of those students who live in Kathmandu district, assuming that the student table has four attributes (ID, name, district, and age). [asked in 2072] [less imp but we can view soln once from collegenote]

imp In past syllabus. Lesser imp in new syllabus ↔ can be escaped if found hard

Note: In this unit we should know how to configure, connect, and execute sql operations in database which is imp, pratically and exam point of view. For this once have a view of soln of above Q. question.

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952