JDBC:-

show databases;  
create database Java\_Selenium;  
USE Java\_Selenium;

create table employee(  
empId int,  
empName varchar(40),  
empSal int  
);

show tables;

insert into employee(empId,empName,empSal)values(1001,'Smith',200000);  
insert into employee values(1002,'Martin',300000);  
insert into employee values  
(1003,'Alex',400000),  
(1004,'Turner',500000),  
(1005,'Steve',600000),  
(1006,'Kavin',700000);

insert into employee(empName,empId,empSal)values('Tom',1007,800000);  
select \* from employee;

-- 1. DDL Commands - (Data Definition Language)   
--    create,alter,drop,truncate,rename

-- 2. DML Commands - Data Manipulation Language  
--    insert,update,delete

-- 3. DCL Commands - Data Control Language  
--    Grant,Revoke

-- 4. TCL Commands - Transaction Control Language  
--    Rollback,Commit,Savepoint     
  
-- 5. Data Query Language             
--    SELECT              
use java\_selenium;  
update employee set empSal=900000 where empid IN(1002,1004);  
update employee set empSal=900000,empname='Martin Miller'   
where empid =1002;  
select \* from employee;  
SelecT \* FrOm eMploYEE where empname='turner';  
delete from employee where empid=1003;

1.  Data: Raw Facts

2.  Information: Useful Facts

3.  Database: It contains information in the form of tables.

4.  Table: It is a combination of ROWS and COLUMSN.

5.  ROW:(TUPLE) A number of objects arranged in a straight   
   line.(Horizontal Manner)  
6.  COLUMN:(ATTRIBUTE or FIELD) It is a VERTICAL set of   
    data values within the table.

      PID    Product      PCat  
      23    Gillette    2345  
  
List of Databases:  
==================  
1.  MySQL(RDBMS) - It is a product of Oracle Corporation.

2.  SQL Server(RDBMS) - It is a product of Microsoft.

3.  Oracle(RDBMS) - It is a product of Oracle Corporation

4.  PostgreSQL(RDBMS) - It is not owned by a single company.  
  It is a open-source product developed by the large number  
  of community consists of developers and companies.

5.  NoSQL - It is not owned by a single company.

6.  Apache Cassandra - It is also not owned by any   
  single company. It is managed by Apache Software Foundation(ASF)

7.  MongoDB  - It is owned as well as developed by MongoDB Inc.        
  It is not a Relational Database Management System.  
  It is a NoSQL database. The information in MongoDB  
  is stored in DOCUMENTS.It is known as Document Oriented Database

show databases;  
create database Java\_Selenium;  
USE Java\_Selenium;

create table employee(  
empId int,  
empName varchar(40),  
empSal int  
);

show tables;

insert into employee(empId,empName,empSal)values(1001,'Smith',200000);  
insert into employee values(1002,'Martin',300000);  
insert into employee values  
(1003,'Alex',400000),  
(1004,'Turner',500000),  
(1005,'Steve',600000),  
(1006,'Kavin',700000);

insert into employee(empName,empId,empSal)values('Tom',1007,800000);  
select \* from employee;

-- 1. DDL Commands - (Data Definition Language)   
--    create,alter,drop,truncate,rename

-- 2. DML Commands - Data Manipulation Language  
--    insert,update,delete

-- 3. DCL Commands - Data Control Language  
--    Grant,Revoke

-- 4. TCL Commands - Transaction Control Language  
--    Rollback,Commit,Savepoint     
  
-- 5. Data Query Language             
--    SELECT

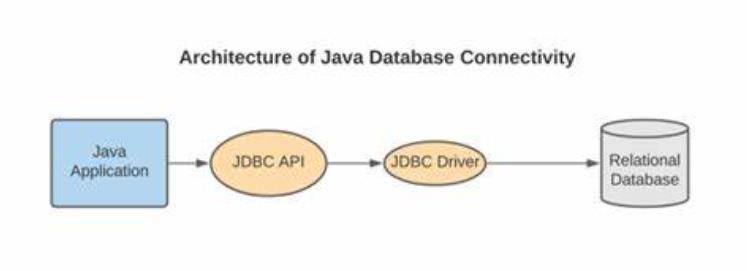
update employee set empSal=900000 where empid IN(1002,1004);  
update employee set empSal=900000,empname='Martin Miller'   
where empid =1002;  
select \* from employee;

**JDBC (Java Database Connectivity)**

**JDBC** is an**API(Application Programming Interface)** that helps applications to communicate with databases, it allows Java programs to connect to a database, run queries, retrieve, and manipulate data. Because of JDBC, Java applications can easily work with different relational databases like MySQL, Oracle, PostgreSQL, and more.

