**ADVANCED JAVA**

**With Core Java knowledge we can develop Stand Alone Applications.**

**The Applications which are running on a Single Machine are called *Stand Alone Applications.***

**Eg: Calculator, MS Word**

**Any Core Java Application**

**If we want to develop Web Applications then we should go for Advanced Java.**

**The Applications which are providing Services over the Web are called *Web Applications.***

**Eg:mallisoftvideos.com, gmail.com, facebook.com, durgasoft.com**

**In Java we can develop Web Applications by using the following Technologies...**

**◈ JDBC**

**◈ Servlets**

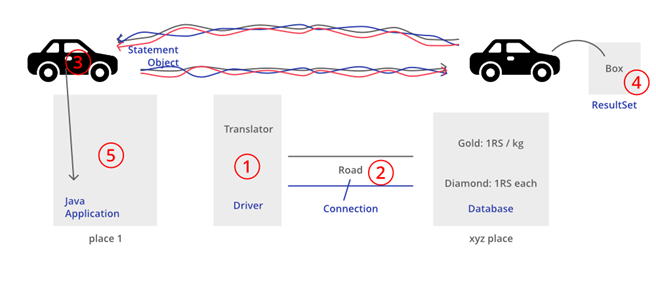
**◈ JSP's**

**Where ever Presentation Logic is required i.e. to display something to the End User then we should go for JSP i.e. JSP meant for View Component.**

**From Java Application (Normal Java Class OR Servlet) if we want to communicate with Database then we should go for JDBC.**

**Eg: To get Astrology Information from Database To get Mails Information from Database**

### **JDBC in Simple Way**

****

**Driver (Translator):**

**To convert Java specific calls into Database specific calls and Database specific calls into Java calls.**

**Connection (Road):**

**By using Connection, Java Application can communicate with Database.**

**Statement (Vehicle):**

**By using Statement Object we can send our SQL Query to the Database and we can get Results from Database.**

**ResultSet:**

**ResultSet holds Results of SQL Query.**

**Basic componets of jdbc:**

**1.driver**

**2.connection**

**3.Statement**

**4.Result set**

**Steps to develop JDBC Application:**

**1.** **Load and Register Driver**

**2.** **Establish Connection between Java Application and Database**

**3.** **Create Statement Object**

**4.** **Send and Execute SQL Query**

**5.** **Process Results from ResultSet**

**6. Close Connection**

## **Storage Areas**

**As the Part of our Applications, we required to store our Data like Customers Information, Billing Information, Calls Information etc..**

**To store this Data, we required Storage Areas. There are 2 types of Storage Areas.**

**1)** **Temporary Storage Areas**

**2)** **Permanent Storage Areas**

**Temporary Storage Areas:**

**These are the Memory Areas where Data will be stored temporarily.**

**Eg: All JVM Memory Areas (like Heap Area, Method Area, Stack Area etc). Once JVM shutdown all these Memory Areas will be cleared automatically.**

**Permanent Storage Areas:**

**Also known as Persistent Storage Areas. Here we can store Data permanently.**

**Eg: File Systems, Databases, Data warehouses, Big Data Technologies etc**

**File Systems:**

**File Systems can be provided by Local operating System.**

**File Systems are best suitable to store very less Amount of Information.**

**Limitations:**

**1)** **We cannot store huge Amount of Information.**

**2)** **There is no Query Language support and hence operations will become very complex.**

**3)** **There is no Security for Data.**

**4)** **There is no Mechanism to prevent duplicate Data. Hence there may be a chance of Data Inconsistency Problems.**

**To overcome the above Problems of File Systems, we should go for Databases.**

**Databases:**

**1)** **We can store Huge Amount of Information in the Databases.**

**2)** **Query Language Support is available for every Database and hence we can perform Database Operations very easily.**

**3)** **To access Data present in the Database, compulsory *username* and *pwd* must be required. Hence Data is secured.**

**4)** **Inside Database Data will be stored in the form of Tables. While developing Database Table Schemas, Database Admin follow various Normalization Techniques and can implement various Constraints like Unique Key Constrains, Primary Key Constraints etc which prevent Data Duplication. Hence there is no chance of Data Inconsistency Problems.**

**Limitations of Databases:**

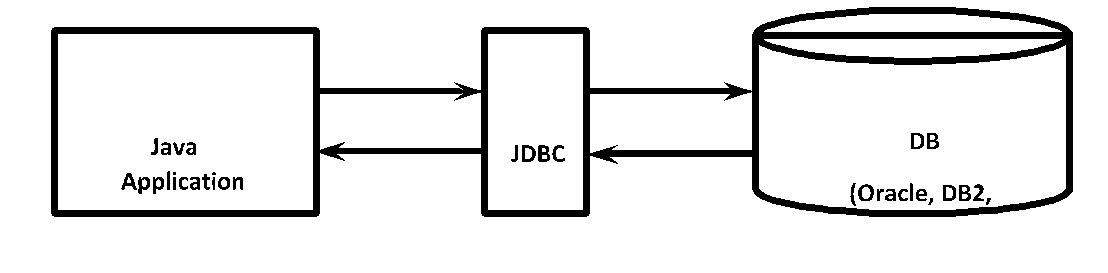
**1)** **Database cannot hold very Huge Amount of Information like Terabytes of Data.**

**2)** **Database can provide support only for Structured Data (Tabular Data OR Relational Data) and cannot provide support for Semi Structured Data (like XML Files) and Unstructured Data (like Video Files, Audio Files, Images etc)**

**To overcome this Problems we should go for more Advanced Storage Areas like Big Data Technologies, Data warehouses etc**

## **JDBC**

**· JDBC is a Technology, which can be used to communicate with Database from Java Application.**

****

**· JDBC is the Part of Java Standard Edition (J2SE|JSE)**

**· JDBC is a Specification defined by Java Vendor (Sun Micro Systems) and implemented by Database Vendors.**

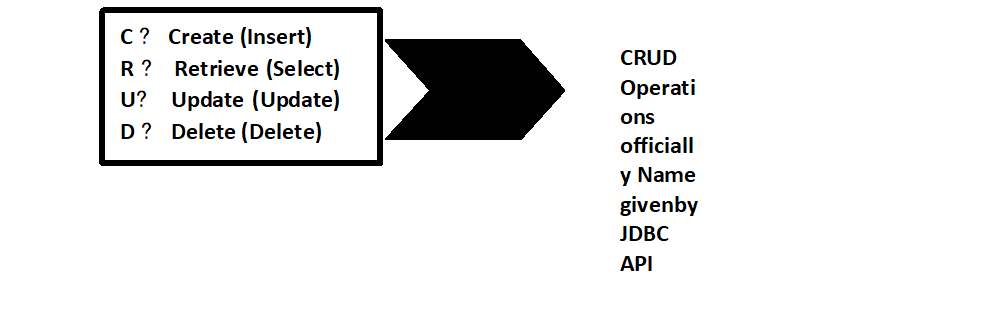
**· Database Vendor provided Implementation is called "Driver Software".**

**JDBC Features:**

**1)** **JDBC API is Standard API. We can communicate with any Database without rewriting our Application i.e. it is Database Independent API.**

**2)** **JDBC Drivers are developed in Java and hence JDBC Concept is applicable for any Platform. i.e. JDBC Is Platform Independent Technology.**

