Database Programming in Java

Jigisha Patel

Introduction to JDBC

- JDBC stands for Java DataBase Connectivity.
- JDBC is an **API** for the Java programming language that defines how a client may access a database.
- Sun Microsystems has included JDBC API as a part of J2SDK to develop Java Applications that can communicate with databases.
- It was first introduced in J2SE.

JDBC Architecture

- Java application cannot directly communicate with a database to submit data and retrieve the results of queries.
- This is because a database can interpret only SQL statements and not Java language statements.
- For this reason, a mechanism to translate Java statements into SQL is required.

- The JDBC architecture provides the mechanism to translate Java statements into SQL statements.
- It can be classified into two layers:
  1. JDBC Application Layer
  2. JDBC Driver Layer

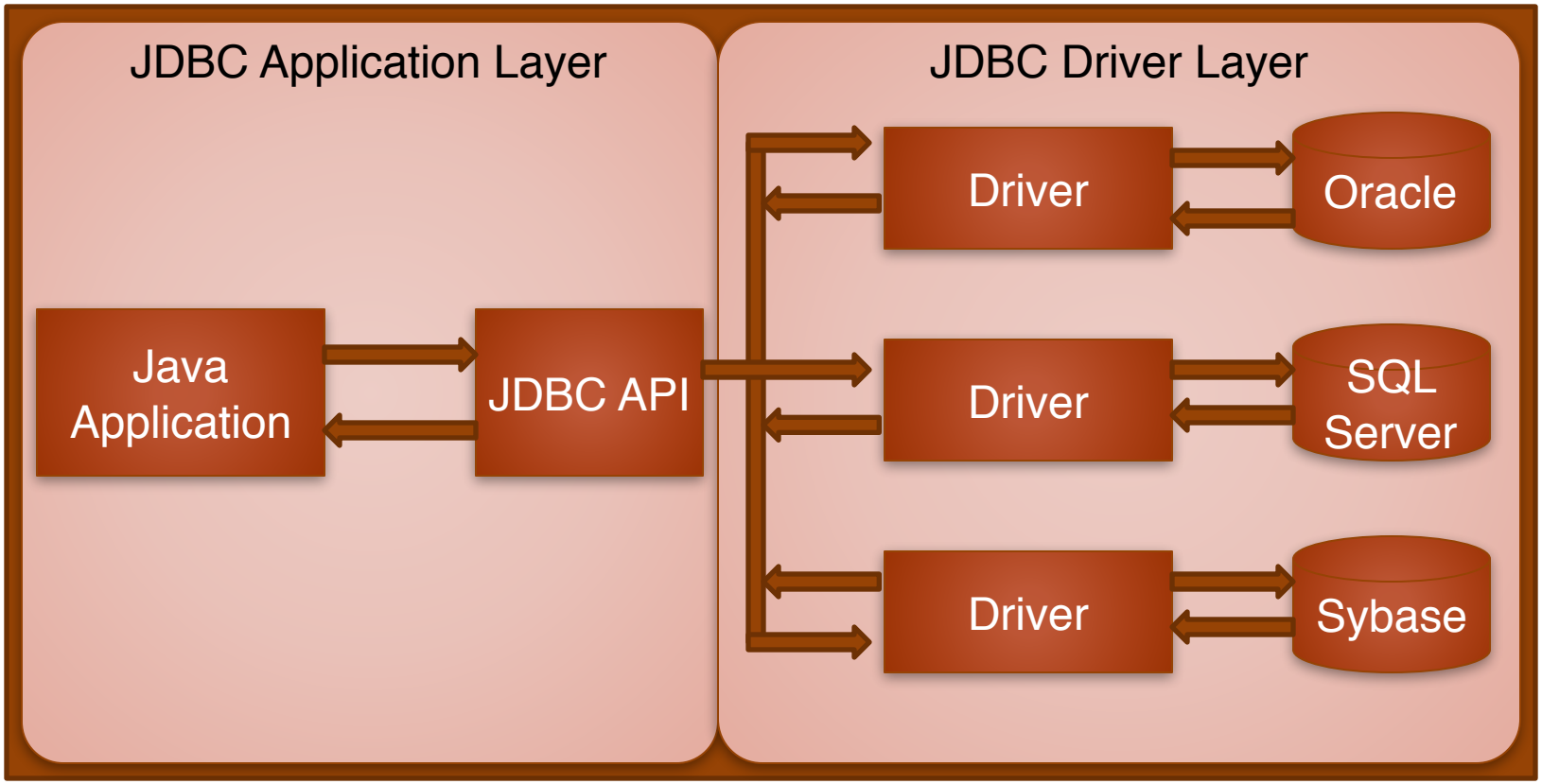


Fig: JDBC Architecture

- JDBC application layer:
  - It signifies a Java application that uses the JDBC API to interact with the JDBC drivers.
  - A JDBC driver is software that a Java application uses to access a database.
  - The JDBC driver manager of JDBC API connects the Java application to the driver.

- JDBC driver layer:
  - It acts as an interface between a Java application and a database.
  - This layer contains a driver, such as SQL Server driver or an oracle driver, which enables connectivity to a database.
