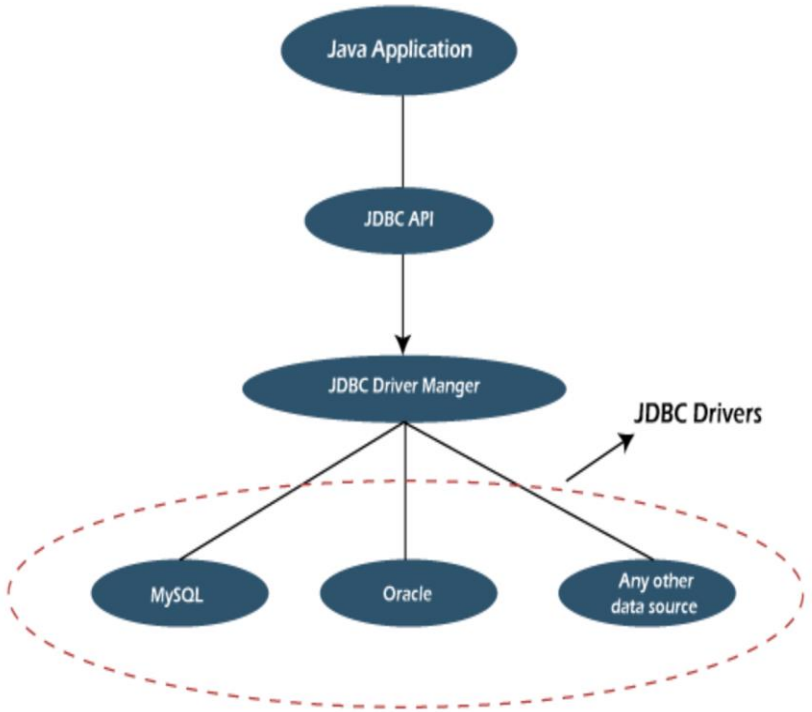
Unit 2 : Database Programming

The Design of JDBC :

- **Java Database Connectivity** (JDBC) is an Application Programming Interface (API), from Sun microsystem that is used by the Java application to communicate with the relational databases from different vendors.

- JDBC and database drivers work in tandem to access spreadsheets and databases. **Design of JDBC** defines the components of JDBC, which is used for connecting to the database.

Components of JDBC :



Database Programming

1. JDBC API:

It provides various methods and interfaces for easy communication with the database.It provides two packages as follows which contains the java SE and java EE platforms to exhibit WORA(write once run everywhere) capabilities.

1. java.sql.*;
2. javax.sql.*;

2. JDBC Driver manager:

It loads database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

3. JDBC Test suite:

It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.

4. JDBC-ODBC Bridge Drivers:

It connects database drivers to the database.This bridge translates JDBC method call to the ODBC function call.It makes the use of

sun.jdbc.odbc package that includes native library to access ODBC characteristics.

JDBC Driver Types :

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

- JDBC bridge is used to access ODBC drivers installed on each client machine.

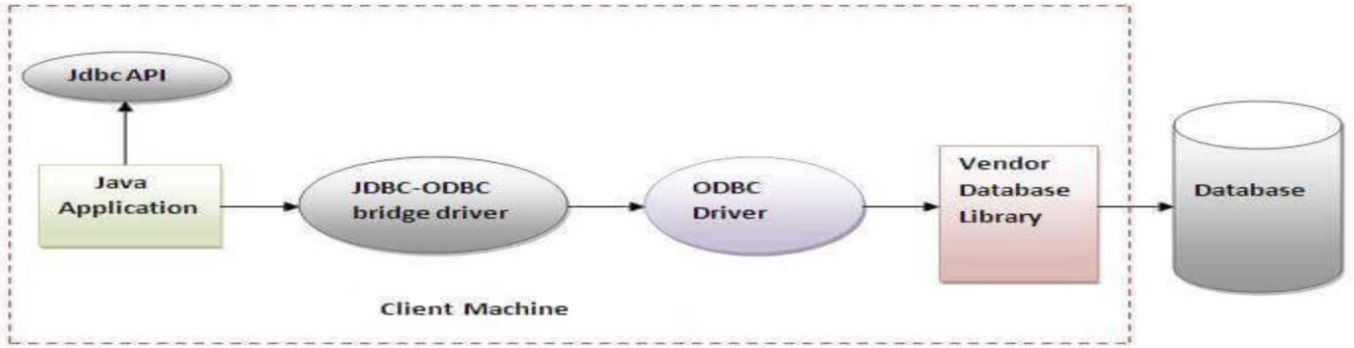


Figure- JDBC-ODBC Bridge Driver

Advantages:

1. easy to use.
2. can be easily connected to any database.

Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

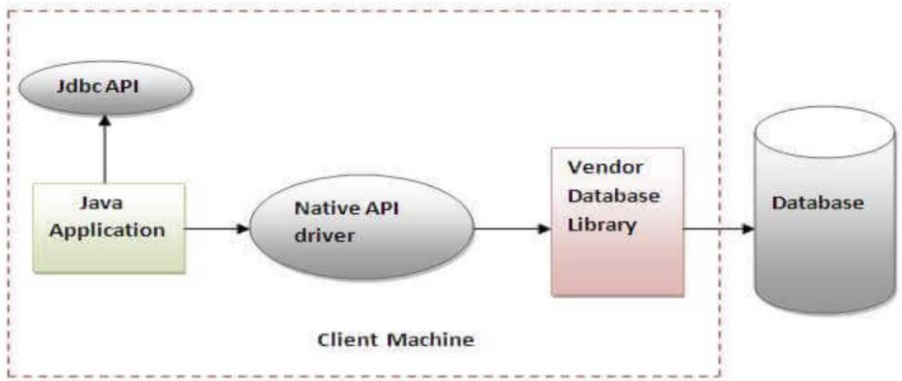


Figure- Native API Driver

Advantages:

1. performance upgraded than JDBC-ODBC bridge driver.

Disadvantages:

1. The Native driver needs to be installed on the each client machine.
2. The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

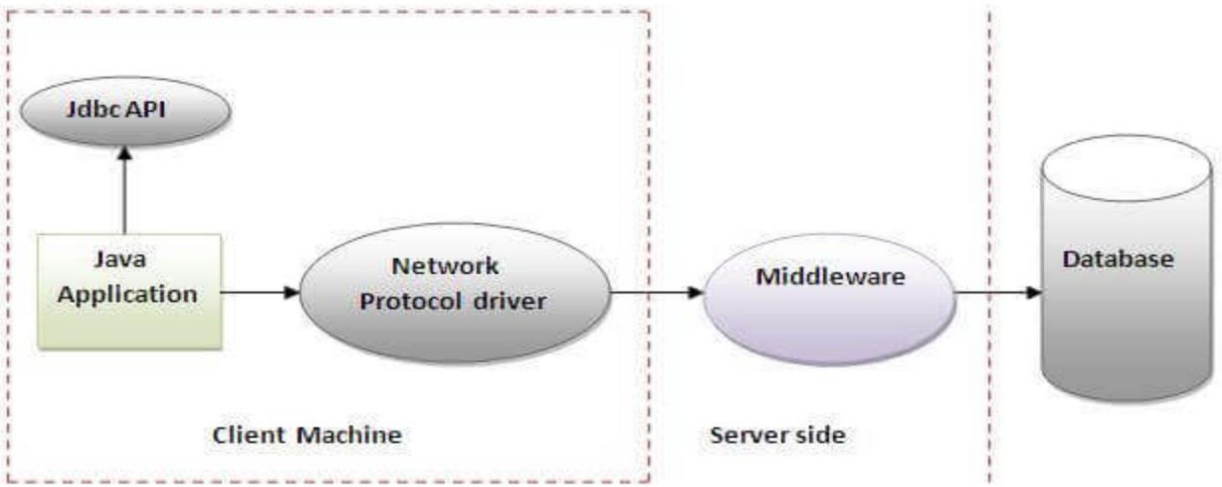


Figure- Network Protocol Driver

Advantages:
1. No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:
1. Network support is required on client machine.
2. Requires database-specific coding to be done in the middle tier.
3. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