**3)** **By using JDBC API, we can perform basic CRUD Operations very easily.**

****

**These Operations also known as CURD/ SCUD Operations (Ameerpet People created Terminology)**

**We can also perform Complex Operations (like Inner Joins, Outer Joins, calling Stored Procedures etc) very easily by using JDBC API.**

**1)** **JDBC API supported by Large Number of Vendors and they developed multiple Products based on JDBC API.**

**List of supported Vendors we can check in the link**

[***http://www.oracle.com/technetwork/java/index-136695.html***](http://www.oracle.com/technetwork/java/index-136695.html)

**JDBC Versions:**

**v JDBC 3.0 is Part J2SE 1.4**

**v No Update in Java SE 5.0**

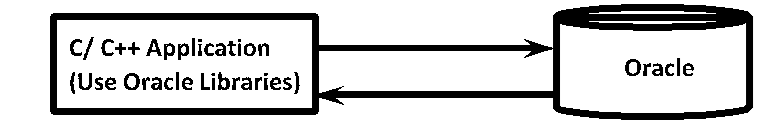
**v JDBC 4.0 is Part Java SE 6.0**

**v JDBC 4.1 is Part Java SE 7.0**

**v JDBC 4.2 is Part Java SE 8.0**

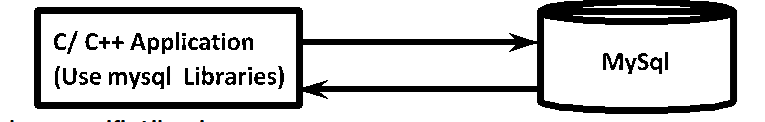
**Evolution of JDBC:**

**·**  **If we want to communicate with Database by using C OR C++, compulsory we have to use database specific Libraries in our Application directly.**

****

**· In the above Diagram C OR C++ Application uses Oracle specific Libraries directly.**

**·   
 The Problem in this Approach is, if we want to migrate Database to another Database then we have to rewrite Total Application once again by using new Database specific Libraries**

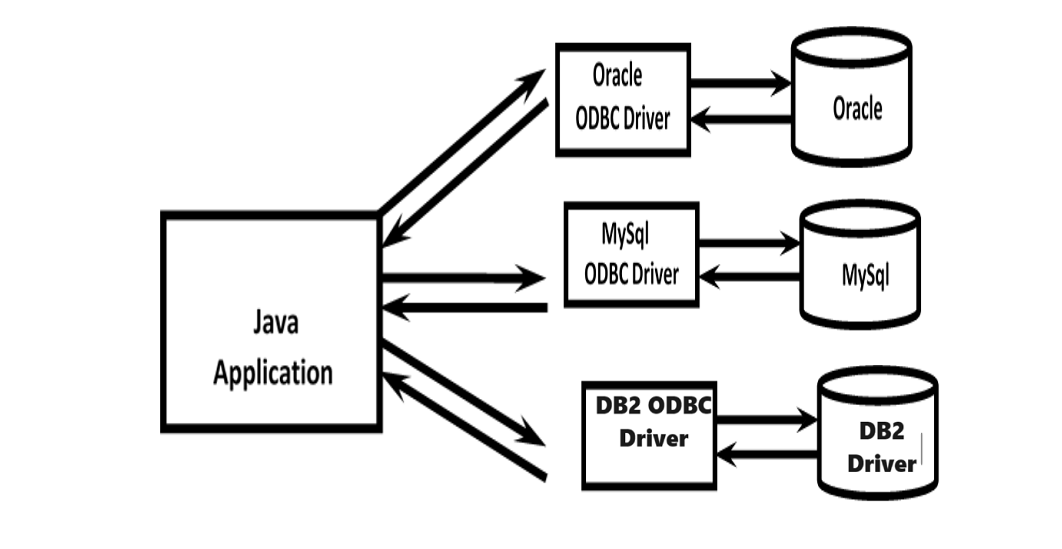
**.**

**· The Application will become Database Dependent and creates Maintenance Problems.**

**· To overcome this Problem, Microsoft People introduced "ODBC" Concept in 1992. It is Database Independent API.**

**· With ODBC API, Application can communicate with any Database just by selecting corresponding ODBC Driver.**

**· We are not required to use any Database specific Libraries in our Application. Hence our Application will become Database Independent.**

****

**Limitations of ODBC:**

**1)** **ODBC Concept will work only for Windows Machines. It is Platform Dependent Technology.**

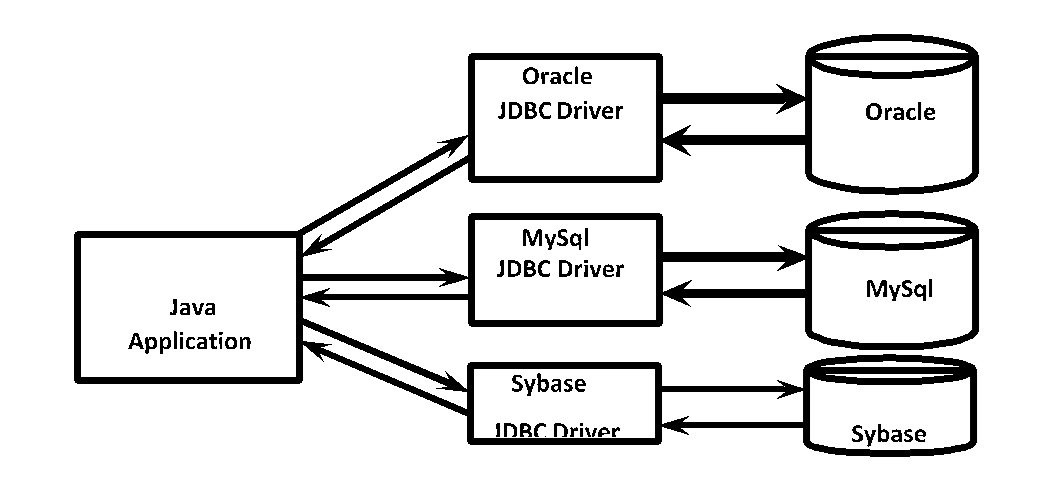
**2)** **ODBC Drivers are implemented in C Language. If we use ODBC for Java Applications, then Performance will be down because of internal conversions from Java to C and C to Java.**

**· Because of above Reasons, ODBC Concept is not suitable for Java Applications.**

**· For Java Applications, SUN People introduced JDBC Concept.**

**· JDBC Concept Applicable for any Platform. It is Platform Independent Technology.**

**· JDBC Drivers are implemented in Java. If we use JDBC for Java Applications, then internal Conversions are not required and hence there is no Effect on Performance.**

****

| **Sybase** |
| --- |

| **Sybase**  **JDBC Driver** |
| --- |

**\*\*\*Note:**

**1)** **ODBC Concept is applicable for any Database and for any Language, but only for Windows Platform.**

**2)** **JDBC Concept is Applicable for any Platform and for any Database, but only for Java Language.**

**Differences Between JDBC and ODBC**

| **ODBC** | **JDBC** |
| --- | --- |
| **1) ODBC Stands for Open Database Connectivity** | **1) JDBC Stands for Java Database Connectivity** |
| **2) Introduced by Microsoft.** | **2) Introduced by Sun Micro Systems.** |
| **3) We can Use ODBC for any Languages like C, C++, Java, Etc.** | **3) We can Use JDBC only for Java Language.** |
| **4) We can use ODBC only for Windows Platforms.** | **4) We can use JDBC for any Platform.** |
| **5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.** | **5) Mostly JDBC Drivers are developed in Java.** |
| **6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.** | **6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.** |