  - A driver sends the request of a Java application to the database.
  - After processing the request, the database sends the response back to the driver.
  - The driver translates and sends the response to the JDBC API.
  - The JDBC API forwards it to the Java Application.

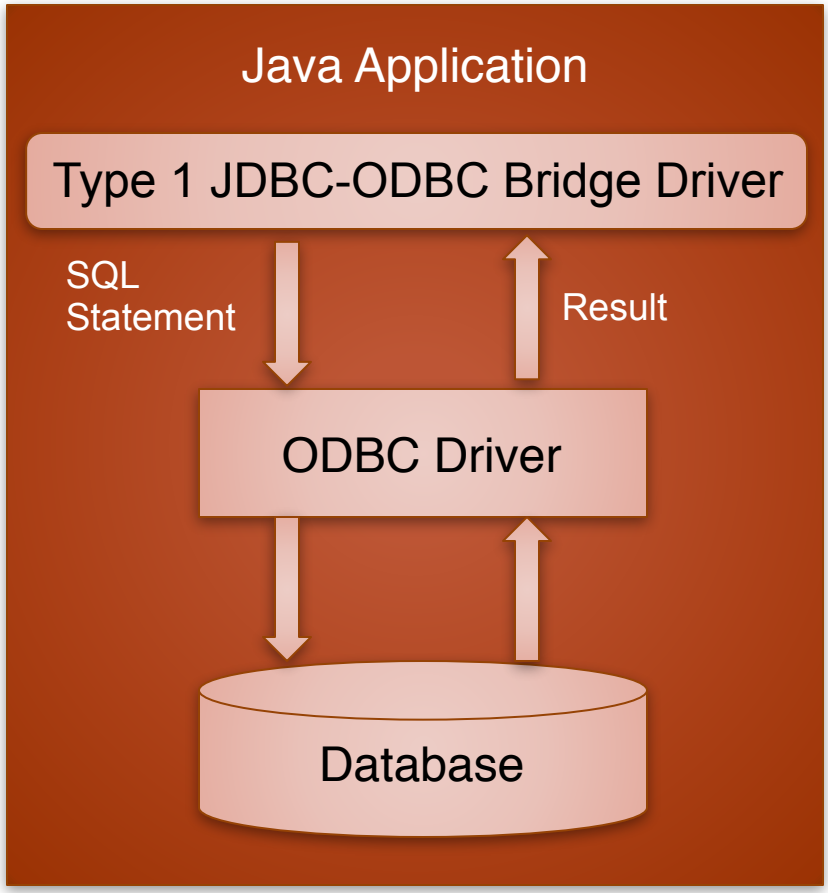
JDBC drivers

- JDBC drivers are required to convert queries into a form that a particular database can interpret.
- The JDBC driver also retrieves the result of SQL statements and converts the result into equivalent JDBC API class object that the Java application uses.
- As the JDBC driver only takes care of interaction with the database, any change made to the database does not affect the application.
- There are 4 types of drivers available:
  1. JDBC-ODBC Bridge driver
  2. Native-API Partly-Java driver
  3. JDBC-Net Pure-Java driver
  4. Native Protocol Pure-Java driver

The JDBC-ODBC Bridge Driver

- It is called Type 1 driver.
- It converts the JDBC calls to Open Database Connectivity (ODBC) calls.
- ODBC is an open standard API to communicate with databases.
- The JDBC-ODBC bridge driver enables a Java application to use any database that supports ODBC driver.
- A Java application cannot interact directly with the ODBC driver.
- For this reason, the application uses the JDBC-ODBC Bridge driver that works as an interface between the application and the ODBC driver.