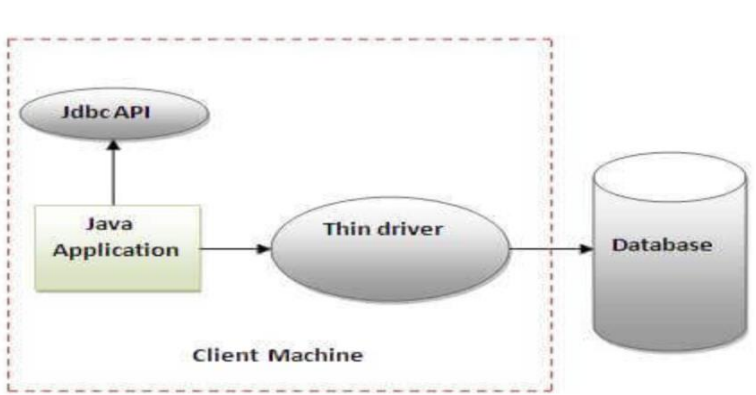


Figure- Thin Driver

Advantages:

1. Better performance than all other drivers.
2. No software is required at client side or server side.

Disadvantages:

1. Drivers depend on the Database.

Which Driver should be Used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4 (**Thin driver)**.
- If your Java application is accessing multiple types of databases at the same time, type 3 (**Network Protocol driver)** is the preferred driver.
- Type 2 drivers (**Native-API driver**) are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver (**JDBC-ODBC bridge driver**) is not considered a deployment-level driver, and is typically used for development and testing purposes only.

SQL

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

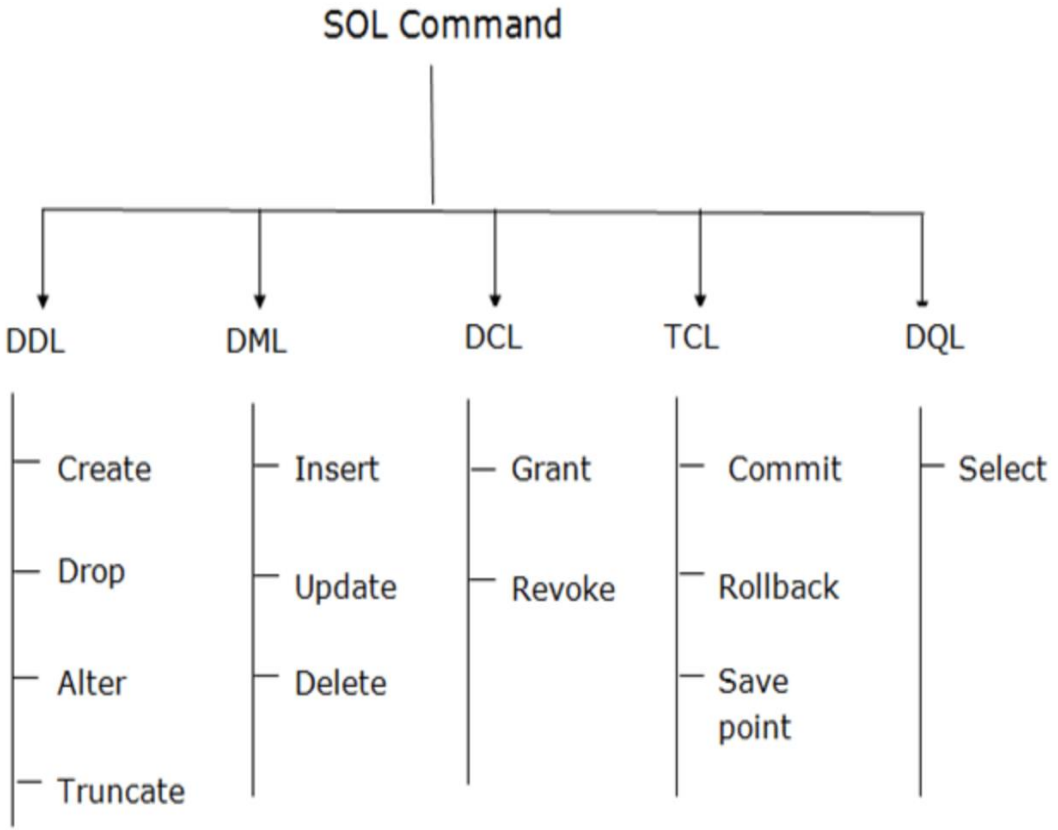
SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

1. Data Definition Language (DDL)
2. Data Manipulation Language(DML)
3. Data Control Language(DCL)
4. Transaction Control Language(TCL)
5. Data Query Language(DQL)



1. Data Definition Language (DDL)

CREATE : It is used to create a new database and table.