**Steps to prepare JDBC Applications**

**------------------------------------------------------**

**1. Load and Register the Driver.**

**2. Establish connection between the java specific calls to database specific calls**

**3. Create the statement**

**4. To prepare and send and execute sql query to the data.**

**5. Process results from ResultSet.**

**6. Close the connection.**

**1.Load and Register the Driver.**

**--------------------------------------------------**

**The forName() method of a class is used to register the driver class. this method is used to dynamically load the driver class.**

**Syntax of forName method();**

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

**Class.forName(“com.mysql.cj.jdbc.Driver)--à mysql**

**Class.forName(“oracle.jdbc.driver.OracleDriver”)----àoracle.**

**2. Establish connection between the java specific calls to database specific calls**

**--------------------------------------------------------------------------------**

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

**Syntax:**

**1) Public static Connection getConnection(Stringurl)throws SqlException**

**2) Public static Connection getConnection(String url,String uname,String password);**

**Ex:- Connection con=DriverManager.getConnection(“jdbc:mysql://localhost:3306/mydb”,”root”,”root”);àmysql**

**Connection con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:mydb”,”scoot”,”tiger”)-àoracle**

**3. Create the statement**

**-------------------------------------**

**Public Statement createStatement();**

**Ex: Statement st=con.createStatement();**

**4. To prepare and send and execute sql query to the data.**

**—---------------------------------------------------------------------------**

**String sqlQuery=st.executeUpdate(“use non-select operations”);--à return type is always int.**

**String sqlQuery=st.executeQuery(“ select operations”)-àreturn type is resultset**

**String sqlQuery=st.execute(“both select and non select queries)-à return type is always Boolean.**

**True--à selectquery**

**False--à non-selectquery.**

**5. Process results from ResultSet.**

**-------------------------------------**

**+-----+-------------+----------+-------------+**

**| eno | ename | esal | eaddr |**

**+-----+-------------+----------+-------------+**

**| 101 | John Doe | 50000.00 | New York |**

**| 102 | Jane Smith | 60000.00 | Los Angeles |**

**| 103 | Bob Johnson | 55000.00 | Chicago |**

**| 444 | vinny | 87398.00 | sec |**

**+-----+-------------+----------+-------------+**

**While(rs.next())**

**{**

**System.out.println(rs.getInt(1) + ”----“ + rs.getString(2) + ”------“ + rs.getDouble(3) +”------“ + rs.getString(4));**

**System.out.println(rs.getInt(eno) + ”---“ + rs.getString(ename) + ”------“ + rs.getDouble(esal) + “-----“ +getString(eaddr) );**

**}**

**6.. Close the connection**

**—-----------------------------------------**

**By using the close method we can close the connection**

**Con.close();**

**Application-1: How to Create a Table**

**package com.codegnan.jdbcpractice;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class CreateEmployeeTable {**

**// JDBC URL, username, and password of MySQL server**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost/Adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**// SQL command to create the employee table**

**String sql = "CREATE TABLE IF NOT EXISTS employee (" + "eno INT AUTO\_INCREMENT PRIMARY KEY,"**

**+ "ename VARCHAR(100)," + "esal DECIMAL(10, 2)," + "eaddr VARCHAR(255)" + ")";**

**Connection connection = null;**

**Statement statement = null;**

**try {**

**// Establish connection**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**statement = connection.createStatement();**

**// Create the employee table**

**statement.execute(sql);**

**System.*out*.println("Employee table created successfully.");**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**// Close statement**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**// Close connection**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-2: How To Delete A Table**

**package com.codegnan.jdbcpractice;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class DeleteTableExample {**

**// JDBC URL, username, and password of MySQL server**

**static final String JDBC\_URL = "jdbc:mysql://localhost/Adjava";**

**static final String USERNAME = "root";**

**static final String PASSWORD = "root";**

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

**// Name of the table to delete**

**String tableName = "employee";**

**Connection connection = null;**

**Statement statement = null;**

**try {**

**// Establish connection**

**connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);**

**statement = connection.createStatement();**

**// SQL command to delete the table**

**String sql = "DROP TABLE IF EXISTS " + tableName;**

**// Execute the SQL command**

**statement.executeUpdate(sql);**

**System.out.println("Table " + tableName + " deleted successfully.");**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**// Close statement**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**// Close connection**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application3:- write a Jdbc program to insert records on employee table?**

**package** com.codegnan.jdbcprogrames;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** InsertEmployeeTabelRecords {

// JDBC URL, username, and password of MySQL server

**private** **static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/Adjava";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** **final** String ***driver*** ="com.mysql.cj.jdbc.Driver";

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

Connection connection = **null**;

Statement statement = **null**;

**try** {

// Register JDBC driver

Class.*forName*(***driver***);

// Open a connection

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// Create a statement

statement = connection.createStatement();

// SQL insert statement

String insertSQL = "INSERT INTO employee (eno,ename,eaddr,esal) VALUES (111,'ali','vij',23000)";

// Execute the insert statement

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

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

**if**(rowsAffected==1) {

System.***out***.println("records inserted succesfully");

}

**else** {

//there is no specific logic

}

} **catch** (SQLException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **finally** {

// Close resources

**try** {

**if**(statement != **null**) {

statement.close();

}

**else**

{

//there is no specific logic

}

**if** (connection != **null**) {

connection.close();

}

**else** {

//there is no specific logic

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

**Note:**

From SQL Plus Command Prompt, if we are performing any Database Operations then compulsory we should perform Commit Operation explicitly because Auto Commit Mode is not enabled.

From JDBC Application if we perform any Database Operations then the Results will be committed automatically and we are not required to Commit explicitly, because in JDBC Auto Commit is enabled by default.

Formatting SQL Queries With Dynamic Input

String sqlQuery="insert into employees values(100,'durga',1000,'Hyd')";

If Data is available in the following Variables eno, ename, esal, eaddr

String sqlQuery="insert into employees values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')";

It is highly recommended to use String Class format() Method while writing SQL Queries with Dynamic Input.