- To use this driver, the ODBC driver requires to be installed on the client computer.
- The Type 1 driver is usually used in stand-alone applications.



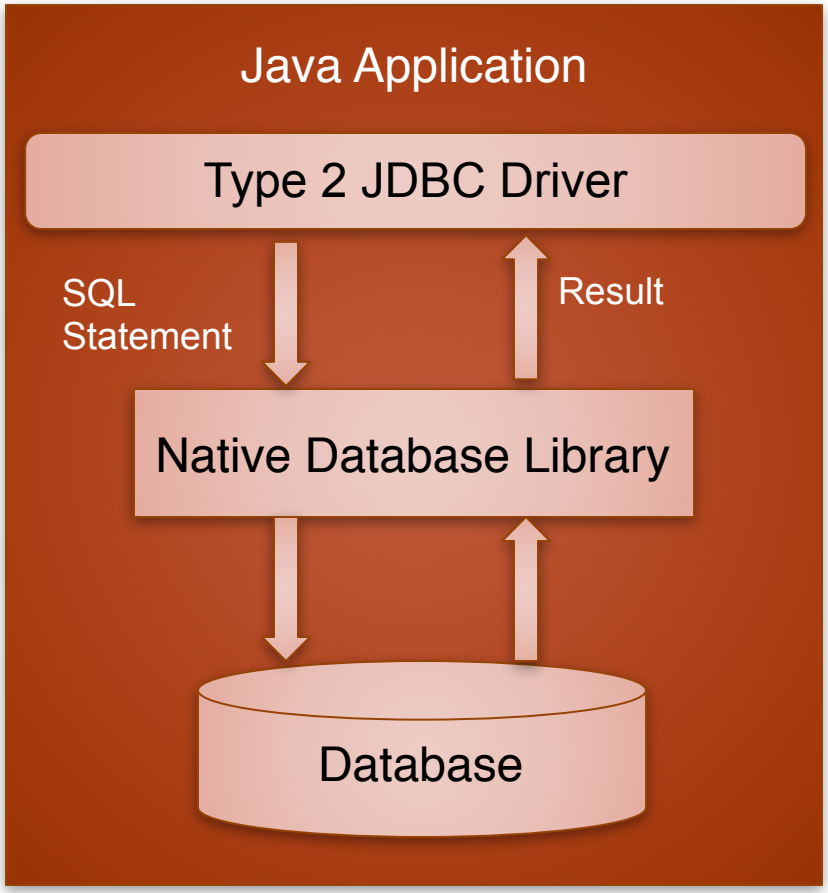
- Advantages:
  - Almost any database for which ODBC driver is installed, can be accessed.
  - A type 1 driver is easy to install.

- Disadvantages:
  - Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
  - The client system requires the ODBC Installation to use the driver.
  - The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system.
  - The performance is degraded since the JDBC call goes through the bridge to the ODBC driver, then to the native database connectivity interface. The result comes back through the reverse process.
  - Type 1 drivers may not be suitable for large-scale applications as well as for applets.

The Native-API Partly-Java Driver

- It is called Type 2 driver.
- It uses the local native libraries provided by the database vendors to access databases.
- The JDBC driver maps the JDBC calls to the native method calls, which are passed to the local native Call Level Interface (CLI).
- This interface consists of functions written in C to access databases.
- To use the Type 2 driver, CLI needs to be loaded on the client computer.
- As opposed to the Type 1 driver, Type 2 driver does not have an ODBC intermediate layer.
- As a result, this driver has a better performance than the JDBC-ODBC bridge driver.
- This driver is usually used for network-based application.

- The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls.
- This driver is specific to a particular database.