Syntax :

a. CREATE DATABASE *databasename*;

b. CREATE TABLE *table_name* (
column1 datatype,
column2 datatype,
column3 datatype,
....
);

DROP: It is used to delete objects such as the table, view, index, database.
A drop statement cannot be rolled back.

Syntax:

a. DROP TABLE table_name;
b. DROP DATABASE *databasename*;

ALTER :
Alter command is used for altering the table structure, such as :

- to add a column to existing table
- to change datatype of any column or to modify its size.
- to drop a column from the table.

Syntax:

1. Add new column :
ALTER TABLE table_name
ADD column_name datatype;

2. Modify an existing Column :

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;

3. Drop column name :
ALTER TABLE table_name
DROP COLUMN column_name;

TRUNCATE :
- command removes all the records from a table. But this command will not destroy the table's structure.

Syntax:
TRUNCATE TABLE table_name

2. Data Manipulation Language(DML)

INSERT:
Insert command is used to insert data into a table.

Syntax:
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

UPDATE :
It is used to update any record of data in a table.

Syntax:
UPDATE table_name SET column_name = new_value WHERE some_condition;

Delete:
It is used to delete data from a table.

Syntax :
DELETE FROM table_name WHERE condition;

Data Query Language(DQL)

SELECT：

It is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the WHERE clause.

Syntax:
SELECT column1, column2, columnN FROM table_name;

Steps to develop/ create JDBC Application (JDBC Configuration)

Steps For Connectivity Between Java Program and Database

1. Import the JDBC packages
2. Loading and Registering a driver
3. Create Connection
4. Create a statement
5. Execute the Statement
6. Processing Result Set
7. Close the connection

1. Import the JDBC packages:

Import all the packages that contain the JDBC classes needed for database programming.

E.g.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
Import java.sql.Statement;
```

2. Loading and Registering a driver

- Provide the code to register your installed driver with your program.
- 2 ways to register driver :

1. Class.ForName()
2. DriverManager.registerDriver() method

Class.ForName() – It loads Jdbc driver directly.

However, Class.ForName() method is valid only for JDK- Compliant JVM. It is not valid for Microsoft JVM. In this case you can use DriverManager.registerDriver() .

3. Create Connection:

Once you loaded the driver, you can establish a connection to the database with static *getConnection*() method of the JDBC Driver Manager class.

Java provide three overloaded DriverManager.getConnection() methods :

DriverManager. *getConnection*(String url)
DriverManager. *getConnection*(String url, String username, String password)
DriverManager. *getConnection*(String url, Properties info)

- The *getConnection*() method return an object of the JDBC connection class.
- e.g.

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/world", "Admin", "Admin");

4. Create a statement :

- createStatement() method of JDBC connection object returns an object of JDBC statement class.

e.g.
Statement statement = con.createStatement();

5. Execute Statement :

- When you execute Statement objects, it generates ResultSet objects, which is a table of data representing a database ResultSet.

e.g.
ResultSet rs = statement.executeQuery(sql);

6. Processing Result Set :

- If you want to process the ResultSet to pull data out of the ResultSet and iterate though it. You can use the next() method of your ResultSet object to loop through the results. This method iterate through the result set row by row, detecting the end of the ResultSet when it is reached.
- e.g.
- **while** (rs.next()) {

  }

7. Close the connection :

- Finally, to end the database session you need to close all the database resources which immediately release the resources it's using.

e.g.

con.close();

E.g. 1. Select Demo:

```
package com.texas.crud;

import java.sql.*;

public class SelectDemo {

public static void main(String[] args) {

try {
Class.forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/texas";
String user = "Admin";
String password = "Admin";
String sql = "SELECT * FROM student;";

Connection con = DriverManager.getConnection(url, user, password);
Statement stm =con.createStatement();
ResultSet rs = stm.executeQuery(sql);

while(rs.next()) {
int id = rs.getInt("ID");
String firstName = rs.getString(2);
System.out.println("ID : "+ id+ " : FirstName : "+firstName);
}

