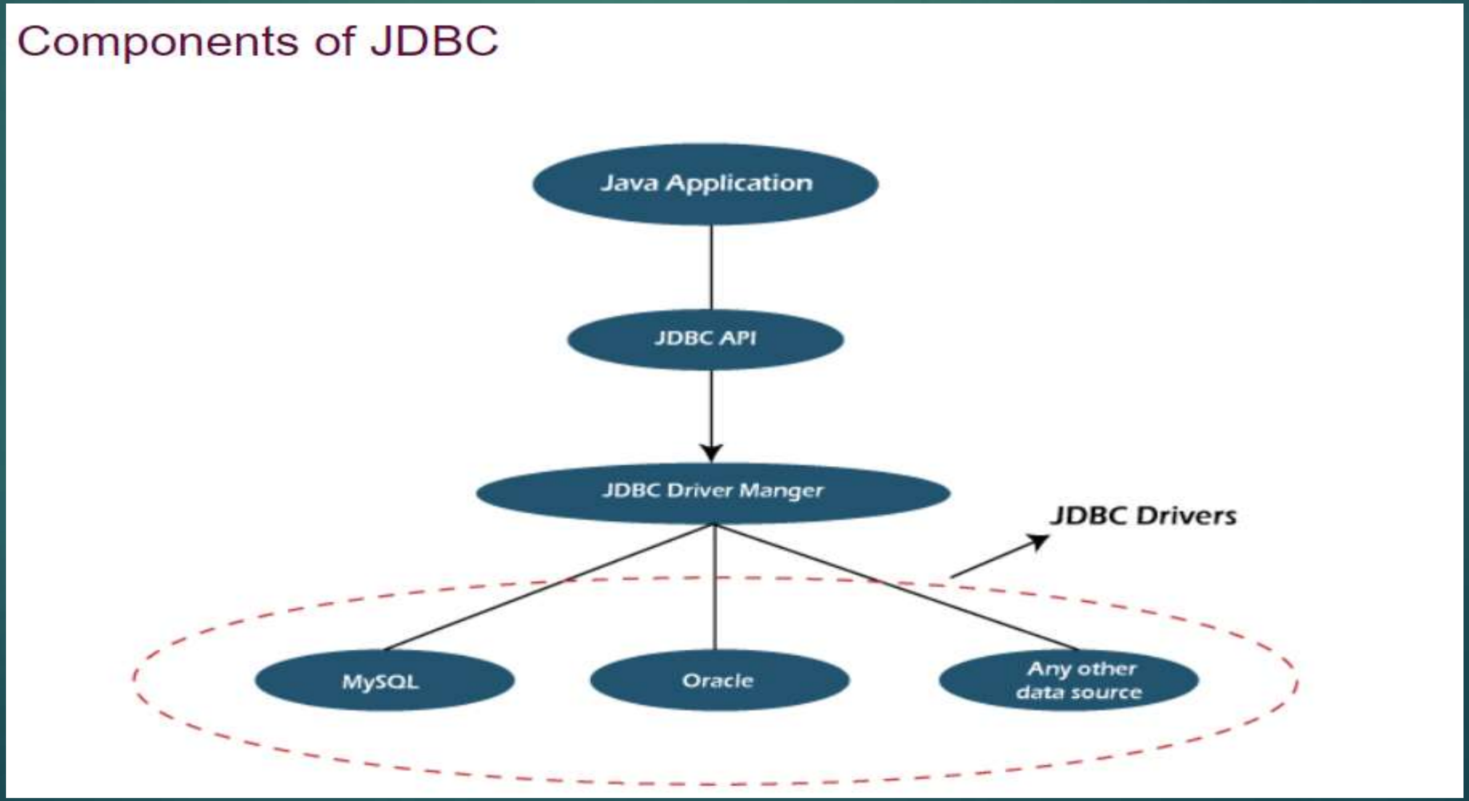
JDBC – Java Database Connectivity

JDBC Design

- JDBC is a Java API to connect and execute the query with the database.
- It is a part of JavaSE (Java Standard Edition).
- 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.
- Design of JDBC defines the components of JDBC, which is used for connecting to the database.



JDBC Design

- **JDBC has four major components that are used for the interaction with the database.**
  - **JDBC API** : JDBC API provides various interfaces and methods to establish easy connection with different databases.
    - javax.sql.*;
    - java.sql.*;
  - **JDBC Test Suite** : JDBC Test suite facilitates the programmer to test the various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers.
  - **JDBC Driver Manger** : DBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database.
  - The JDBC Driver manager is also used to make the database-specific call to the database in order to do the processing of a user request.
  - **JDBC Drivers : it** is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:
    - **Type-1 driver or JDBC-ODBC bridge driver**
    - **Type-2 driver or Native-API driver (partially java driver)**
    - **Type-3 driver or Network Protocol driver (fully java driver)**
    - **Type-4 driver or Thin driver (fully java driver)**

ODBC Vs JDBC

ODBC	JDBC
1. ODBC Stands for Open Database Connectivity.	1. JDBC Stands for Java database connectivity.
2. Introduced by Microsoft in 1992.	2. Introduced by SUN Micro Systems in 1997.
3. We can use ODBC for any language like C, C++, Java etc.	3. We can use JDBC only for Java languages.
4. We can choose ODBC only on Windows platform.	4. We can use JDBC on any platform.
5. Mostly ODBC Driver is developed in native languages like C, and C++.	5. JDBC is developed in Java language.
6. For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform-dependent.	6. For Java applications it is highly recommended to use JDBC because there are no performance & platform dependent problems.
7. ODBC is procedural.	7. JDBC is object-oriented.