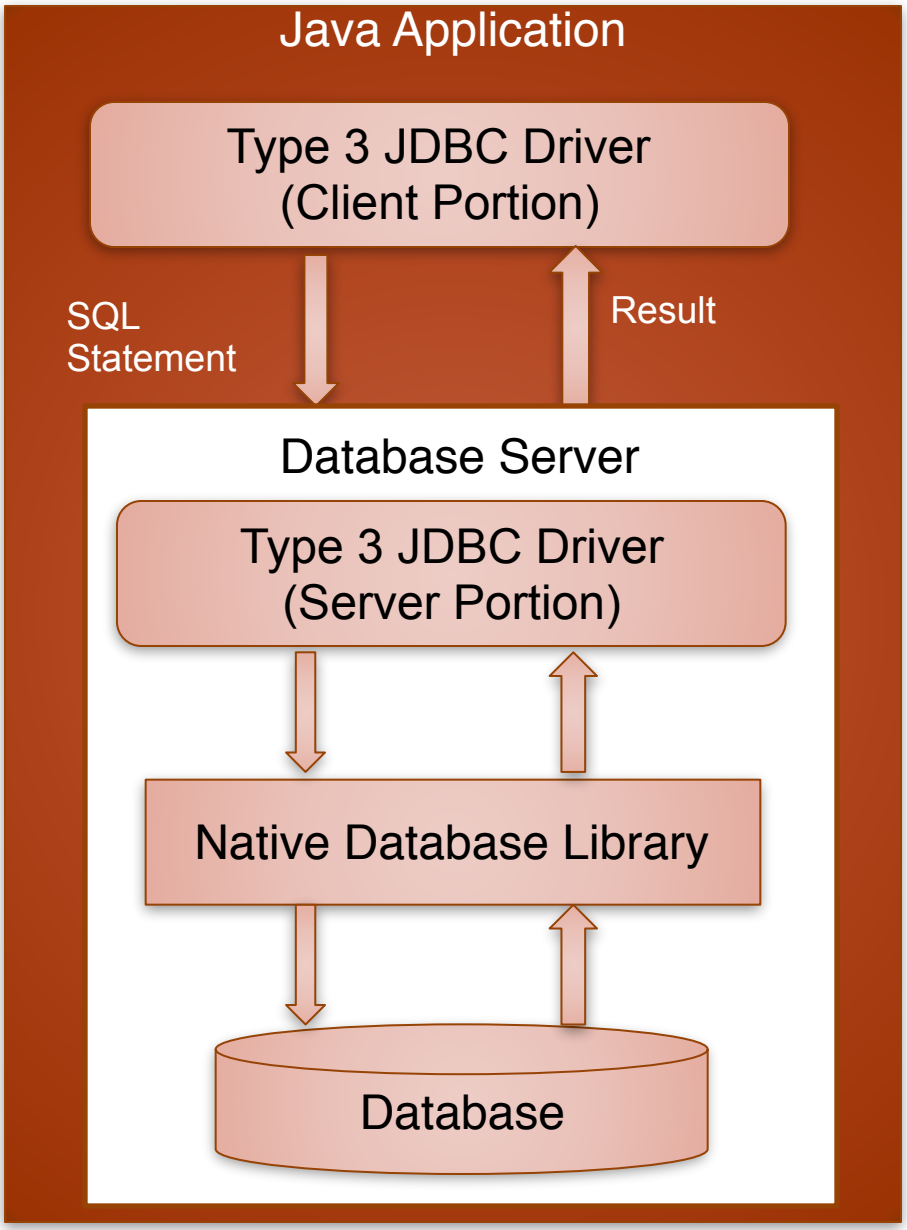
- Advantages:
  - Type 2 JDBC drivers offer better performance than the type
    1.
  - Provides fastest performance than all 3 drivers as it calls native APIs(MySQL, Oracle,...etc).

- Disadvantages:
  - Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
  - It is not written in Java Language which makes it platform dependent.
  - If we change the database we have to change the native API as it is specific to a database.
  - Not all databases have a client side library.

The JDBC-Net Pure-Java Driver

- It is called Type 3 driver.
- This driver can be used over the web while connecting applets with databases.
- It consist of client and server portions.
- The client portion contains pure Java functions and the server portion contains Java and native methods.
- The Java application sends JDBC calls to the client portion, which in turn, translates JDBC calls into database calls.
- The database calls are sent to the server portion that forwards the request to the database.
- When type 3 driver is used, CLI native libraries are loaded on the server.