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


}

}
```

2. Update Demo

```
package com.texas.crud;

import java.sql.*;

public class UpdateDemo {

public static void main(String[] args) {

try {
Class.forName("com.mysql.cj.jdbc.Driver");
String url = "jdbc:mysql://localhost:3306/texas";
String user = "Admin";
String password = "Admin";
String sql = "update student set FirstName=?,MiddleName=?,LastName=?,Dob=? where ID=?;";

Connection con = DriverManager.getConnection(url, user, password);
PreparedStatement pstm= con.prepareStatement(sql);
pstm.setString(1, "Sandeep");
pstm.setString(2, "");
pstm.setString(3, "Shrestha");
pstm.setString(4, "2021-05-01");
pstm.setInt(5, 3);

int res = pstm.executeUpdate();
if(res>0) {
System.out.println("Data Updated successfully!");
}
con.close();
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}



}
}
```

3. Delete Demo

```
package com.texas.crud;

import java.sql.*;

public class DeleteDemo {

public static void main(String[] args) {
try {
Class.forName("com.mysql.cj.jdbc.Driver");
String url = "jdbc:mysql://localhost:3306/texas";
String user = "Admin";
String password = "Admin";
String sql = "delete from student where ID=?;";
Connection con = DriverManager.getConnection(url, user, password);

PreparedStatement pstm =  con.prepareStatement(sql);
pstm.setInt(1, 2);

int res = pstm.executeUpdate();
if(res>0) {
System.out.println("Data Deleted Successfully!");
}

con.close();

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



}
}
```

4. Insert Demo

```
package com.texas.crud;

import java.sql.*;
public class InsertDemo {

public static void main(String[] args) {

try {
Class.forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/texas";
String user = "Admin";
String password = "Admin";
String sql = "insert into student (FirstName,MiddleName,LastName,Dob) values(?,?,?,?);";
Connection con = DriverManager.getConnection(url, user, password);

PreparedStatement pstm =  con.prepareStatement(sql);
pstm.setString(1, "EE");
pstm.setString(2, "aa");
pstm.setString(3, "BB");
pstm.setString(4, "2021-03-04");
int res = pstm.executeUpdate();
if(res>0) {
System.out.println("Data inserted successfully!");
}

con.close();

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


}

}
```

JDBC Statements :

The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database. There are different types of statements that are used in JDBC as follows:

1. Create Statement/ Statement
2. Prepared Statement
3. Callable Statement

1. Create a Statement:

Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

Syntax:

```
Statement statement = connection.createStatement();
```

Implementation: Once the Statement object is created, there are three ways to execute it.

1. boolean execute(String SQL): If the ResultSet object is retrieved, then it returns true else false is returned.

2. int executeUpdate(String SQL): Returns number of rows that are affected by the execution of the statement, used when you need a number for INSERT, DELETE or UPDATE statements.

3. ResultSet executeQuery(String SQL): Returns a ResultSet object. Used similarly as SELECT is used in SQL.

2. Prepared Statement : - Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

Syntax:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(columnNo,”Value");
ptstmt.setInt(columnNo,Value);
```

Implementation: Once the PreparedStatement object is created, there are three ways to execute it:

1. execute(): This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.
2. executeQuery(): Returns a ResultSet from the current prepared statement.

3. executeUpdate(): Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

3. Callable Statement - Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Syntax:

```
CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");
```


Implementation: Once the callable statement object is created

execute() is used to perform the execution of the statement.

Difference between Statement and PreparedStatement :

Statement	PreparedStatement
It is used when SQL query is to be executed only once.	It is used when SQL query is to be executed multiple times.
You can not pass parameters at runtime.	You can pass parameters at runtime.
Used for CREATE, ALTER, DROP statements.	Used for the queries which are to be executed multiple times.
Performance is very low.	Performance is better than Statement.
It is base interface.	It extends statement interface.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
We can not used statement for reading binary data.	We can used Preparedstatement for reading binary data.
It is used for DDL statements.	It is used for any SQL Query.
We can not used statement for writing binary data.	We can used Preparedstatement for writing binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

Difference between CallableStatement and PreparedStatement :

CallableStatement	PreparedStatement
It is used when the stored procedures are to be executed.	It is used when SQL query is to be executed multiple times.
You can pass 3 types of parameter IN, OUT, INOUT.	You can pass any type of parameters at runtime.
Used to execute functions.	Used for the queries which are to be executed multiple times.
Performance is very high.	Performance is better than Statement.
Used to call the stored procedures.	Used to execute dynamic SQL queries.
It extends PreparedStatement interface.	It extends Statement Interface.
No protocol is used for communication.	Protocol is used for communication.