Jdbc Drivers – Type 1 Driver

- **Type-1 driver or JDBC-ODBC bridge driver :**
  - The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database.
  - The JDBC-ODBC bridge driver converts JDBC method calls into ODBC function calls.
  - Type-1 driver is also called Universal driver because it can be used to connect to any of the databases
  - This is now discouraged because of the thin driver.
  - **Oracle does not support the JDBC-ODBC Bridge from Java 8.**

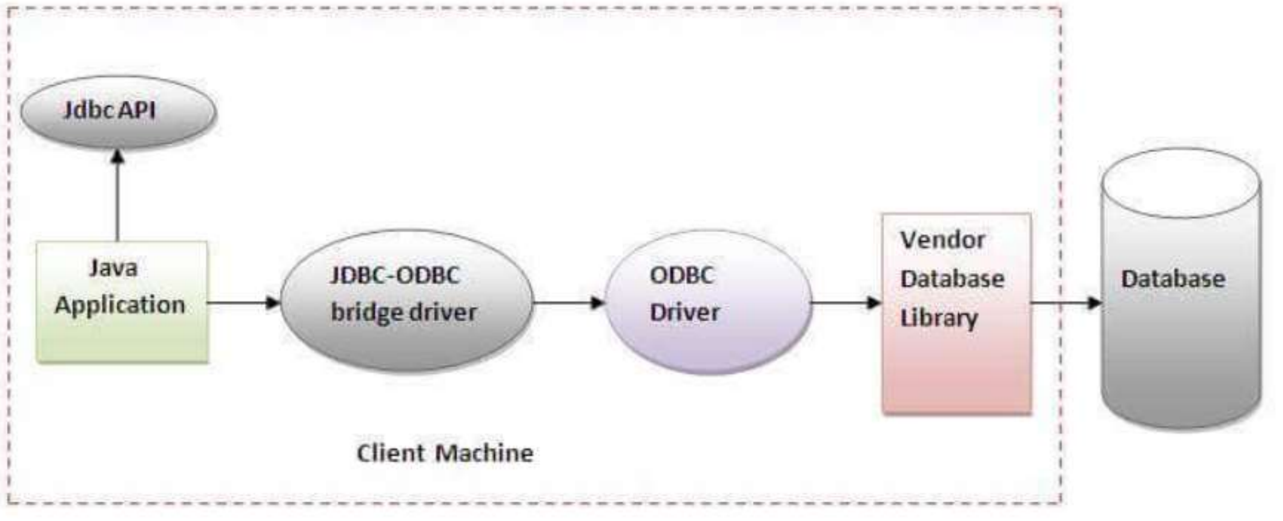


Figure- JDBC-ODBC Bridge Driver

Type-1 driver

- **Advantages**
    - easy to use.
    - can be easily connected to any database
- **Disadvantages:**
    - Performance degraded because JDBC method call is converted into the ODBC function calls.
    - The ODBC driver needs to be installed on the client machine.
    - The Vendor client library needs to be installed on client machine

Jdbc Drivers – Type 2 Driver

- **Type-2 driver or Native-API driver (partially java driver) :**
  - It 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