- Makes use of a middle tier between the calling program and the database.
- The protocol conversion logic resides not at the client, but in the middle-tier.
- Type 3 driver is written entirely in Java.
- The type 3 driver is platform-independent as the platform-related differences are taken care by the middleware.


Java Application
Type 3 JDBC Driver
(Client Portion)
SQL
Statement
Result
Database Server
Type 3 JDBC Driver
(Server Portion)
Native Database Library
Database

- Advantages:
  - This driver is server-based, so there is no need for any vendor database library to be present on client machines.
  - This driver is fully written in Java and hence portable.
  - It is suitable for the web.
  - This driver is very flexible allows access to multiple databases using one driver.

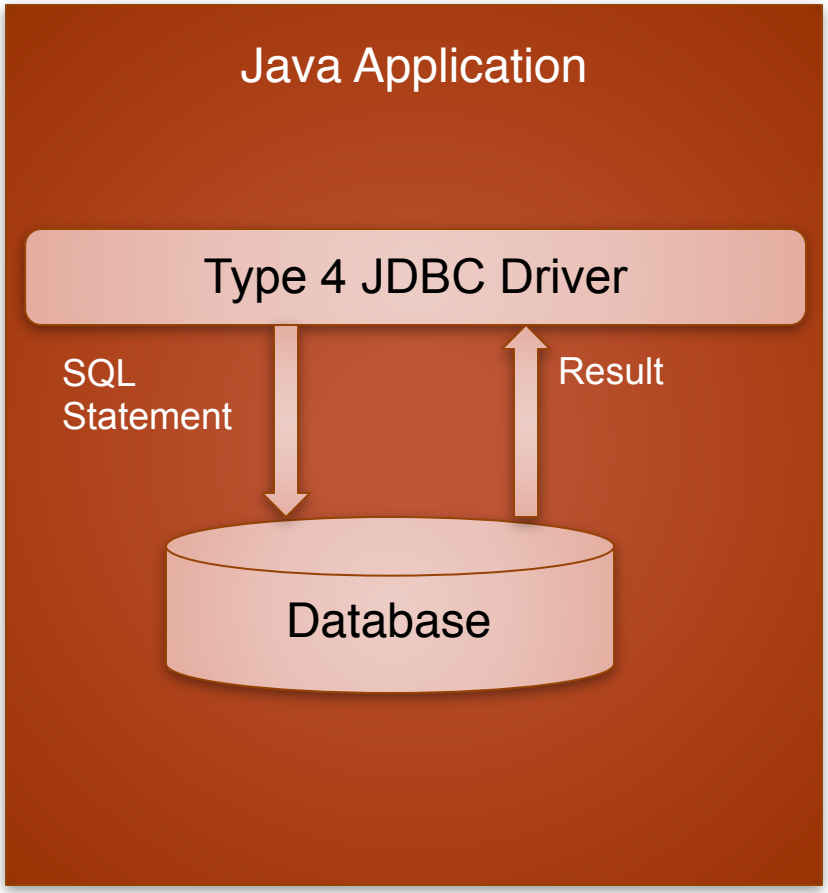
- Disadvantages:
  - Traversing the recordset may take longer, since the data comes through the backend server.
  - Requires database-specific coding to be done in the middle tier.

The Native-Protocol Pure-Java Driver

- It is called Type 4 driver.
- It is a Java driver that interact with the database directly using a vendor-specific network protocol.

- As opposed to the other JDBC drivers, it is not required to install any vendor-specific libraries to use the Type 4 driver.

- DataDirect Technologies provide Type 4 driver for various databases such as MS SQL server, AS/400, and DB2.
- This driver is usually used for enterprise applications.


Java Application
Type 4 JDBC Driver
SQL
Statement
Result
Database

- Advantages:
  - Written completely in Java, type 4 drivers are thus platform independent.
  - This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls.
  - You don't need to install special software on the client or server.

- Disadvantages:
  - With type 4 drivers, the user needs a different driver for each database.
  - It is database dependent.

Using JDBC API

- It is required to use database drivers and the JDBC API while developing a Java application to retrieve or store data in a database.