.

**JDBC Architecture**



**Explanation:**

* **Application:** It can be a Java application or servlet that communicates with a data source.
* **The JDBC API:** It allows Java programs to execute SQL queries and get results from the database. **Some key components of JDBC API include**
* Interfaces like Driver, ResultSet, RowSet, PreparedStatement, and Connection that helps managing different database tasks.
* Classes like DriverManager, Types, Blob, and Clob that helps managing database connections.
* **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
* **JDBC drivers:**These drivers handle interactions between the application and the database.

The JDBC architecture consists of two-tier and three-tier processing models to access a database. They are as follows:

**1. Two-Tier Architecture**

A Java Application communicates directly with the database using a JDBC driver. It sends queries to the database and then the result is sent back to the application. For example, in a client/server setup, the user's system acts as a client that communicates with a remote database server.

**Structure (Two-Tier Architecture)**

*Java Application (Client) -> JDBC Driver -> Database*

**2. Three-Tier Architecture**

In this, user queries are sent to a middle-tier services, which interacts with the database. The database results are processed by the middle tier and then sent back to the user.

**Structure:**

*Java Application (Client) -> Application Server -> JDBC Driver -> Database*

**JDBC Components**

There are generally **4** **main components of JDBC** through which it can interact with a database mentioned below:

**1. JDBC API**

It provides various methods and interfaces for easy communication with the database. It includes two key packages

* **java.sql**: This package, is the part of**Java Standard Edition (Java SE) ,** whichcontains the core **interfaces and classes for accessing and processing data** **in relational databases.** It also provides essential functionalities like establishing connections, executing queries, and handling result sets
* **javax.sql**: This package is the part of **Java Enterprise Edition (Java EE) ,**whichextends the capabilities of **java.sql** by offering additional features like connection pooling, statement pooling, and data source management.

It also provides a standard to connect a database to a client application.

**2. JDBC Driver Manager**

**Driver manager is responsible for loading the correct database-specific driver to establish a connection with the database.** It manages the available drivers and ensures the right one is used to process user requests and interact with the database.

**3. JDBC Test Suite**

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

**4. JDBC Drivers**

JDBC drivers are **client-side adapters** (**installed on the client machine, not on the server)** that convert requests from Java programs to a protocol that the RDBMS can understand.

**There are 4 types of JDBC drivers:**

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver (partially java driver)
3. Type-3 driver or Network Protocol driver (fully java driver)
4. **Type-4 driver or Thin driver (completely java driver) - It is a widely used driver.**

**JDBC Classes and Interfaces**

| **Class/Interfaces** | **Description** |
| --- | --- |
| DriverManager | Manages JDBC drivers and establishes database connections. |
| Connection | Represents a session with a specific database. |
| Statement | Used to execute static SQL queries. |
| PreparedStatement | Precompiled SQL statement, used for dynamic queries with parameters. |
| CallableStatement | Used to execute stored procedures in the database. |
| ResultSet | Represents the result set of a query, allowing navigation through the rows. |
| SQLException | Handles SQL-related exceptions during database operations. |

**Steps to Connect to MySQL Database Using JDBC**

**Step 1: Load the JDBC Driver**

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

**Step 2: Establish a Connection**

*Connection connection = DriverManager.getConnection*

*(*

*"jdbc:mysql://localhost:3306/your\_database",*

*"your\_username",*

*"your\_password"*

*);*

**Step 3: Create a Statement**

*Statement statement = connection.createStatement();*

**Step 4: Execute a Query**

*String query = "INSERT INTO students (id, name) VALUES (101, 'John Doe')";*

*int rowsAffected = statement.executeUpdate(query);*

*System.out.println("Rows affected: " + rowsAffected);*

**Step 5: Close the Connection**

*statement.close();*

*connection.close();*

**Create a Simple JDBC Application**

Following Java program demonstrates ***how to establish a MYSQL database connection using JDBC*** *and execute a query.*

**A simple JDBC application**

**import** **java.sql.\***;

**public** **class** **JDBCImpl**{

**public** **static** void main(String[] args)

   {

*// Database URL, username, and password*

*// Replace with your database name*

       String url

           = "jdbc:mysql://localhost:3306/your\_database";