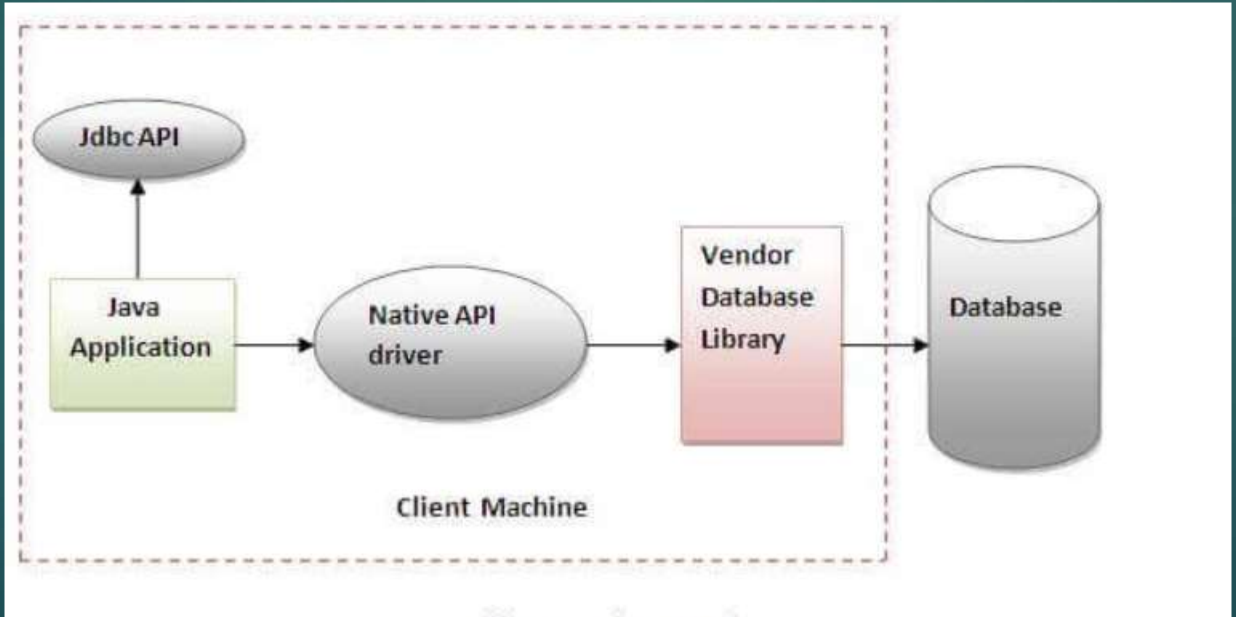


Figure- Native API Driver

Jdbc Drivers – Type 2 Driver

- **Advantage**:
  - performance upgraded than JDBC-ODBC bridge driver.
- **Disadvantage**:
  - The Native driver needs to be installed on the each client machine.
  - The Vendor client library needs to be installed on client machine.

Jdbc Drivers – Type 3 Driver

- **Network Protocol driver (Fully Java) :**
  - 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.

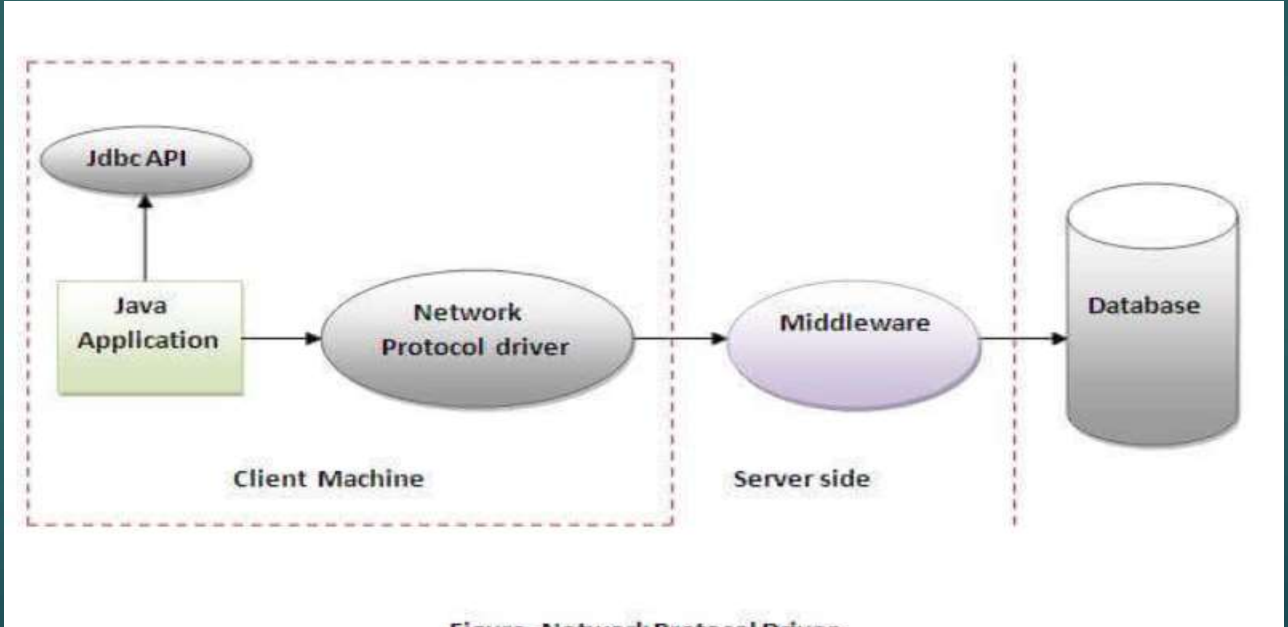


Figure- Network Protocol Driver

Jdbc Drivers – Type 3 Driver

- **Advantage**:
  - No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- **Disadvantages**:
  - Network support is required on client machine.
  - Requires database-specific coding to be done in the middle tier.
  - Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Jdbc Drivers – Type 4 Driver

- **Type-4 driver or Thin driver (fully java 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.