**Application-4: write a Jdbc program to insert a record with Dynamic\_Input ?**

**package** com.codegnan.jdbcprogrames;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**public** **class** InsertRecordByDynamicInput{

// JDBC URL, username, and password of MySQL server

**private** **static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/Adjava";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** **final** String ***driver*** ="com.mysql.cj.jdbc.Driver";

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

Connection connection = **null**;

Statement statement = **null**;

Scanner scanner = **new** Scanner(System.***in***);

**try** {

// Register JDBC driver

Class.*forName*(***driver***);

// Open a connection

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// Create a statement

statement = connection.createStatement();

// Prompt user for input

System.***out***.println("Enter employee no :");

**int** eno=scanner.nextInt();

System.***out***.println("Enter employee name: ");

String ename = scanner.next();

System.***out***.println("Enter employee salary: ");

**double** esal = scanner.nextDouble();

System.***out***.println("Enter employee Address: ");

String eaddr=scanner.next();

// SQL insert statement

String insertSQL = "INSERT INTO employee (eno,ename,esal,eaddr) VALUES ('" + eno + "', " + ename + ", " + esal + "," + eaddr +")";

// Execute the insert statement

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

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

**if**(rowsAffected==1)

{

System.***out***.println("records inserted succesfully");

}

**else**

{

//there is no specific logic

}

} **catch** (SQLException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **finally** {

// Close resources

**try** {

**if** (statement != **null**)

{

statement.close();

}

**else**

{

//there is no specific logic

}

**if** (connection != **null**)

{

connection.close();

}

**else**

{

//there is no specific logic

}

scanner.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

**Application-6: write a Jdbc program to insert a multiple records by using dynamic input data using for loop?**

**package** com.codegnan.jdbcprogrames;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**public** **class** InsertMultipleRecords {

// JDBC URL, username, and password of MySQL server

**private** **static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/Adjava";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** **final** String ***DRIVER*** = "com.mysql.cj.jdbc.Driver";

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

Connection connection = **null**;

Statement statement = **null**;

Scanner scanner = **new** Scanner(System.***in***);

**try** {

// Register JDBC driver

Class.*forName*(***DRIVER***);

// Open a connection

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// Create a statement

statement = connection.createStatement();

// Prompt user for number of records to insert

System.***out***.print("Enter number of records to insert: ");

**int** numRecords = scanner.nextInt();

scanner.nextLine(); // Consume newline character

// Input data for each record and execute insert statement

**for** (**int** i = 0; i < numRecords; i++) {

System.***out***.println("Enter details for record " + (i + 1) + ":");

System.***out***.print("Employee Number: ");

**int** eno = scanner.nextInt();

scanner.nextLine(); // Consume newline character

System.***out***.print("Employee Name: ");

String ename = scanner.nextLine();

System.***out***.print("Employee Salary: ");

**double** esal = scanner.nextDouble();

scanner.nextLine(); // Consume newline character

System.***out***.print("Employee Address: ");

String eaddr = scanner.nextLine();

// SQL insert statement with dynamic values

String insertSQL = "INSERT INTO employee (eno, ename, esal, eaddr) VALUES (" +

eno + ", '" + ename + "', " + esal + ", '" + eaddr + "')";

// Execute the insert statement

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

**if** (rowsAffected == 1) {

System.***out***.println("Record " + (i + 1) + " inserted successfully");

} **else** {

System.***out***.println("Failed to insert record " + (i + 1));

}

}

} **catch** (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} **finally** {

// Close resources

**try** {

**if** (statement != **null**) {

statement.close();

} **else** {

// No specific logic

System.***out***.println("Statement was not initialized");

}

**if** (connection != **null**) {

connection.close();

} **else** {

// No specific logic

System.***out***.println("Connection was not initialized");

}

**if** (scanner != **null**) {

scanner.close();

} **else** {

// No specific logic

System.***out***.println("Scanner was not initialized");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

Application-7: Write a Jdbc Program To update\_data in mysql?

**package** com.codegnan.jdbcprogrames;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**public** **class** UpdateData {

// JDBC URL, username, and password of MySQL server

**private** **static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/Adjava";

**private** **static** **final** String ***USERNAME*** = "root";

**private** **static** **final** String ***PASSWORD*** = "root";

**private** **static** **final** String ***DRIVER*** = "com.mysql.cj.jdbc.Driver";

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

Connection connection = **null**;

Statement statement = **null**;

Scanner scanner = **new** Scanner(System.***in***);

**try** {

// Register JDBC driver

Class.*forName*(***DRIVER***);

// Open a connection

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// Create a statement

statement = connection.createStatement();

// Prompt user for update details

System.***out***.print("Enter employee number to update: ");

**int** eno = scanner.nextInt();

scanner.nextLine(); // Consume newline character

System.***out***.print("Enter new employee name: ");

String newName = scanner.nextLine();

System.***out***.print("Enter new employee salary: ");

**double** newSalary = scanner.nextDouble();

scanner.nextLine(); // Consume newline character

System.***out***.print("Enter new employee address: ");

String newAddress = scanner.nextLine();

// SQL update statement with dynamic values

String updateSQL = "UPDATE employee SET ename = '" + newName + "', esal = " + newSalary +

", eaddr = '" + newAddress + "' WHERE eno = " + eno;

// Execute the update statement

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

**if** (rowsAffected > 0) {

System.***out***.println("Data updated successfully for employee number " + eno);

} **else** {

System.***out***.println("No data found for employee number " + eno + ". Update operation failed.");

}

} **catch** (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} **finally** {

// Close resources

**try** {

**if** (statement != **null**) {

statement.close();

}

**if** (connection != **null**) {

connection.close();

}

**if** (scanner != **null**) {

scanner.close();

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

**Application-8: How to Update Multiple Records in the Table**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

public class UpdateMultipleRecordsBasedOnSalary {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER = "com.mysql.cj.jdbc.Driver";

public static void main(String[] args) {

Connection connection = null;

Statement statement = null;

Scanner scanner = new Scanner(System.in);

try {

// Register JDBC driver

Class.forName(DRIVER);

// Open a connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a statement

statement = connection.createStatement();

// Prompt user for update details

System.out.print("Enter employee salary threshold: ");

double salaryThreshold = scanner.nextDouble();

System.out.print("Enter salary increment: ");

double salaryIncrement = scanner.nextDouble();

// SQL update statement with dynamic values and WHERE clause

String updateSQL = "UPDATE employee SET esal = esal + " + salaryIncrement +

" WHERE esal <" + salaryThreshold;

// Execute the update statement

int rowsAffected = statement.executeUpdate(updateSQL);

System.out.println(rowsAffected + " records updated successfully");

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

// Close resources

try {

if (statement != null) {

statement.close();

}

if (connection != null) {

connection.close();

}

if (scanner != null) {

scanner.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

**Application-9(A): How to Delete single Record from the Table**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

public class DeleteSingleRecord {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER = "com.mysql.cj.jdbc.Driver";

public static void main(String[] args) {

Connection connection = null;

Statement statement = null;

Scanner scanner = new Scanner(System.in);

try {

// Register JDBC driver

Class.forName(DRIVER);

// Open a connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a statement

statement = connection.createStatement();

// Prompt user for the primary key value of the record to delete

System.out.print("Enter the employee number to delete: ");

int enoToDelete = scanner.nextInt();

// SQL delete statement with WHERE clause

String deleteSQL = "DELETE FROM employee WHERE eno = " + enoToDelete;

// Execute the delete statement

int rowsAffected = statement.executeUpdate(deleteSQL);

if (rowsAffected > 0) {

System.out.println("Record with employee number " + enoToDelete + " deleted successfully");

} else {

System.out.println("No record found with employee number " + enoToDelete);

}

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

// Close resources

try {

if (statement != null) {

statement.close();

}

if (connection != null) {

connection.close();

}

if (scanner != null) {

scanner.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

**Application-9(B): How to Delete multiple Records from the Table**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

public class DeleteMultipleRecords {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER = "com.mysql.cj.jdbc.Driver";

public static void main(String[] args) {

Connection connection = null;

Statement statement = null;

Scanner scanner = new Scanner(System.in);

try {

// Register JDBC driver

Class.forName(DRIVER);