- The JDBC API classes and interfaces are available in the java.sql and javax.sql packages.

- The classes and interfaces perform a number of tasks, such as establish and close a connection with a database, send a request to a database, retrieve data from a database, and update data in a database.

- The commonly used classes and interfaces in the JDBC API are as follows.

1. DriverManager class:
   - Loads the driver for a database.

2. Driver Interface:
   - Represents a database driver.
   - All JDBC driver classes must implement the Driver interface

3. Connection interface:
   - Enables you to establish a connection between a Java application and a database.

4. **Statement** interface:
   - Enables you to execute SQL statements.

5. **ResultSet** interface:
   - Represents the information retrieved from a database.

6. **SQLException** class:
   - Provides information about the exceptions that occur while interacting with databases.

- To query a database and display the result using Java application, the following four steps are required to perform:
  1. Load a driver
  2. Connect to a database
  3. Create and execute JDBC statements
  4. Handle SQL exceptions

Load a Driver

- The first step to develop a JDBC application is to load and register the required driver using the driver manager.

- There are two ways to load and register a driver:
  1. Programmatically
     - Using the forName() method
     - Using the registerDriver() method
  2. Manually
     - By setting system property

- Using the forName() method
  - This method is available in the java.lang.Class class.
  - The forName() method loads the JDBC driver and registers the driver with the driver manager.
  - Syntax:

    Class.forName("<driver_name>");
  - E.g.

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

- Using the registerDriver() method
  - An instance of the Driver class is to be created to load a JDBC driver.
  - Syntax

    Driver d = new <driver name>;
  - E.g. Driver d = new sun.jdbc.odbc.JdbcOdbcDriver();
  - After creating the Driver object, registerDriver() method has to be called to register it with DriverManager.
  - E.g. DriverManager.registerDriver(d);

- Setting System Property
  - Drivers can also be loaded by setting system property for JDBC drivers.
  - To load the JDBC driver, the driver name has to added to the jdbc.drivers system property.
  - The command line option –D is used to set the system property on the command line.
  - E.g.

```
java –Djdbc.drivers = sun.jdbc.odbc.JdbcOdbcDriver
     Sample Application
```

  - jdbc.drivers is the property name
  - sun.jdbc.odbc.JdbcOdbcDriver is the value of the property.

Connecting to a Database

- An object of the Connection interface has to be created to establish a connection of the Java application with a database.
- Multiple Connection objects can also be created to access and retrieve data from multiple databases.
- The DriverManager class provides the getConnection() method to create a Connection object.
- The getConnection() method is an overloaded method that has the following three formats:

1. Connection getConnection (String <url>)

- Accepts the JDBC URL of the database as a parameter, which need to be accessed.

- The following is the code snippet to connect to a database using the getConnection() method with a single parameter:

```
String url = "jdbc:odbc:MyDataSource";
Connection con = DriverManager.getConnection(url);
```

- The syntax for a JDBC URL that is passed as a parameter to the getConnection() method is:

<protocol>:<subprotocol>:<subname>

- The JDBC URL has the following three components:

1. Protocol name:

- Indicates the name of the protocol that is used to access a database.
- In JDBC, the name of the access protocol is always jdbc.

2. Sub-protocol name:

- Indicates the mechanism to retrieve data from a database.
- For Type 1 driver the name of the sub-problem is odbc.

3. Subname:

- Indicates the Data Source Name(DSN) that contains database information, such as the name of a database, location of the database server, user name and password to access a database server.

2. Connection getConnection (String <url>, String <username>, String <password>)

- Accepts the JDBC url of a database.
- It also accepts the user name and password of the authorized database user.

- E.g.

```
String url = "jdbc:odbc:MyDataSource";
Connection con =
  DriverManager.getConnection(url,"NewUser","NewPa
  ssword");
```

3. Connection getConnection (String <url>,Properties <properties>)

- Accepts the JDBC URL of a database and an object of java.util.Properties as parameters.
- Information such as username and password can be specified in the Properties object by using the setProperty() method.

```
String url = “jdbc:odbc:MyDataSource”;
Properties p = new Properties();
p.setProperty(“user”,”NewUser”);
p.setProperty(“password”,”NewPassword”);
Connetion con = DriverManager.getConnection(url,p);
```

- *p* is reference to an object of the **Properties** Class.
- The username and password properties are set using **setProperty()** method.