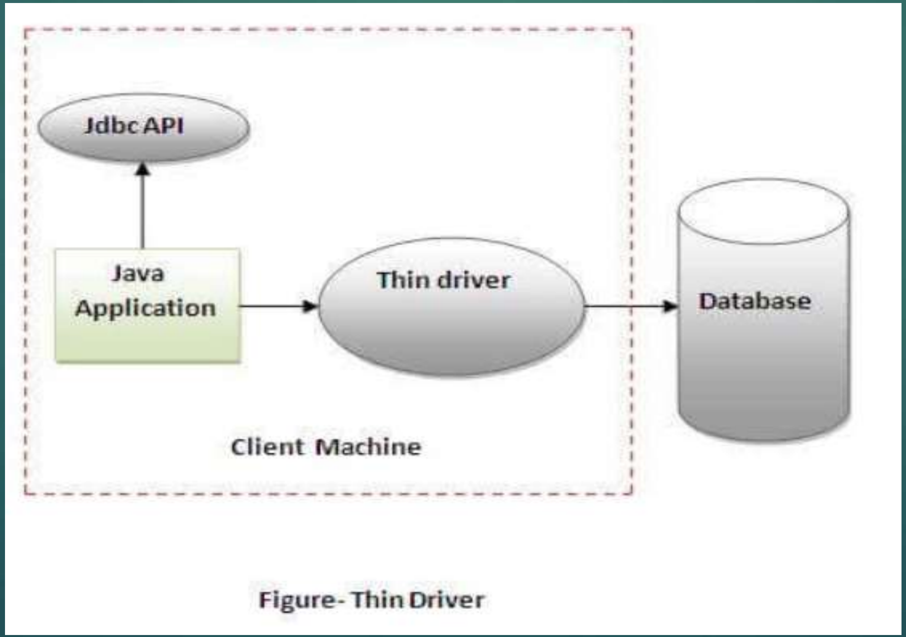


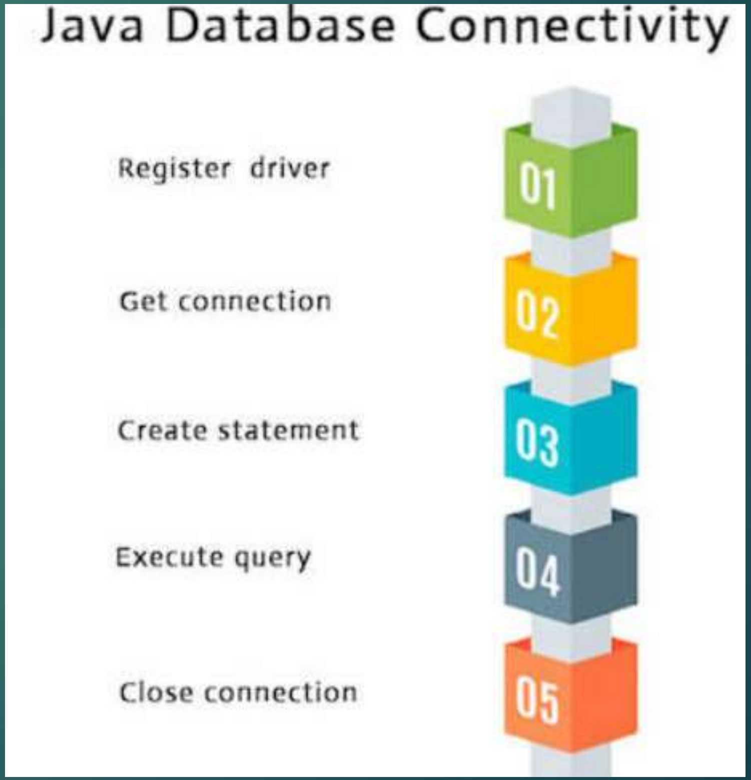
Figure- Thin Driver

Jdbc Drivers – Type 4 Driver

- **Advantage**:
    - Better performance than all other drivers.
    - No software is required at client side or server side.
- **Disadvantage**:
    - Drivers depend on the Database.

Steps To Connect To The Database In Java

- **Java Database Connectivity with 5 Steps** :
  - There are 5 steps to connect any java application with the database using JDBC.
  - These steps are as follows:
    - Register for the Driver class
    - Create connection
    - Create statement
    - Execute queries
    - Close connection



Steps To Connect To The Database In Java

- **Step 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.
- **Method forName() syntax :**
  - **public static void** forName(String className)**throws** ClassNotFoundException
- **Example to register the Driver class**
  - For Oracel → Class.forName("oracle.jdbc.driver.OracleDriver");
  - For MySql → Class.forName("com.mysql.jdbc.Driver");
  -

Note :

Since JDBC 4.0, explicitly registering the driver is optional. We just need to put the vendor's Jar in the classpath, and then the JDBC driver manager can detect and load the driver automatically.

Steps To Connect To The Database In Java

- **Step 2: Create the connection object:**
  - The **getConnection()** method of DriverManager class is used to establish connection with the database.
- **Method getConnection() syntax :**
  - **public static** Connection getConnection(String url)**throws** SQLException
  - **public static** Connection getConnection(String url,String name,String password)

    throws SQLException

- **Example to establish connection with**
- **Oracle**

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","passw
ord");

- **MySql**

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","mydbuser",
"mydbuser");

Steps To Connect To The Database In Java

- **Step 3: Create the Statement object**
  - The createStatement() method of the Connection interface is used to create a statement.
  - The object of the statement is responsible for executing queries with the database.
- **Method** createStatement**() syntax :**
  - **public** Statement createStatement()**throws** SQLException
- **Example to create the statement object**

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

Steps To Connect To The Database In Java

- **Step 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.
- **Method** executeQuery **syntax :**
  - **public** ResultSet executeQuery(String sql)**throws** SQLException
  - **Example to create the statement object**

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Steps To Connect To The Database In Java

- **Step 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.
  - **Method** close() **syntax :**
    - **public void** close()**throws** SQLException
  - **Example to close connection**
    - **con.close();**

Note

Note: Since Java 7, JDBC has the ability to use try-with-resources statements to automatically close resources of type Connection, ResultSet, and Statement.