Scrollable ResultSet in JDBC

- A scrollable ResultSet allows us to retrieve the data in forward direction as well as backward direction but no updating are allowed.
- This provides the ability to read the last, first, next, previous and absolute records.
- To create scrollable ResultSet, we must use a Statement/PreparedStatement object and provide scroll type to createStatement/prepareStatement method.

Syntax:

- Statement stmt = conn.**createStatement**(**Scrolltype**,**concurrency**);
- PreparedStatement pstmt = conn.**prepareStatement**(sql, **Scrolltype**, **concurrency**);

Types of Scrollable Resultset in JDBC

1. ResultSet.TYPE_FORWARD_ONLY

- Default option, in which cursor moves from start to end (forward only).

2. ResultSet.TYPE_SCROLL_INSENSITIVE

- Cursor can move through the data set in both forward and backward direction.
- If there are changes to be underlying data while moving through the dataset, they are ignored.

3. ResultSet.TYPE_SCROLL_SENSITIVE

- This is similar to the SCROLL_INSENSITIVE type, however for this type the dataset immediately reflects any changes to the underlying data.

ResultSet Concurrency

1. ResultSet.CONCUR_READ_ONLY
   - The ResultSet can not be used to update the database.

2. ResultSet.CONCUR_UPDATABLE
   - The ResultSet can be used to update the database.

Syntax (Scrollable ResultSet)

Statement stmt = con.createStatement(ResultSet.**TYPE_SCROLL_INSENSITIVE**, ResultSet.**CONCUR_READ_ONLY**);

PreparedStatement pstmt = con.prepareStatement(sql,ResultSet.**TYPE_SCROLL_INSENSITIVE**, ResultSet.**CONCUR_READ_ONLY**);

Methods for Moving to a New Position :

boolean next (); It returns true when rs contains next record otherwise false.

void beforeFirst (); It is used for making the ResultSet object to point to just before the first record (it is by default)

boolean first (); It is used to point the ResultSet object to first record.

boolean previous (); It returns true when rs contains previous record otherwise false.

void afterLast (); It is used for making the ResultSet object to point to just after the last record.

boolean last (); It is used to point the ResultSet object to last record.

boolean absolute (int n); you can set the cursor to a particular row number.

boolean relative (int n); if n is positive, the cursor moves forward, if n is negative, it moves backward, if n = 0, the call has no effect.

Methods for Checking the Current Position

boolean isFirst (); It returns true when rs is pointing to first record otherwise false.

boolean isLast (); It returns true when rs is pointing to last record otherwise false.

boolean isAfterLast (); It returns true when rs is pointing after last record otherwise false.

boolean isBeforeFirst (); It returns true when rs pointing to before first record otherwise false.

int getRow() : Returns the row number of the current row, or returns 0 if there is no valid current row.

Scrollable ResultSet Demo:

Step 1: Create DBConnect class

```
package com.texas.crud;

import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnect {

	public static Connection getDBConnection() {
		Connection con = null;

		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			String url = "jdbc:mysql://localhost:3306/texas";
			String user = "Admin";
			String password = "Admin";
			con = DriverManager.getConnection(url, user, password);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return con;
	}
}
```

Scrollable ResultSet Demo:

Step 2:

```
package com.texas.crud;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ScrollableresultSetDemo {

		public static void main(String[] args) {
			Connection con = DBConnect.getDBConnection();
			try {
				Statement stm = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
				String sql = "select * from student;";
				ResultSet rs = stm.executeQuery(sql);

				rs.first();
				System.out.println("Row number : "+rs.getRow()+" Name : "+rs.getString(2));

				rs.last();
				System.out.println("Row number : "+rs.getRow()+" Name : "+rs.getString(2));

				rs.absolute(3);
				System.out.println("Row number : "+rs.getRow()+" Name : "+rs.getString(2));


			} catch (SQLException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

		}
}
```

Updatable ResultSet in JDBC

If you want to be able to edit resultSet data and have the changes automatically reflected in the database, you need to create an updatable resultSet.