// Open a connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a statement

statement = connection.createStatement();

// Prompt user for the condition to delete records

System.out.print("Enter the salary threshold below which records will be deleted: ");

double salaryThreshold = scanner.nextDouble();

// SQL delete statement with WHERE clause

String deleteSQL = "DELETE FROM employee WHERE esal < " + salaryThreshold;

// Execute the delete statement

int rowsAffected = statement.executeUpdate(deleteSQL);

System.out.println(rowsAffected + " records deleted successfully");

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

// Close resources

try {

if (statement != null) {

statement.close();

}

if (connection != null) {

connection.close();

}

if (scanner != null) {

scanner.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

Database Operations: Select Operation

**Use Cases Of Select Operation:**

**1.** **Display all Trains Information from HYD to Mumbai**

**2.** **Display all Book Names written by Greene**

**3.** **Display all Movies Names in HYD City**

**Application-10: How to Select all Rows from the Table**

**package** com.codegnan.jdbcexamples;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** SelectAllRows {

**static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/adjava";

**static** **final** String ***USERNAME*** = "root";

**static** **final** String ***PASSWORD*** = "root";

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

Connection connection = **null**;

Statement statement = **null**;

ResultSet resultSet = **null**;

**try** {

// establish connection.

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// create statement object.

statement = connection.createStatement();

// prepare sql query

String selectSQL = "select\*from employees";

// execute the sql query

resultSet = statement.executeQuery(selectSQL);

System.***out***.println("ENO\tENAME\tESAL\tEADDR");

System.***out***.println("-------------------------------------------------------");

// process the resultSet.

**while** (resultSet.next()) {

System.***out***.println(resultSet.getInt(1) + "\t" + resultSet.getString(2) + "\t" + resultSet.getDouble(3)

+ "\t" + resultSet.getString(4));

System.***out***.println();

}

} **catch** (SQLException e) {

e.printStackTrace();

} **finally** {

**if** (statement != **null**) {

**try** {

statement.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (resultSet != **null**) {

**try** {

resultSet.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (connection != **null**) {

**try** {

connection.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

}

**11) How to select Particular Columns.**

**package** com.codegnan.jdbcexamples;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** SelectParticularColumns {

**static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/adjava";

**static** **final** String ***USERNAME*** = "root";

**static** **final** String ***PASSWORD*** = "root";

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

Connection connection = **null**;

Statement statement = **null**;

ResultSet resultSet = **null**;

**try** {

// establish connection.

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// create statement object.

statement = connection.createStatement();

// prepare sql query

String selectSQL = "select eno,ename from employees";

// execute the sql query

resultSet = statement.executeQuery(selectSQL);

System.***out***.println("ENO\tENAME");

System.***out***.println("-------------------------------------------------------");

// process the resultSet.

**while** (resultSet.next()) {

System.***out***.println(resultSet.getInt(1) + "\t" + resultSet.getString(2));

System.***out***.println();

}

} **catch** (SQLException e) {

e.printStackTrace();

} **finally** {

**if** (statement != **null**) {

**try** {

statement.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (resultSet != **null**) {

**try** {

resultSet.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (connection != **null**) {

**try** {

connection.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

}

Aggregate Functions

Oracle Database defines several Aggregate Functions to get Summary Results like the Number of Records, Maximum Value of a particular Column etc

count(\*)➔ Returns The Number of Records

max(esal)➔ Returns Maximum Salary

min(esal)➔ Returns Minimum Salary

**Application-12: To Display Number of Rows by SQL Aggregate Function count(\*)**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class DisplayNumberOfRecords {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**ResultSet resultSet = null;**

**try {**

**// establish connection.**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// create statement object.**

**statement = connection.createStatement();**

**// prepare sql query;**

**String sqlQuery = "select count(\*) as row\_count from employees";**

**// execute the query.**

**resultSet = statement.executeQuery(sqlQuery);**

**while (resultSet.next()) {**

**int rowCount = resultSet.getInt("row\_count");**

**System.*out*.println("the number of records in a employee table is : " + rowCount);**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (resultSet != null) {**

**try {**

**resultSet.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-13: How to Select highest salaried Employee Information by using SQL Aggregate Function Max**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class HighestSalariedEmployee {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**ResultSet resultSet = null;**

**try {**

**// establish connection.**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// create statement object.**

**statement = connection.createStatement();**

**String sqlQuery = "select\*from employees where esal=(select max(esal) from employees)";**

**// execute the sql query**

**resultSet = statement.executeQuery(sqlQuery);**

**System.*out*.println("ENO\tEname\tESAL\tEaddr");**

**System.*out*.println("=================================================");**

**while (resultSet.next()) {**

**System.*out*.println(resultSet.getInt(1) + "\t" + resultSet.getString(2) + "\t" + resultSet.getDouble(3)**

**+ "\t" + resultSet.getString(4));**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (resultSet != null) {**

**try {**

**resultSet.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-14: How to Select highest salaried Employee Information by using SQL Aggregate Function Min**

**package** com.codegnan.jdbcexamples;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** HighestSalariedEmployee {

**static** **final** String ***JDBC\_URL*** = "jdbc:mysql://localhost:3306/adjava";

**static** **final** String ***USERNAME*** = "root";

**static** **final** String ***PASSWORD*** = "root";

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

Connection connection = **null**;

Statement statement = **null**;

ResultSet resultSet = **null**;

**try** {

// establish connection.

connection = DriverManager.*getConnection*(***JDBC\_URL***, ***USERNAME***, ***PASSWORD***);

// create statement object.

statement = connection.createStatement();

String sqlQuery = "select\*from employees where esal=(select min(esal) from employees)";

// execute the sql query

resultSet = statement.executeQuery(sqlQuery);

System.***out***.println("ENO\tEname\tESAL\tEaddr");

System.***out***.println("=================================================");

**while** (resultSet.next()) {

System.***out***.println(resultSet.getInt(1) + "\t" + resultSet.getString(2) + "\t" + resultSet.getDouble(3)

+ "\t" + resultSet.getString(4));

}

} **catch** (SQLException e) {

e.printStackTrace();

} **finally** {

**if** (statement != **null**) {

**try** {

statement.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (resultSet != **null**) {

**try** {

resultSet.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if** (connection != **null**) {

**try** {

connection.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

}

**Application-15: average salary of employees**

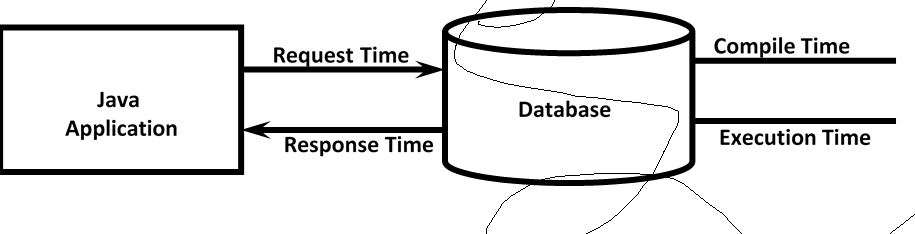
### **PreparedStatement (I)**

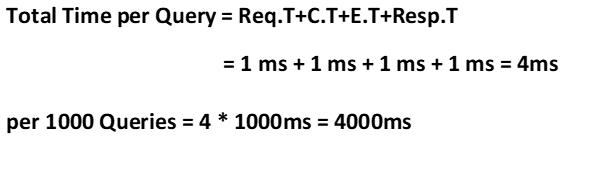
**Need of PreparedStatement:**

In the case of normal Statement, whenever we are executing SQL Query, every time compilation and execution will be happened at database side.