Example To Connect To The Oracle Database

To connect java application with the oracle database, we need to follow 5 steps defined in previous slides. And we need to gather few information like below before we start :

1. **Driver class :** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**
2. **Connection URL :** "jdbc:oracle:thin:@localhost:1521:xe"
3. **Username :**
4. **Password :**

Statement Types

There are 3 types of Satement in JDBC they are -

1. **Statement**
2. **PreparedStatement**
3. **Callable**

Statement :

- The Statement interface represents the static SQL statement.
- It helps you to create general-purpose SQL statements using Java.

For Example

```
Statement stmt = conn.createStatement( );
ResultSet rs = stmt.executeQuery("select * from Student");
```

Executing the Statement object

- Once you have created the statement object you can execute it using one of the execute methods namely,
  - **boolean execute(String SQL)**
  - **int executeUpdate(String SQL)** and,
  - **ResultSet executeQuery(String SQL)**

Executing the Statement object

- **boolean execute(String SQL) :**
  - This method is used to execute SQL DDL statements, it returns a boolean value specifying whether the ResultSet object can be retrieved.
  - **Example** :

```
static final String UPDATE_QUERY = "UPDATE Employees set age=30 WHERE id=103";
Boolean ret = stmt.execute(UPDATE_QUERY);
```

- **int executeUpdate(String SQL):**
  - This method is used to execute SQL DML statements such as insert, update, delete.
  - It returns an integer value representing the number of rows affected.
  - **Example** :

```
int rows = stmt.executeUpdate(UPDATE_QUERY)
```

- **ResultSet executeQuery(String SQL) :**
  - This method is used to execute statements that return tabular data (for example, SELECT statement).
  - It returns an object of the class ResultSet.
  - **Example** :

```
static final String QUERY = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(QUERY);
```

Prepared Statement

- The **PreparedStatement** interface extends the Statement interface
- It represents a precompiled SQL statement which can be executed multiple times
- This accepts parameterized SQL quires and you can pass 0 or more parameters to this query
- Initially, this statement uses place holders "**?**" instead of parameters, later on, you can pass arguments to these dynamically using the **setXXX()** methods of the **PreparedStatement** interface.

Creating a PreparedStatement

- You can create an object of the **PreparedStatement** (interface) using the **prepareStatement()** method of the Connection interface
- This method accepts a query (parameterized) and returns a PreparedStatement object
- When you invoke this method the Connection object sends the given query to the database to compile and save it.
- If the query got compiled successfully then only it returns the object.
- To compile a query, the database doesn't require any value, so, you can use (zero or more) **placeholders** (Question marks "?") in the place of values in the query.

Creating a PreparedStatement

- Assume we have the following table created

```
CREATE TABLE Employee(Name VARCHAR(255), Salary INT NOT NULL, Location
VARCHAR(255));
```

- Then, you can use a **PreparedStatement** to insert values into it as shown below.

```
//Creating a Prepared Statement
String query="INSERT INTO Employee(Name, Salary, Location)VALUES(?, ?, ?)";
Statement pstmt = con.prepareStatement(query);
```

Setting values to the place holders

- The PreparedStatement interface provides several setter methods such as

setInt(), setFloat(), setArray(), setDate(), setDouble() etc.. to set values to the place holders of the prepared statement.

- These methods accepts two arguments one is an integer value representing the placement index of the place holder and the other is an int or, String or, float etc… representing the value you need to insert at that particular position
- Example :
    - pstmt.setString(1, "Amit"); // setting name
    - pstmt.setInt(2, 3000); // setting salary
    - pstmt.setString(3, "Hyderabad"); //setting Location

Executing the Prepared Statement

- Once you have created the PreparedStatement object you can execute it using one of the execute() methods of the PreparedStatement interface namely, execute(), executeUpdate() and, executeQuery().
- Example :
  - Boolean ret = Pstmt.execute();
  - ResultSet result = Pstmt.executeQuery();
  - int rows = Pstmt.executeUpdate()

Callable Statement

- A CallableStatement is used to execute stored procedures in the database. Stored procedures are precompiled SQL statements that can be called with parameters. They are useful for executing complex operations that involve multiple SQL statements.
- **Syntax: To create a CallableStatement,**

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

  - Above line Calls a stored procedure named ProcedureName with placeholders ? for input parameters.
- **Methods to Execute:**
  - execute(): Executes the stored procedure and returns a boolean indicating whether the result is a ResultSet (true) or an update count (false).
  - executeQuery(): Executes a stored procedure that returns a ResultSet.
  - executeUpdate(): Executes a stored procedure that performs an update and returns the number of rows affected.
  - **Example 1: Procedure call without parameters**
    - CallableStatement cs = con.prepareCall("{call GetPeopleInfo()}");
    - ResultSet result = cs.executeQuery();

