**Core Java-JDBC**

**Q1-> What is Java JDBC?**

**Ans->** JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and
* Thin Driver

We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API are given below:

* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

A list of popular *classes* of JDBC API are given below:

* DriverManager class
* Blob class
* Clob class
* Types class

**Q2-> What are the steps to Java Database Connectivity?**

|  |
| --- |
| **Ans->**There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |

**1) Register the driver class**

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

Syntax of forName() method

**public** **static** **void** forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

Class.forName("oracle.jdbc.driver.OracleDriver");

2) Create the connection object

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

Syntax of getConnection() method

1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
2. 2) **public** **static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

Example to establish connection with the Oracle database

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

3) Create the Statement object

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

Syntax of createStatement() method

**public** Statement createStatement()**throws** SQLException

Example to create the statement object

Statement stmt=con.createStatement();

4) Execute the query

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

5) Close the connection object

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

Syntax of close() method

1. **public** **void** close()**throws** SQLException

Example to close connection

con.close();

**Q3-> What is connection Interface?**

**Ans->**A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

### Commonly used methods of Connection interface:

**1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**2) public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

**3) public void setAutoCommit(boolean status):** is used to set the commit status. By default, it is true.

**4) public void commit ():** saves the changes made since the previous commit/rollback is permanent.

**5) public void rollback():** Drops all changes made since the previous commit/rollback.

**6) public void close():** closes the connection and Releases a JDBC resources immediately.

## **Connection Interface Fields**

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

**TRANSACTION\_NONE**: No transaction is supported, and it is indicated by this constant.

**TRANSACTION\_READ\_COMMITTED**: It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.

**TRANSACTION\_READ\_UNCOMMITTED**: It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.

**TRANSACTION\_REPEATABLE\_READ**: It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.

**TRANSACTION\_SERIALIZABLE**: It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

**Q4-> Write an example to store an image in Oracle.**

**Ans->**import java.sql.\*;

import java.io.\*;

public class InsertImage {

public static void main(String[] args) {

try{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");

ps.setString(1,"sonoo");

FileInputStream fin=new FileInputStream("d:\\g.jpg");

ps.setBinaryStream(2,fin,fin.available());

int i=ps.executeUpdate();

System.out.println(i+" records affected");

con.close();

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

}

}

**Q5->What is Batch Processing in JDBC?**

**Ans->**Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast. It is because when one sends multiple statements of SQL at once to the database, the communication overhead is reduced significantly, as one is not communicating with the database frequently, which in turn results to fast performance.

The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Advantage of Batch Processing

Fast Performance

Methods of Statement interface

The required methods for batch processing are given below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| void addBatch(String query) | The addBatch(String query) method of the CallableStatement, PreparedStatement, and Statement is used to single statements to a batch. |
| int[] executeBatch() | The executeBatch() method begins the execution of all the grouped together statements. The method returns an integer array, and each of the element of the array represents the updated count for respective update statement. |
| boolean DatabaseMetaData.supportsBatchUpdates() throws SQLException | If the target database facilitates the batch update processing, then the method returns true. |
| void clearBatch() | The method removes all the statements that one has added using the addBatch() method. |

Example of batch processing in JDBC

Let's see the simple example of batch processing in JDBC. It follows following steps:

* Load the driver class
* Create Connection
* Create Statement
* Add query in the batch
* Execute Batch
* Close Connection