Statement st = con.createStatement();

ResultSet rs = st.executeQuery ("select \* from employees");





Sometimes in our application,we required to execute same query multiple times with same or different input values.

**Eg1:**

In IRCTC application,it is common requirement to list out all possible trains between 2 places

select \* from trains where source='XXX' and destination='YYY';

Query is same but source and destination places may be different. This query is required to execute lakhs of times per day.

**Eg2:**

In BookMyShow application, it is very common requirement to display theatre names where a particular movie running/playing in a particular city

select \* from theatres where city='XXX' and movie='YYY';

In this case this query is required to execute lakhs of times per day. May be with different movie names and different locations.

For the above requirements if we use Statement object, then the query is required to compile and execute every time, which creates performance problems.

To overcome this problem, we should go for PreparedStatement.

The main advantage of PreparedStatement is the query will be compiled only once even though we are executing multiple times, so that overall performance of the application will be improved.

We can create PreparedStatement by using prepareStatement() method of Connection interface. public PreparedStatement prepareStatement(String sqlQuery) throws SQLException

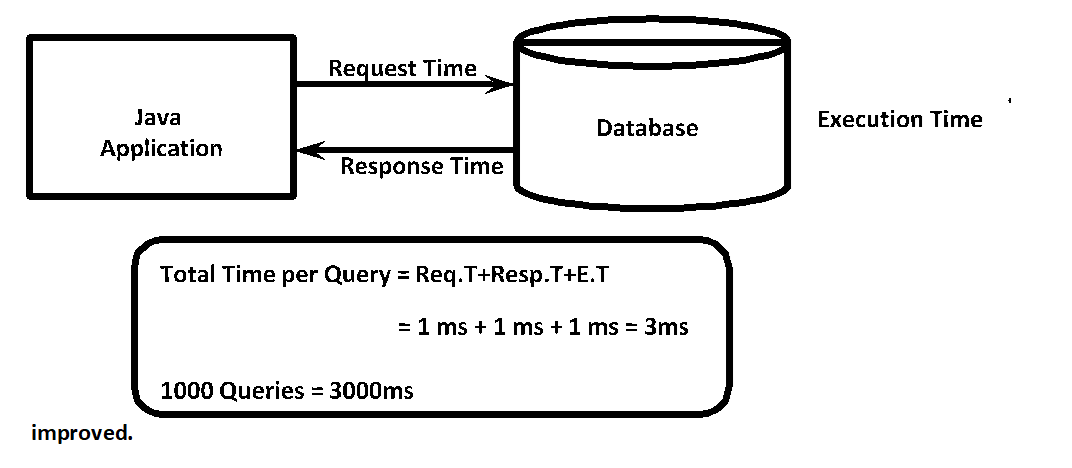
Eg: PreparedStatment pst=con.prepareStatement(sqlQuery);

At this line,sqlQuery will send to the database. Database engine will compile that query and stores in the database.

That pre compiled query will be returned to the java application in the form of PreparedStatement object.

Hence PreparedStatement represents "pre compiled sql query".

Whenever we call execute methods,database engine won't compile query once again and it will directly execute that query,so that overall performance will be improved



**Steps to develop JDBC Application by using PreparedStatement**

**1.** **Prepare SQLQuery either with parameters or without parameters.**

Eg: insert into employees values(100,'malli',1000,'hyd');

| **insert into employees values(?, ?, ?, ?);**    **Positional Parameter OR Place Holder OR IN Parameter** |
| --- |

**2.** **Create PreparedStatement object with our sql query.**

PreparedStatement pst = con.prepareStatement(sqlQuery); At this line only query will be compiled.

**3.** **If the query is parameterized query then we have to set input values to these parameters by using corresponding setter methods.**

We have to consider these positional parameters from left to right and these are 1 index based. i.e index of first positional parameter is 1 but not zero.

pst.setInt(1,100);

pst.setString(2,"malli");

pst.setDouble(3,1000);

pst.setString(4,"Hyd");

**Note:**

Before executing the query, for every positional parameter we have to provide input values otherwise we will get SQLException

**4.** **Execute SQL Query:**

PreparedStatement is the child interface of Statement and hence all methods of Statement interface are bydefault available to the PreparedStatement.Hence we can use same methods to execute sql query.

executeQuery()

executeUpdate()

execute()

**Note:**

We can execute same parameterized query multiple times with different sets of input values. In this case query will be compiled only once and we can execute multiple times.

**Note:**

We can use ? only in the place of input values and we cannot use in the place of sql keywords,table names and column names.

**Static Query vs Dynamic Query:**

The sql query without positional parameter(?) is called static query.

Eg: delete from employees where ename='malli'

The sql query with positional parameter(?) is called dynamic query.

Eg: select \* from employees where esal>?

**Application-16: Write a Jdbc Program to insert one\_record by using preparedStatement?**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.SQLException;**

**public class InsertRecordsByPreparedStatement {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**PreparedStatement preparedStatement = null;**

**Connection connection = null;**

**try {**

**// establish the connection.**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// Prepare sql query with parameters or without parameters.**

**String insertQuery = "insert into employee (eno,ename,esal,eaddr) values(?,?,?,?)";**

**// create preparedstatement object**

**preparedStatement = connection.prepareStatement(insertQuery);**

**// set the values to a positional p[arameters.**

**preparedStatement.setInt(1, 111);**

**preparedStatement.setString(2, "codegnan");**

**preparedStatement.setDouble(3, 60000);**

**preparedStatement.setString(4, "hyd");**

**// execute the query.**

**int rowsAffected = preparedStatement.executeUpdate();**

**System.*out*.println(rowsAffected + " rows Inserted Succesfully");**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (preparedStatement != null) {**

**try {**

**preparedStatement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-17: Write a Jdbc Program by using Dynamic input data and insert multiple Records?**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.SQLException;**

**import java.util.Scanner;**

**public class InsertRecordsByPreparedStatement {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**PreparedStatement preparedStatement = null;**

**Connection connection = null;**

**Scanner scanner;**

**try {**

**// establish the connection.**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// Prepare sql query with parameters or without parameters.**

**String insertQuery = "insert into employee (eno,ename,esal,eaddr) values(?,?,?,?)";**

**preparedStatement = connection.prepareStatement(insertQuery);**

**scanner = new Scanner(System.*in*);**

**System.*out*.println("Enetr the Number of Records to Insert : ");**

**int numOfRecords = scanner.nextInt();**

**for (int i = 0; i < numOfRecords; i++) {**

**System.*out*.println("Enter details for record " + (i + 1) + " : ");**

**System.*out*.println("Enter Employee Number : ");**

**int eno = scanner.nextInt();**

**System.*out*.println("Enter the Employee Name : ");**

**String ename = scanner.next();**

**System.*out*.println("Enter the employee salary : ");**

**double esal = scanner.nextDouble();**

**System.*out*.println("Enter employee address : ");**

**String eaddr = scanner.next();**

**preparedStatement.setInt(1, eno);**

**preparedStatement.setString(2, ename);**

**preparedStatement.setDouble(3, esal);**

**preparedStatement.setString(4, eaddr);**

**int rowsAffected = preparedStatement.executeUpdate();**

**System.*out*.println(rowsAffected + "row(s) Inserted succesfully");**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (preparedStatement != null) {**