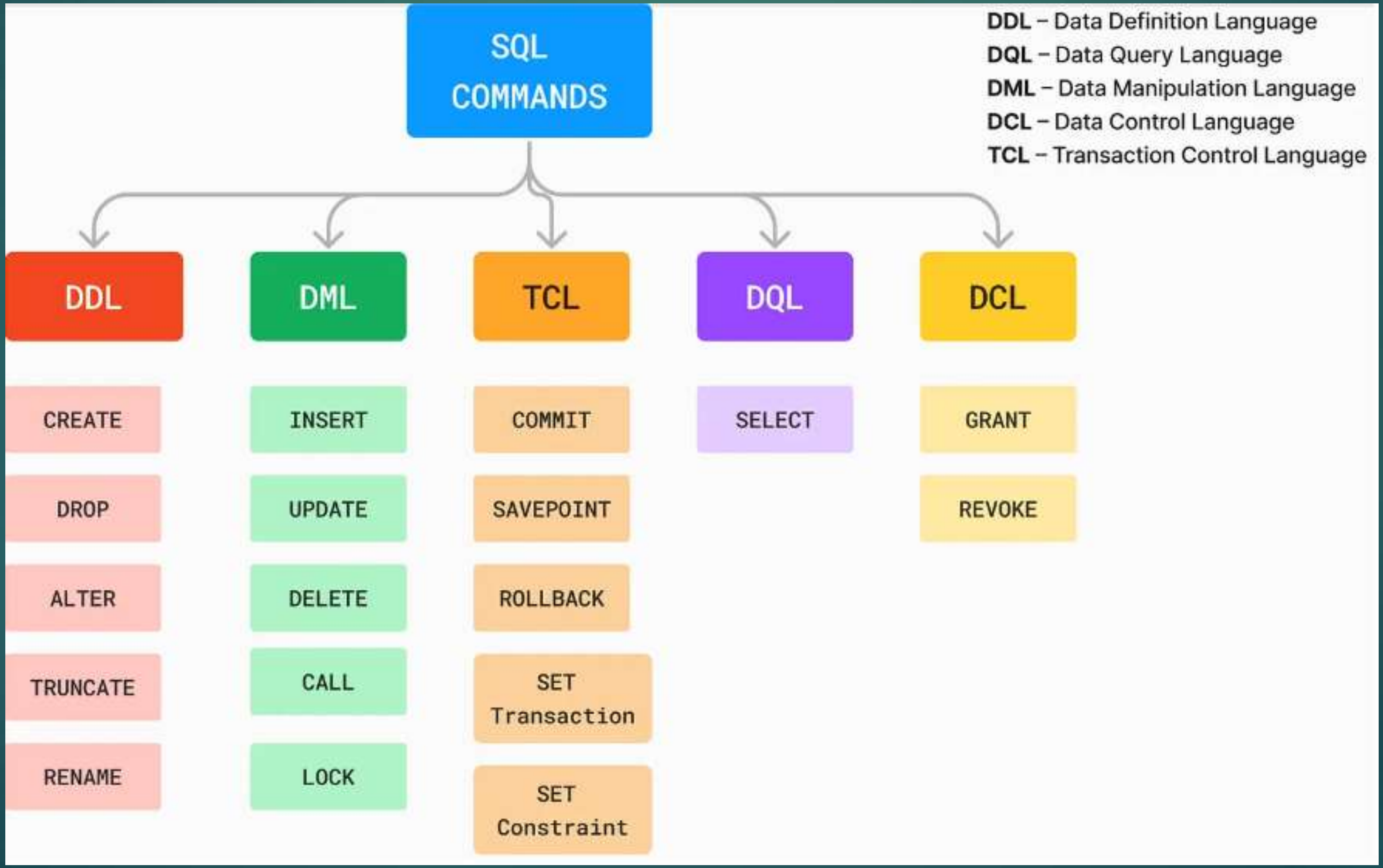
Callable Statement

- **Example 2: Procedure call with input params**

```
CallableStatement stmt=con.prepareCall("{call insertIntoStudent(?,?)}");
 stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();
```

SQL

- Stuctured Query Language used to query database to create insert update and delete Records.
- SQL is further devided into subtypes DDL , DML , DQL , TCL , DCL



SQL - Procedures

- A stored procedure in SQL is a group of SQL queries that can be saved and reused multiple times
- t is very useful as it reduces the need for rewriting SQL queries. It enhances efficiency, reusability, and security in database management.
- Users can also pass parameters to stored procedures so that the stored procedure can act on the passed parameter values.
- Stored Procedures are created to perform one or more DML operations on the Database
- It is nothing but a group of **SQL statements** that accepts some input in the form of parameters, performs some task, and may or may not return a value.
- Syntax :

```
CREATE PROCEDURE procedure_name
(parameter1 data_type, parameter2 data_type, …)
AS
BEGIN
   — SQL statements to be executed
END
```

SQL - Procedures

- **Syntax to Execute the Stored Procedure**

EXEC procedure_name parameter1_value, parameter2_value, ..

Example :

Let create a stored procedure : getMarksScoredByStudent(studentId)

Which returns marks stored by student based on studentId else it would return all student marks by default.