- After creating connection, the JDBC statement that are to be executed has to be written.

Creating and Executing JDBC Statements

- A **Statement** object is required to be created to send request to and retrieve results from a database.
- The **Connection** object provides the createStatement() method to create **Statement** object.
- E.g.

```
Connection con = DriverManager.getConnection
("jdbc:odbc:MyDataSource","NewUser","NewPasswo
rd");

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

- The static SQL statement can be used to send request to a database.
- The SQL statement that do not contain runtime parameter are called Static SQL statement.
- SQL statement can be send to database using the **Statement** object.

- The **Statement** Interface contains following methods to send static SQL statement to a database.

1. ResultSet executeQuery(String str)

- Executes an SQL statement and returns a single object of type, **ResultSet**.
- This object provide methods to access the data from resultset.
- The **executeQuery()** method should be used when the data is required to be retrived from a database table using the **SELECT** statement.
- The syntax to use **executeQuery()** method is :

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(<SQL Statement>);
```

2. int executeUpdate(String str)

- Execute the SQL statements and returns the number of data rows that are affected after processing of the SQL statement.
- When the data in the database is needed to be modified using Data Manipulation Language (DML) statement, INSERT, DELETE & UPDATE, executeUpdate() method() can be us.

- The syntax is as follows:

```
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(<SQL
    Statement>);
```

3. boolean execute(String str)

- Executes an SQL statement and returns boolean value.
- When the type of SQL statement passed as a parameter is not known or when the statement being executed returns a result set or an update count, this method can be used.
- The **execute**() method returns **true** if the result of the SQL statement is an object of **ResultSet** and **false** if it is an update count.
- The syntax is:

```
Statement stmt = con. createStatement();
stmt.execute(<SQL statement>);
```

- Various database operations that can be performed using a Java application are:
  1. Querying a table
  2. Inserting rows in a table
  3. Updating rows in a table
  4. Deleting rows from a table
  5. Creating a table
  6. Altering a table
  7. Dropping a table

Querying a table

- The data can be retrieved from a table using the SELECT statement.
- The SELECT statement is executed using the executeQuery() method and returns the output in the form of a ResultSet object.

```
String str = "SELECT * FROM students";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(str);
```

Inserting rows in a table

- Rows can be added in an existing table using the INSERT statement.
- The executeUpdate() method enables you to add rows in a table.

```
String str = "INSERT INTO students (st_id, st_name,
    st_course) VALUES (‘M09001‘, ‘Abc', ‘MCA’)";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

Updating rows in a table

- Existing records in the table can be modified using UPDATE statement.

```
String str = "UPDATE students SET st_name='Xyz'
    WHERE st_id='M09001' ";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

Deleting rows in a table

- Existing records in the table can be deleted using DELETE statement.

```
String str = "DELETE FROM students WHERE
    st_id='M09001' ";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

Creating a table

- The CREATE TABLE statement is used to create and define the structure of a table in a database.
- E.g.

```
String str="CREATE TABLE Marks (st_id VARCHAR(6),st_name VARCHAR(25), total_marks INTEGER)";
Statement stmt=con.createStatement();
stmt.execute(str);
```

Altering and Dropping a table

- DDL provides the ALTER statement to modify the definition of database object.

```
String str="ALTER TABLE Marks  ADD percentage FLOAT";
Statement stmt=con.createStatement();
stmt.execute(str);
```

- DDL provides the DROP TABLE statement to drop a table from a database.