**try {**

**preparedStatement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-18. Write a Jdbc Program by using dynamic input to read one record ?**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class ReadOneRecord {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER\_NAME = "com.mysql.cj.jdbc.Driver"; // MySQL driver name

public static void main(String[] args) {

// SQL query to select data from table

String selectQuery = "SELECT \* FROM employee WHERE eno = ?";

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

Scanner scanner = null;

try {

// Load the MySQL JDBC driver

Class.forName(DRIVER\_NAME);

// Establishing connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a prepared statement

preparedStatement = connection.prepareStatement(selectQuery);

scanner = new Scanner(System.in);

// Prompt user to enter employee number

System.out.print("Enter employee number to search: ");

int eno = scanner.nextInt();

// Set the parameter for the prepared statement

preparedStatement.setInt(1, eno);

// Execute the query

resultSet = preparedStatement.executeQuery();

// Check if record exists and print the result

if (resultSet.next()) {

int empNumber = resultSet.getInt("eno");

String empName = resultSet.getString("ename");

double empSalary = resultSet.getDouble("esal");

String empAddress = resultSet.getString("eaddr");

System.out.println("Employee details:");

System.out.println("Employee Number: " + empNumber);

System.out.println("Employee Name: " + empName);

System.out.println("Employee Salary: " + empSalary);

System.out.println("Employee Address: " + empAddress);

} else {

System.out.println("No record found for employee number: " + eno);

}

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

} finally {

// Close resources in finally block

if (resultSet != null) {

try {

resultSet.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (preparedStatement != null) {

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

} else {

System.out.println("Connection was not established.");

}

if (scanner != null) {

scanner.close();

}

}

}

}

**Application-19. Write a Jdbc Program by using dynamic input to read Multiple Records?**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class ReadMultipleRecords {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER\_NAME = "com.mysql.cj.jdbc.Driver"; // MySQL driver name

public static void main(String[] args) {

// SQL query to select data from table

String selectQuery = "SELECT \* FROM employee WHERE eno = ?";

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

Scanner scanner = null;

try {

// Load the MySQL JDBC driver

Class.forName(DRIVER\_NAME);

// Establishing connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a prepared statement

preparedStatement = connection.prepareStatement(selectQuery);

scanner = new Scanner(System.in);

// Prompt user for the number of records to read

System.out.print("Enter the number of records to read: ");

int numOfRecords = scanner.nextInt();

// Iterate to read multiple records

for (int i = 0; i < numOfRecords; i++) {

// Prompt user to enter employee number

System.out.print("Enter employee number " + (i + 1) + " to search: ");

int eno = scanner.nextInt();

// Set the parameter for the prepared statement

preparedStatement.setInt(1, eno);

// Execute the query

resultSet = preparedStatement.executeQuery();

// Check if record exists and print the result

if (resultSet.next()) {

int empNumber = resultSet.getInt("eno");

String empName = resultSet.getString("ename");

double empSalary = resultSet.getDouble("esal");

String empAddress = resultSet.getString("eaddr");

System.out.println("Employee details:");

System.out.println("Employee Number: " + empNumber);

System.out.println("Employee Name: " + empName);

System.out.println("Employee Salary: " + empSalary);

System.out.println("Employee Address: " + empAddress);

} else {

System.out.println("No record found for employee number: " + eno);

}

}

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

} finally {

// Close resources in finally block

if (resultSet != null) {

try {

resultSet.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (preparedStatement != null) {

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

} else {

System.out.println("Connection was not established.");

}

if (scanner != null) {

scanner.close();

}

}

}

}

**Application-20. Write a Jdbc Program to delete the Record by using prepared statement?**

package com.codegnan.jdbcprogrames;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Scanner;

public class DeleteRecord {

// JDBC URL, username, and password of MySQL server

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/Adjava?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";

private static final String USERNAME = "root";

private static final String PASSWORD = "root";

private static final String DRIVER\_NAME = "com.mysql.cj.jdbc.Driver"; // MySQL driver name

public static void main(String[] args) {

// SQL query to delete data from table

String deleteQuery = "DELETE FROM employee WHERE eno = ?";

Connection connection = null;

PreparedStatement preparedStatement = null;

Scanner scanner = null;

try {

// Load the MySQL JDBC driver

Class.forName(DRIVER\_NAME);

// Establishing connection

connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);

// Create a prepared statement

preparedStatement = connection.prepareStatement(deleteQuery);

scanner = new Scanner(System.in);

// Prompt user to enter employee number to delete

System.out.print("Enter employee number to delete: ");

int eno = scanner.nextInt();

// Set the parameter for the prepared statement

preparedStatement.setInt(1, eno);

// Execute the delete query

int rowsAffected = preparedStatement.executeUpdate();

// Check if record is deleted successfully

if (rowsAffected > 0) {

System.out.println(rowsAffected + " row(s) deleted successfully.");

} else {

System.out.println("No record found for employee number: " + eno);

}

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

} finally {

// Close resources in finally block

if (preparedStatement != null) {

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

} else {

System.out.println("Connection was not established.");

}

if (scanner != null) {

scanner.close();

}

}

}

}

**Advantages of PreparedStatement**:

**1.** **Performance will be improved when compared with simple Statement b'z query will be compiled only once.**

**2.** **Network traffic will be reduced between java application and database b'z we are not required to send query every time to the database.**

**3.** **We are not required to provide input values at the beginning and we can provide dynamically so that we can execute same query multiple times with different sets of values.**

**4.** **It allows to provide input values in java style and we are not required to convert into database specific format.**

**5.Best suitable to insert Date values**

**6.** **Best Sutitable to insert Large OBjects(CLOB,BLOB)**

**7.** **It prevents SQL Injection Attack.**

**Limitation of PreparedStatement:**

We can use PreparedStatement for only one sql query (Like CDMA Phone), but we can use simple Statement to work with any number of queries (Like GSM Phone).