Resultsets

- The object of ResultSet maintains a cursor pointing to a row of a table
- Initially, cursor points to before the first row
- By default, ResultSet object can be moved forward only and it is not updatable
- But we can make this object to move forward and backward direction by passing either **TYPE_SCROLL_INSENSITIVE** or **TYPE_SCROLL_SENSITIVE** in createStatement(int,int) method
- we can make this object as updatable
- Syntax :

```
Statement stmt = con.createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE
                                    );
```

Resultsets

Types of ResultSet

- There are three different characteristics by which **ResultSet** types are differentiated

1. **Scrollability:** Determines whether you can move back and forth in the ResultSet
   - **TYPE_FORWARD_ONLY:** Can only move forward through the rows
   - **TYPE_SCROLL_INSENSITIVE:** Can move forward and backward but changes are not reflect ResultSet
   - **TYPE_SCROLL_SENSITIVE:** Can move forward and backward but changes are affect the ResultSet
2. **Concurrency:** Determines whether you can update the ResultSet
   - **CONCUR_READ_ONLY:** Can only read data
   - **CONCUR_UPDATABLE:** Allows updates to the ResultSet
3. **Holdability:** Determines what happens to the ResultSet when a Transaction is committed.
   - **HOLD_CURSORS_OVER_COMMIT:** The ResultSet remains open after a commit
   - **CLOSE_CURSORS_AT_COMMIT:** The ResultSet closes after a commit

Resultset - TYPE_FORWARD_ONLY

- In forward only ResultSet you can move the cursor only in forward direction.
- By default, a ResultSet is of type forward only.

Resultset - TYPE_SCROLL_INSENSITIVE

- This represents a scrollable ResultSet i.e. the cursor moves in forward or backward directions.
- This type of ResultSet is insensitive to the changes that are made in the database i.e. the modifications done in the database are not reflected in the ResultSet.
- Which means if we have established a connection with a database using JDBC program and retrieved a ResultSet holding all the records in a table named SampleTable and, meanwhile if we add some more records to the table (after retrieving getting the ResultSet), these recent changes will not be reflected in the ResultSet object we previously obtained.

Resultset - TYPE_SCROLL_SENSITIVE

- This represents is a scrollable ResultSet i.e. the cursor moves in forward or backward directions.
- This type of ResultSet is sensitive to the changes that are made in the database i.e. the modifications done in the database are reflected in the ResultSet
- Which means if we have established a connection with a database using a JDBC program and retrieved a ResultSet holding all the records in a table named student.
- Meanwhile, if we have added some more records to the table (after retrieving the ResultSet), these recent changes will be reflected in the ResultSet object we previously obtained.