*// Replace with your MySQL username*

       String username = "your\_username";

*// Replace with your MySQL password*

       String password = "your\_password";

*// Updated query syntax for modern databases*

       String query

           = "INSERT INTO emp (id, name) VALUES (1009, 'Harry')";

*// Establish JDBC Connection*

**try** {

*// Load MySQL Type-4 driver class*

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

*// Establish connection*

           Connection c = DriverManager.getConnection(

               url, username, password);

*// Create a statement*

           Statement st = c.createStatement();

*// Execute the query*

           int count = st.executeUpdate(query);

           System.out.println(

               "Number of rows affected by this query: "

               + count);

*// Close the connection*

           st.close();

           c.close();

           System.out.println("Connection closed.");

       }

**catch** (ClassNotFoundException e) {

           System.err.println("JDBC Driver not found: "

                              + e.getMessage());

       }

**catch** (SQLException e) {

           System.err.println("SQL Error: "

                              + e.getMessage());

       }

  }

}

                                                                          -------------------

package jdbcmysql;

import java.sql.\*;  
public class JDBCImpl{  
    public static void main(String[] args)  
    {  
        // Database URL, username, and password

        // Replace with your database name  
        String url  
            = "jdbc:mysql://localhost:3306/java\_selenium";

        // Replace with your MySQL username  
        String username = "root";

        // Replace with your MySQL password  
        String password = "root@123";

        // Updated query syntax for modern databases  
        String query  
            = "INSERT INTO employee(empid, empname,empsal) VALUES (1008, 'Harry',950000)";

        // Establish JDBC Connection  
        try {  
  
  
          // Load MySQL Type-4 driver class  
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establish connection  
            Connection c = DriverManager.getConnection(url, username, password);

            // Create a statement  
            Statement st = c.createStatement();

            // Execute the query  
            int count = st.executeUpdate(query);  
            System.out.println(  
                "Number of rows affected by this query: "  
                + count);

            // Close the connection  
            st.close();  
            c.close();  
            System.out.println("Connection closed.");  
        }  
        catch (ClassNotFoundException e) {  
            System.err.println("JDBC Driver not found: "  
                               + e.getMessage());  
        }  
        catch (SQLException e) {  
            System.err.println("SQL Error: "  
                               + e.getMessage());  
        }  
    }  
}

Primary Key :- A PK is a column or set of columns in the  table that "UNIQUELY IDENTIFIES EACH ROW" within that table.

1.  Duplicate values are not allowed  
2. No NULLs  
3. There will be only 1 PK per table whether that is column level or table level  
  
Why we use PK?  
==============  
We use PK to form relation between the other tables using Foreign Key.

FOREIGN KEY  
===========  
A foreign key is a column or set of columns in a table that refers to the PK column in another table.

In FOREIGN KEY:  
===============  
1. You can enter only PK values.  
2. Duplicate values are NOT ALLOWED except PK values.  
3. NULLs are allowed.

Constraints  
============  
These are the rules that are imposed on the column(s) of  
the database tale.  
1.  NOT NULLs  
2.  UNIQUE "BUT NULLs are allowed"  
3.  CHECK -   
4.  PK  
5.  FK  
6.  DEFAULT

**Prepared Statement**

A Prepared Statement is used in JDBC to execute parameterized SQL queries. It's more secure and efficient than using a regular Statement, especially when dealing with user input, as it helps prevent SQL injection attacks.

**Key Features of PreparedStatement:**

* Allows setting parameters dynamically using? placeholders.
* Precompiled by the database for better performance.
* Automatically escapes strings to prevent SQL injection.

**Example: Using PreparedStatement in JDBC**

Suppose you have a users table like this:

CREATE TABLE users (

   id INT PRIMARY KEY AUTO\_INCREMENT,

   name VARCHAR(100),

   email VARCHAR(100)

);

**Java Code Example:**

import java.sql.\*;

public class PreparedStatementExample {

   public static void main(String[] args) {

       String url = "jdbc:mysql://localhost:3306/your\_database"; // Replace with your DB URL

       String user = "your\_username";

       String password = "your\_password";

       String insertSQL = "INSERT INTO users (name, email) VALUES (?, ?)";

       try (

           Connection conn = DriverManager.getConnection(url, user, password);

           PreparedStatement pstmt = conn.prepareStatement(insertSQL);

       ) {

           // Set values for the placeholders

          pstmt.setString(1, "adgg@gmail.com");

          pstmt.setString(2, "alice@example.com");

           int rowsInserted = pstmt.executeUpdate();

           System.out.println(rowsInserted + " row(s) inserted.");

       } catch (SQLException e) {

          e.printStackTrace();

       }

   }

}

**Explanation:**

* ? in the SQL string is a **placeholder**.
* pstmt.setString(1, "Alice") sets the **first** placeholder to "Alice".
* pstmt.executeUpdate() runs the query (use executeQuery() for SELECT).