Syntax:

Statement stmt = conn.createStatement(ResultSet.**TYPE_SCROLL_SENSITIVE**, ResultSet.**CONCUR_UPDATABLE**);

PreparedStatement pstmt = conn.prepareStatement(sql, ResultSet.**TYPE_SCROLL_SENSITIVE**, ResultSet.**CONCUR_UPDATABLE**);

Perform Delete :

The result set **deleteRow**() method will delete the current row.

e.g.
rs.absolute(5);
rs.deleteRow();

Perform Insert :

As with UPDATE operations, result set INSERT operations require separate steps to first write the data to the insert-row and then copy it to the database.

Step 1:

Move to the insert-row by calling the result set **moveToInsertRow** method.

e.g.

rs.**moveToInsertRow**();

Step 2:

As with UPDATE operations, use the appropriate update*XXX* methods to write data to the columns.

e.g.

rs.**updateString**(1, "mystring");

rs.**updateFloat**(2, 10000.0f);

Step 3:

Copy the changes to the database by calling the result set insertRow method.

e.g.

rs.insertRow();

Perform UPDATE :

Performing a result set UPDATE operation requires two separate steps to first update the data in the result set and then copy the changes to the database.

Step 1:

Call the appropriate update*XXX* methods to update the data in the columns you want to change.

E.g.

```
rs.absolute(5);
rs.updateString(1, "mystring");
 rs.updateFloat(2, 10000.0f);
```

Step 2:

Call the **updateRow** method to copy the changes to the database or the **cancelRowUpdates** method to cancel the changes.