Resultset - TYPE_SCROLL_SENSITIVE

- Sample program :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
public class ScrollSensitive {
  public static void main(String[] args) throws Exception {
    //Registering the Driver
    DriverManager.registerDriver(new com.mysql.jdbc.Driver());
    //Getting the connection
    String url = " jdbc:oracle:thin:@localhost:1521:xe";
    Connection con = DriverManager.getConnection(url, USER, PASSWORD);
    //Creating a Statement object
    Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    stmt.setFetchSize(1);
  }
}
```

Resultset - Concurrency

CONCUR_READ_ONLY

- This type of result set is not updatable
- i.e. once you get a ResultSet object you cannot update its contents.

CONCUR_UPDATABLE

- A ResultSet with this as concurrency is updatable.
- i.e. once you get a ResultSet object you can update its contents.

Resultset - Holdability

- ResultSet holdability determines whether the ResultSet objects (cursors) should be closed or held open when a transaction (that contains the said cursor/ ResultSet object) is committed using the commit() method of the Connection interface.
- The ResultSet interface provides two values to specify the holdability of a ResultSet namely
    - CLOSE_CURSORS_AT_COMMIT and,
    - HOLD_CURSORS_OVER_COMMIT.
- **CLOSE_CURSORS_AT_COMMIT**
    - If the holdability of the ResultSet object is set to this value, Whenever you commit/save a transaction using the commit() method of the Connection interface, the ResultSet objects created in the current transaction (that are already opened) will be closed.
- **HOLD_CURSORS_OVER_COMMIT**
    - If the holdability of the ResultSet object is set to this value. Whenever you commit/save a transaction using the commit() method of the Connection interface, the ResultSet objects created in the current transaction (that are already opened) will be held open.

Rowset

- RowSet is an interface in java that is present in the javax.sql package
- *RowSet is present in package **javax.sql** while ResultSet is present in package **java.sql***
- The instance of RowSet is the java bean component because it has properties and a java bean notification mechanism
- The JDBC RowSet interface is a RowSet extension. It's a wrapper for the ResultSet object that adds some extra features.
- It is introduced in JDK5
- A JDBC RowSet provides a way to store the data in tabular form
- It makes the data more flexible and easier than a ResultSet
- The connection between the RowSet object and the data source is maintained throughout its life cycle
- RowSet is alternate to ResultSet but is more effective than ResultSet

Rowset

- RowSets are classified into five categories based on how they are implemented which are listed namely as below:
  - JdbcRowSet
  - CachedRowSet
  - WebRowSet
  - FilteredRowSet
  - JoinRowSet
- The advantage of RowSet is as follows
  - It is easy and flexible to use.
  - It is Scrollable and Updatable by default.

Example of JdbcRowSet

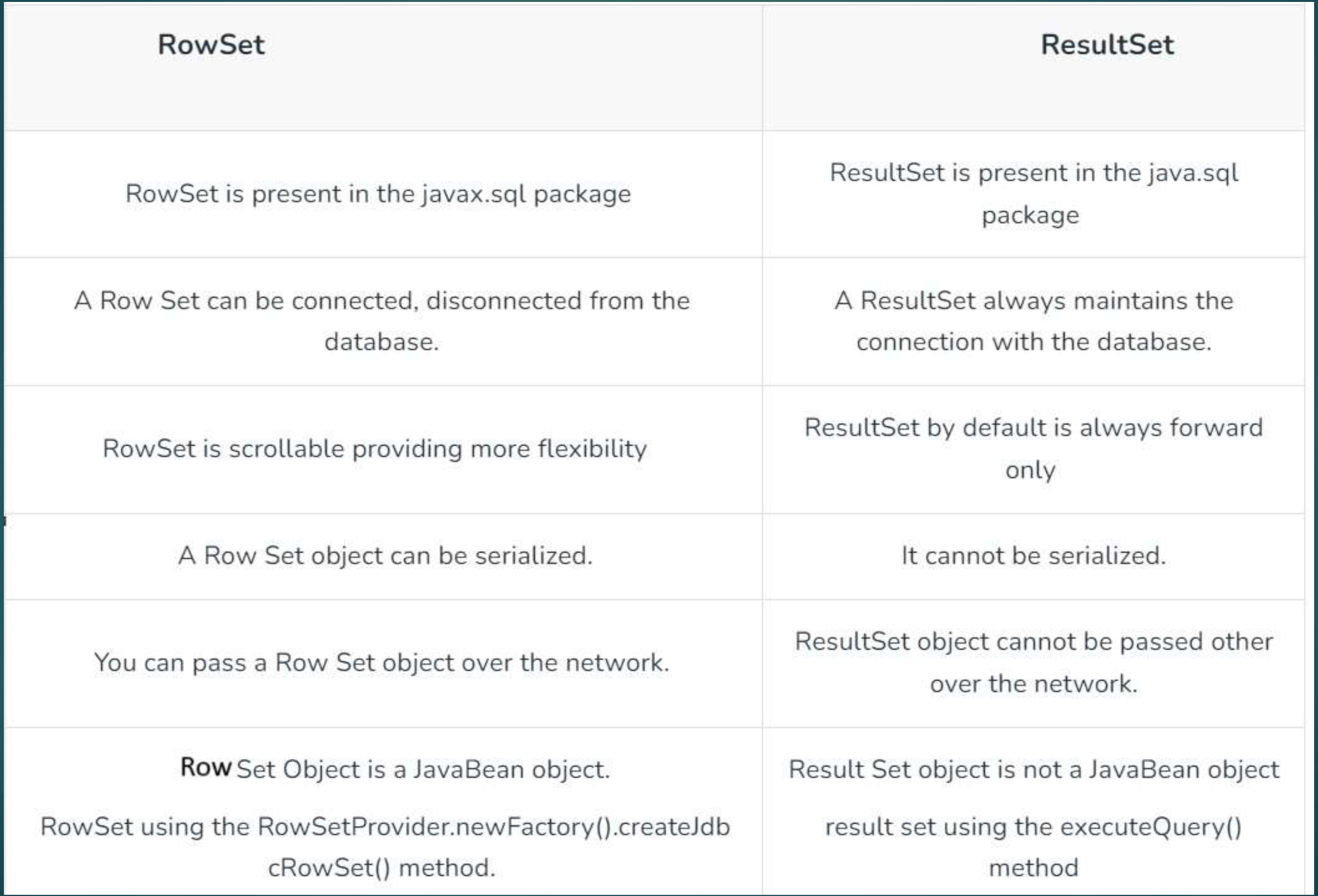
```
import java.sql.*;
import javax.sql.*;
public class RowSetExample {
 public static void main(String[] args) throws Exception {
               Class.forName("oracle.jdbc.driver.OracleDriver");
              //Creating and Executing RowSet
         JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("oracle");
        rowSet.setCommand("select * from emp400");
        rowSet.execute();
```

Example of JdbcRowSet

```
while (rowSet.next()) {
                // Generating cursor Moved event
                System.out.println("Id: " + rowSet.getString(1));
                System.out.println("Name: " + rowSet.getString(2));
                System.out.println("Salary: " + rowSet.getString(3));
            }


        }
}
```

Difference between Rowset and ResultSet



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.
Row Set Object is a JavaBean object. RowSet using the RowSetProvider.newFactory().createJdbcRowSet() method.	Result Set object is not a JavaBean object result set using the executeQuery() method

LAB Assignment 1

1. Demonstrate JDBC connection for MySQL DB
2. Implement Statement, PreparedStatement examples with all three types of execute queries with create, insert, and select queries
3. Implement scrollable Resultset using Scanner and Preparedment.
4. Refer https://www.geeksforgeeks.org/jdbc-result-set/ for program for CRUD operations

Note :

1. Create 2 tables Employee(Eid , Ename ,Salary, address, Did) , Department(Did , Dname) , here Did is foreign key in Employee table referring to Department's Did.
2. Insert minimum 5 rows all using java code
3. Write seperate program for each statement type
4. All have to show oracle and mysql JDBC program and upload it in git hub