**Example: SELECT with PreparedStatement**

String selectSQL = "SELECT \* FROM users WHERE email = ?";

PreparedStatement pstmt = conn.prepareStatement(selectSQL);

pstmt.setString(1, "alice@example.com");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

  System.out.println("User: " + rs.getString("name"));

}

Stored Procedure in MySQL  
=========================  
A procedure or stored procedure is a subroutine stored in a database.   
In the case of MySQL, procedures are written in MySQL and stored in the MySQL database.

A MySQL procedure has:  
1.  A name,  
2.  A parameter list and   
3.  SQL statement(s).

There are four different types of MySQL procedures:   
===================================================  
1.  Procedure with no parameters: A procedure without parameters does not take any input or casts an output indirectly. It is simply called with its procedure name followed by () (without any parameters).   
  It is used for simple queries.   
  Example: Consider two tables author and book:

create table author (author\_id integer primary key,   
                            authorName varchar(30),   
                            email varchar (25),   
              gender varchar (6));

create table book (BookId integer not null unique,   
                        ISBN integer primary key,   
                       book\_name varchar (30) not null,   
                        author integer, ed\_num integer,   
                      price integer, pages integer,   
          foreign key (author) references   
          author (author\_id));

Inserting values into them:  
===========================  
insert into author values   
              (1, "Harry Potter", "hpotter@gmail.com", "Male");  
insert into author values  
              (2, "janet", "janet@gmail.com", "Female");  
insert into book values  
              (1, 001, "Java", 1, 1, 650, 396);  
insert into book values  
              (2, 002, "MySQL", 1, 1, 650, 396);  
insert into book values  
              (3, 003, "Oracle", 2, 1, 799, 500);  
insert into book values  
              (4, 004, "Python", 2, 1, 499, 330);

Procedure (with no parameters) to display all the books:  
========================================================

delimiter //  
create procedure display\_book()   
                      begin   
                      select \*from book;   
                      end   
//

call display\_book(); //

2.   Procedure with IN parameter: An IN parameter is used   
  to take a parameter as input such as an attribute.   
  When we define an IN parameter in a procedure,   
  the calling program has to pass an argument to the   
  stored procedure. In addition, the value of an IN   
  parameter is protected. It means that even if the   
  value of the IN parameter is changed inside the   
  procedure, its original value is retained after the   
  procedure ends (like pass by value). In other words, the procedure only works on the copy of the IN parameter. Example: Procedure to update price of a book taking ISBN of the book and its new price as input: (considering the tables above)

delimiter //  
create procedure update\_price (IN temp\_ISBN varchar(10), IN new\_price integer)  
               begin  
        update book set price=new\_price where ISBN=temp\_ISBN;  
               end; //  
call update\_price(001, 600); //

3.   Procedure with OUT parameter: An OUT parameter is used   
  to pass a parameter as output or display like the   
  select operator, but implicitly (through a set value).   
  The value of an OUT parameter can be changed inside the   
  procedure and its new value is passed back to the   
  calling program. A procedure cannot access the initial   
  value of the OUT parameter when it starts.   
  Example: Procedure to display the highest price among all   
  the books with an output parameter:

delimiter //  
create procedure disp\_max(OUT highestprice integer)  
                 begin  
                 select max(price) into highestprice from book;  
                 end;   
//  
call disp\_max(@Max);  
select @Max;

Output:   
+-----+  
| @M  |  
+-----+  
| 799 |  
+-----+  
1 row in set (0.0005 sec)

4.   Procedure with IN-OUT parameter:   
  An INOUT parameter is a combination of IN and OUT   
  parameters. It means that the calling program may pass   
  the argument, and the stored procedure can modify the   
  INOUT parameter and pass the new value back to the   
  calling program.