E.g.

```
rs.updateRow();
```

```
package com.texas.crud;

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

public class UpdatableResultSetDemo {

	public static void main(String[] args) {
		Connection con = DBConnect.getDBConnection();
		try {
			Statement stm =  con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
			String sql = "select * from student;";
			ResultSet rs =  stm.executeQuery(sql);

			//delete
			rs.absolute(3);
			rs.deleteRow();
			System.out.println("deleted Successfully!");

			//insert
			rs.moveToInsertRow();
			rs.updateString(2,"Lokesh");
			rs.updateString(3, "Pd");
			rs.updateString(4, "Shrestha");
			rs.updateString(5, "2021-09-01");
			rs.insertRow();
			System.out.println("Data inserted Successfully!");

			//update
			rs.absolute(6);
			rs.updateString(2, "Prakash");
			rs.updateString("MiddleName", "Bd");
			rs.updateRow();
			System.out.println("Data Updated Successfully!");

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

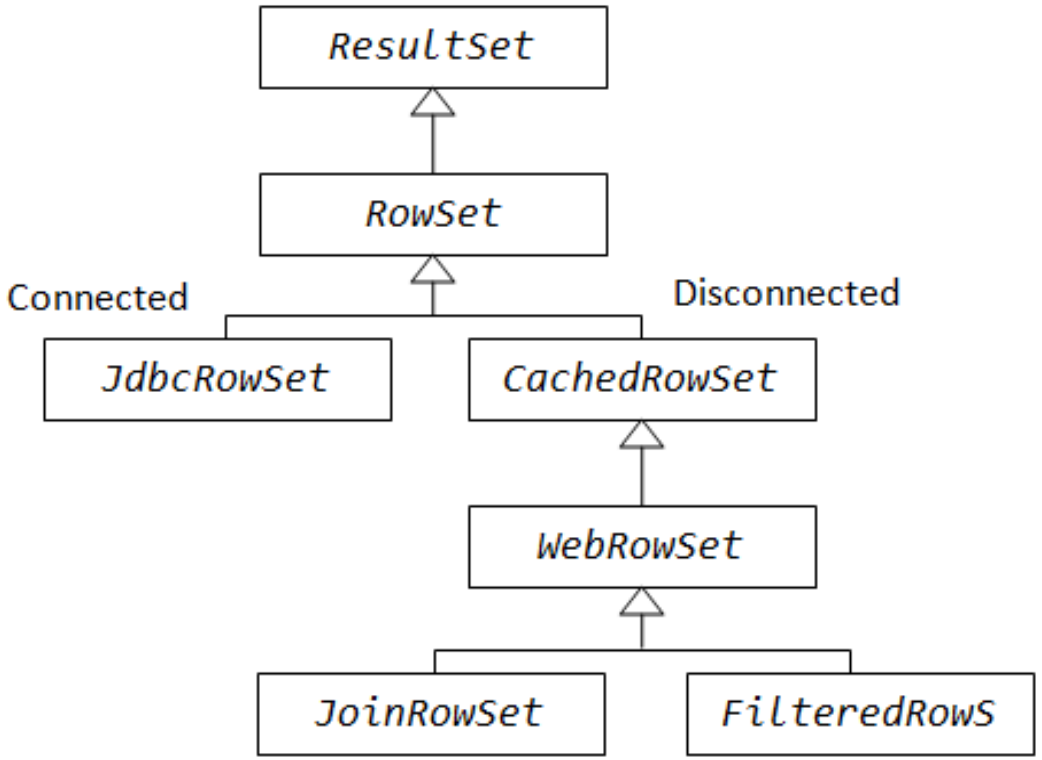
RowSet

- A JDBC RowSet provides a way to store the data in tabular form. It makes the data more flexible and easier than a ResultSet.

- A RowSet is a wrapper around a **ResultSet** Object. It can be connected, disconnected from the database and can be serialized. It maintains a JavaBean component by setting the properties. You can pass a RowSet object over the network. By default, RowSet object is scrollable and updatable and it is used to make a ResultSet object scrollable and updatable.

RowSets are classified into five categories based on how they are implemented which are listed namely as below:

1. JdbcRowSet
2. CachedRowSet
3. WebRowSet
4. FilteredRowSet
5. JoinRowSet



You Can get a RowSet using the

- **RowSetProvider.newFactory().createJdbcRowSet()** method.

In order to connect RowSet with the database, the RowSet interface provides methods for configuring Java bean properties which are depicted below:

- void setURL(String url);
- void setUserName(String user_name);
- void setPassword(String password);
- void setCommand(String query);

RowSet Demo:

```
package com.texas.crud;

import java.sql.*;

import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetDemo {

	public static void main(String[] args) {

		String url = "jdbc:mysql://localhost:3306/texas";
		String user = "Admin";
		String password = "Admin";
		String sql = "select * from student;";

		try {
			JdbcRowSet jdbcrw= RowSetProvider.newFactory().createJdbcRowSet();
			jdbcrw.setUrl(url);
			jdbcrw.setUsername(user);
			jdbcrw.setPassword(password);
			jdbcrw.setCommand(sql);
			jdbcrw.execute();


			while(jdbcrw.next()) {
				String ID = jdbcrw.getString(1);
				String Name = jdbcrw.getString(2);
				System.out.println("Id : "+ID + " Name : "+Name);
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}


	}
}
```

ResultSet

An SQL result set is a set of rows from a database, as well as metadata about the query such as the column names, and the types and sizes of each column. Depending on the database system, the number of rows in the result set may or may not be known.

Characteristics:

- It maintains a connection to a database and because of that, it can't be serialized.
- it can not pass the Result set object from one class to another class across the network.
- ResultSet object maintains a cursor pointing to its current row of data. Initially, the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.
- ResultSet alone cannot be used as a JavaBeans component.

Difference :

RowSet	ResultSet
RowSet is present in the javax.sql package	ResultSet is present in the java.sql package
A Row Set can be connected, disconnected from the database.	A ResultSet always maintains the connection with the database.
RowSet is scrollable providing more flexibility	ResultSet by default is always forward only
A Row Set object can be serialized.	It cannot be serialized.
You can pass a Row Set object over the network.	ResultSet object cannot be passed other over the network.
Result Set Object is a JavaBean object. RowSet using the RowSetProvider.newFactory().createJdb cRowSet() method.	Result Set object is not a JavaBean object result set using the executeQuery() method