```
String str="DROP TABLE Marks";
Statement stmt=con.createStatement();
stmt.execute(str);
```

Handling SQL Exception

- The java.sql package provides the SQLException class, which is derived from the java.lang.Exception class.
- You can catch the SQLException in a Java application using the try and catch exception handling block.
- The SQLException class contains various methods that provide error information, these methods are:

1. int getErrorCode(): Returns the error code associated with the error occurred.
2. String getSQLState(): Returns X/Open error code.
3. SQLException getNextException(): Returns the next exception in the chain of exceptions.

Accessing Result Sets

- When a query to retrieve data from a table is executed using a Java application, the output of the query is stored in a ResultSet object in a tabular format.
- A ResultSet object maintains a cursor that enables you to move through the rows stored in a ResultSet object.
- By default, the ResultSet object maintains a cursor that moves in the forward direction only.
- As a result, it moves from the first row to the last row in the ResultSet.
- The default ResultSet object cannot be updated.
- The cursor in the ResultSet object initially points before the first row.

- **Types of Result Sets**:
  - The various types of ResultSet objects to store the output returned by a database are:

1. **Read only**: Allows you to only read the rows in a ResultSet object.
2. **Forward only**: Moves the result set cursor from first row to last row in forward direction only.
3. **Scrollable**: Moves the result set cursor forward or backward through the result set.
4. **Updatable**: Allows you to update the result set rows retrieved from a database table.

- The following table lists various fields of ResultSet interface that you can use to specify the type of a ResultSet object:

ResultSet Fields	Description
TYPE_SCROLL_SENTITIVE	Specifies that the cursor of the ResultSet object is scrollable and it reflects the changes in the data made by other users.
TYPE_SCROLL_INSENSITIVE	Specifies that the cursor of the ResultSet object is scrollable and it does not reflect changes in the data made by other users.
TYPE_FORWARD_ONLY	Specifies that the cursor of the ResultSet object moves in forward direction only from the first row to the last row.

- The following table lists various fields of the ResultSet interface that you can use to specify different concurrency modes of result sets:

ResultSet Fields	Description
CONCUR_READ_ONLY	Specifies the concurrency mode that does not allow you to update the ResultSet object.
CONCUR_UPDATABLE	Specifies the concurrency mode that allows you to update the ResultSet object.

- The following table lists various fields of the ResultSet interface that you can use to specify different cursor states of result sets:

ResultSet Fields	Description
HOLD_CURSORS_OVER_COMMIT	Specifies that a ResultSet object should not be closed after data is committed to the database.
CLOSE_CURSORS_AT_COMMIT	Specifies that a ResultSet object should be closed after data is committed to the database.

- The createStatement() method has the following three overloaded forms:

1. Statement createStatement():
   - Does not accept any parameter.
   - This method creates a default ResultSet object that only allows forward scrolling.
2. Statement createStatement(int, int) :
   - Accepts two parameters.
   - The first parameter indicates the ResultSet type that determines whether or not, a result set cursor can move backward.
   - The second parameter indicates the concurrency mode for the result set that determines whether the data in result set can be updated.
   - This method creates a ResultSet object with the given type and concurrency.

3. Statement createStatement(int, int, int) :

- In addition to the ResultSet types and the concurrency mode, this method accepts a third parameter.
- This parameter indicates whether or not, the result set is closed after committing data to the database.
- This method creates a ResultSet object with the given type, concurrency, and state.

Methods of ResultSet Interface

- The following tables lists the methods of ResultSet interface:

Method	Description
boolean first()	Shifts the control of a result set cursor to the first row of the result set.
boolean isFirst()	Determines whether the result set cursor points to the first row of the result set.
boolean beforeFirst()	Shifts the control of a result set cursor before the first row of the result set.
boolean isBeforeFirst()	Determines whether the result set cursor points before the first row of the result set.
boolean last()	Shifts the control of a result set cursor to the last row of the result set.
boolean isLast()	Determines whether the result set cursor points to the last row of the result set.
boolean afterLast()	Shifts the control of a result set cursor after the last row of the result set.
boolean isAfterLast()	Determines whether the result set cursor points after the last row of the result set.
boolean previous()	Shifts the control of a result set cursor to the previous row of the result set.
boolean absolute(int i)	Shifts the control of a result set cursor to the row number that you specify as a parameter.

- JDBC allows you to create an updatable result set that enables you to modify the rows in the result set.
- The following table lists some of the methods used with updatable result set:

Method	Description
void updateRow()	Updates a row of the current ResultSet object and the underlying database table.
void insertRow()	Inserts a row in the current ResultSet object and the underlying database table.
void deleteRow()	Deletes a row from the current ResultSet object and the underlying database table.
void updateString()	Updates the specified column with the given string value.
void updateInt()	Updates the specified column with the given int value.