Example: Procedure to take gender type input   
('Male'/'Female' here) with an in-out parameter which   
reflects the number of authors falling in that gender   
category/type:

delimiter //  
create procedure disp\_gender(INOUT mfgender integer, IN emp\_gender varchar(6))    
                     begin   
                     select COUNT(gender)   
                         INTO mfgender FROM author where gender = emp\_gender;     
                     end;   
//  
delimiter ;

call disp\_gender(@M, "Male");  
select @M;

call disp\_gender(@F, "Female");  
select @F;

Output:   
+----+  
| @M |  
+----+  
|  1 |  
+----+  
1 row in set (0.0004 sec)  
+----+  
| @F |  
+----+  
|  1 |  
+----+  
1 row in  
set (0.0005 sec)

PRACTICAL - Stored Procedure MySQL

=========================

delimiter //  
create procedure display\_book()   
                      begin   
            select \* from book;   
                      end   
//

call display\_book();

delimiter //  
create procedure update\_price (IN temp\_ISBN varchar(10), IN new\_price integer)  
               begin  
        update book set price=new\_price where ISBN=temp\_ISBN;  
               end;   
//

call update\_price(1,600);  
  
delimiter //  
create procedure disp\_max(OUT highestprice integer)  
                 begin  
                 select max(price) into highestprice from book;  
                 end;   
//

call disp\_max(@M);  
select @M;

call disp\_max(@Max);  
select @Max;

delimiter //  
create procedure disp\_gender(INOUT mfgender integer, IN emp\_gender varchar(6))    
                     begin   
                     select COUNT(gender)   
                         INTO mfgender FROM author where gender = emp\_gender;     
                     end;   
//

call disp\_gender(@M, "Male");  
select @M;

call disp\_gender(@F, "Female");  
select @F;

 package jdbcmysql;

import java.sql.\*;  
public class PreparedStatementEx{  
    public static void main(String[] args) {  
        String url = "jdbc:mysql://localhost:3306/java\_selenium"; // Replace with your DB URL  
        String user = "root";  
        String password = "root@123";

        String insertSQL = "INSERT INTO users (name, email) VALUES (?, ?)";  
        String delSQL ="delete from users where name=?";  
        //String updSQL="update users set name=? where name='Harry Potter'";  
        String updSQL="update users set NAME=? where id=8";  
  
        try(  
            Connection conn = DriverManager.getConnection(url, user, password);  
  
            PreparedStatement pstmtIns = conn.prepareStatement(insertSQL);  
            PreparedStatement pstmtDel = conn.prepareStatement(delSQL);  
            PreparedStatement pstmtUpd= conn.prepareStatement(updSQL);  
  
          )     
          {  
            // Set values for the placeholders  
          pstmtIns.setString(1, "Adam");  
          pstmtIns.setString(2, "adam@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.setString(1, "Kavin");  
          pstmtIns.setString(2, "kavin@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.setString(1, "Miller");  
          pstmtIns.setString(2, "miller@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.executeBatch();  
          //conn.commit();  
          System.out.println("Batch Executed Successfully");  
  
  
  
          /\*  //For deleting  
            pstmtDel.setString(1,"Adam");  
  
            //For updating  
            pstmtUpd.setString(1,"Harry");  
  
            //Result  
            int rowsInserted = pstmtIns.executeUpdate();  
            int rowsDeleted= pstmtDel.executeUpdate();  
            int rowsUpdated= pstmtUpd.executeUpdate();  
  
            System.out.println(rowsInserted + " row(s) inserted.");  
            System.out.println(rowsDeleted+ " row(s) deleted.");  
            System.out.println(rowsUpdated+ " row(s) updated.");  
        \*/  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}

package jdbcmysql;

import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Future;

// Callable is a functional interface with only 1 abstract method "call()"  
// It is a part of "java.util.concurrent" package.  
// This interface is the parent interface. Callable interface is   
// not extending any Parent interface  
/\* With Runnable interface we cannot return a value.  
\* With Callable interface we can return a value.  
\*/

class MyCallable implements Callable<String>{  
  public String call() throws Exception{  
    Thread.sleep(1000);  
    return "With Callable interface we can return a value";  
  }  
  
}

public class CallableExample {  
  public static void main(String[] args) throws Exception{  
    //Step 1 Creates a Thread pool and invoke the call()   
    //method of a Callable Interface  
    ExecutorService executor = Executors.newSingleThreadExecutor();  
  
    // Step 2: Creates a Callable task  
    MyCallable task = new MyCallable();   
  
    //Step 3: We will return the future object  
    Future<String> future = executor.submit(task);   
    String result = future.get();  
    System.out.println(result);  
    executor.shutdown();  
  }  
}