Statement st = con.createStatement();

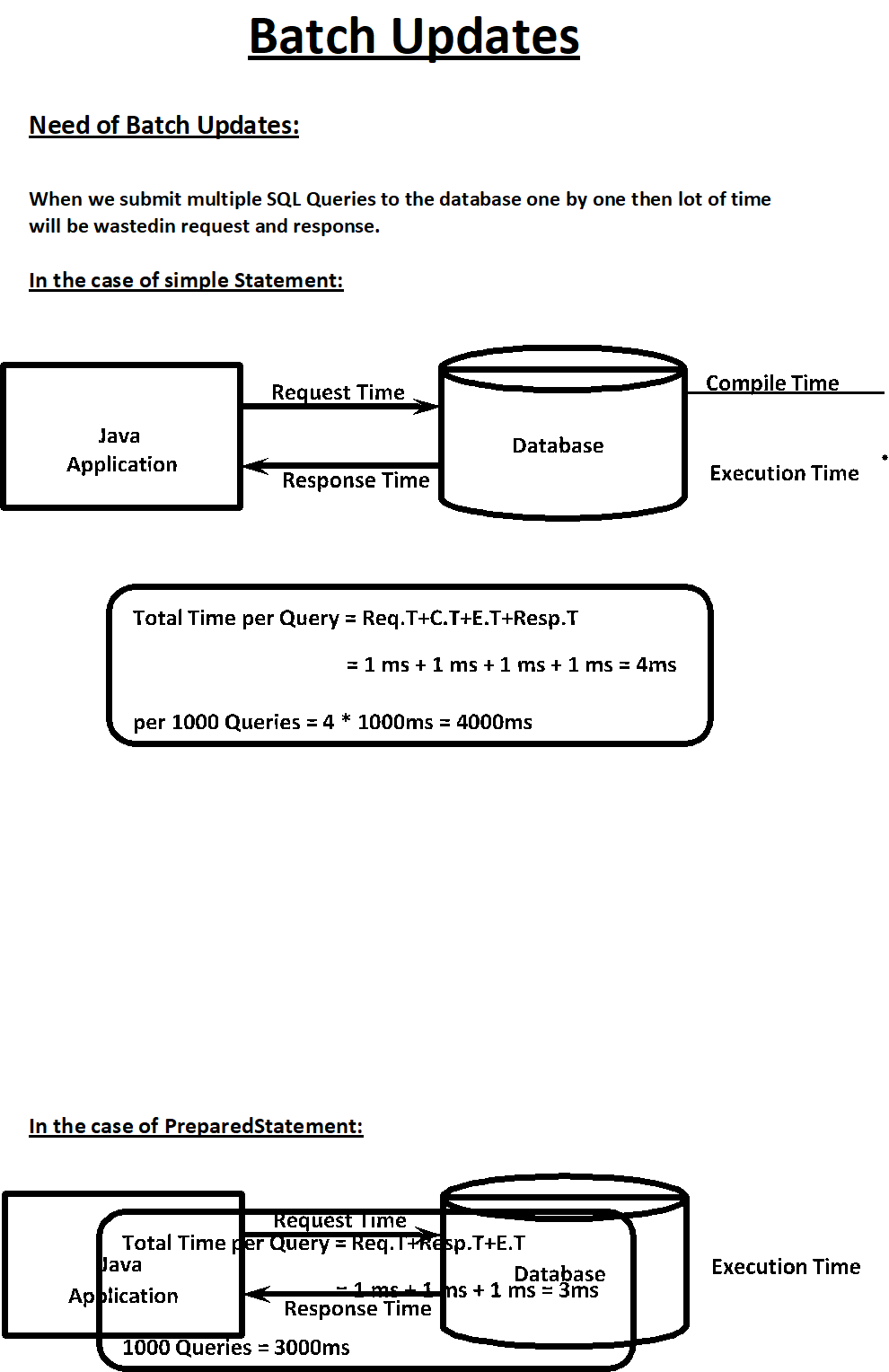
st.executeUpdate("insert into ...");

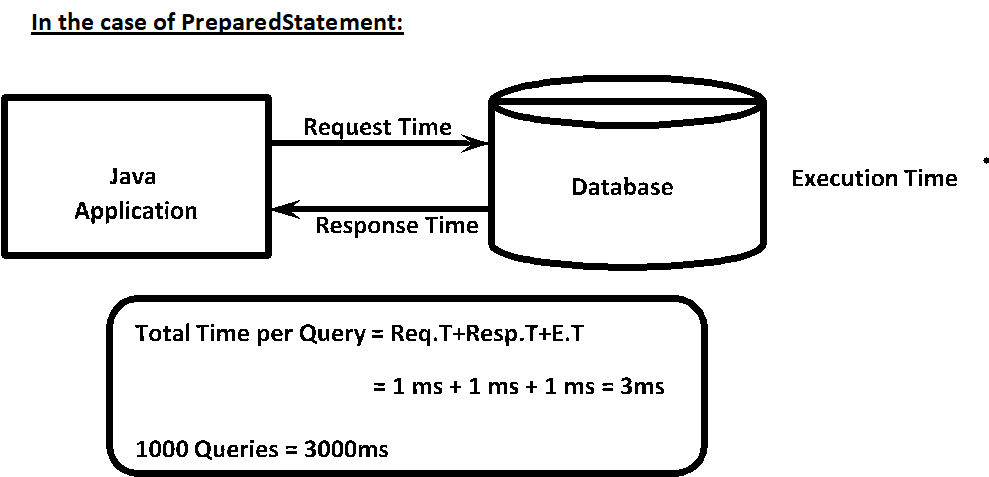
st.executeUpdate("update employees...");

**Differences Between Statement And PreparedStatement**

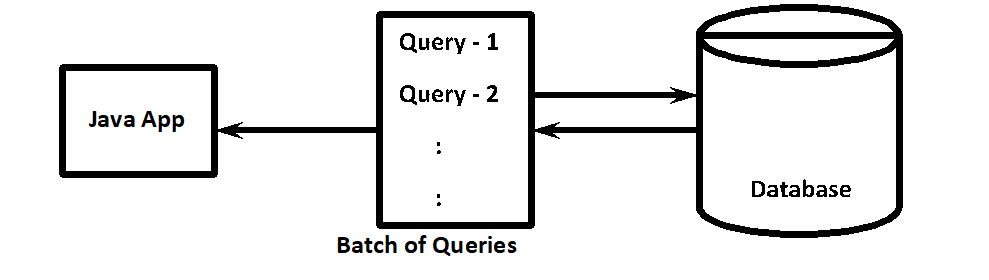
|  | **Statement** |  |  | **PreparedStatement** |  |
| --- | --- | --- | --- | --- | --- |
| **1) At the time of creating Statement Object, we are not required to provide any Query. Statement st = con.createStatement();**  **Hence Statement Object is not associated with any Query and we can use for multiple Queries.** | | | **1) At the time of creating PreparedStatement, we have to provide SQL Query compulsory and will send to the Database and will be compiled. PS pst = con.prepareStatement(query);**  **Hence PS is associated with only one Query.** | | |
| **2) Whenever we are using execute Method, every time Query will be compiled and executed.** | | | **2) Whenever we are using execute Method, Query won't be compiled just will be executed.** | | |
| **3) Statement Object can work only for Static Queries.** | | | **3) PS Object can work for both Static and Dynamic Queries.** | | |
| **4) Relatively Performance is Low.** | | | **4) Relatively Performance is High.** | | |
| **5) Best choice if we want to work with multiple Queries.** | | | **5) Best choice if we want to work with only one Query but required to execute multiple times.** | | |
| **6) There may be a chance of SQL Injection Attack.** | | | **6) There is no chance of SQL Injection Attack.** | | |
| **7) Inserting Date and Large Objects (CLOB and BLOB) is difficult.** | | | **7) Inserting Date and Large Objects (CLOB and BLOB) is easy.** | | |

**Batch Updates**

****

**In the above 2 cases , we are trying to submit 1000 queries to the database one by one. For submitting 1000 queries we need to communicate with the database 1000 times. It increases network traffic Between java application and database and even creates performance problems also.**

**To overcome these problems, we should go for Batch updates. We can group all related SQL Queries into a single batch and we can send that batch at a time to the database.**

****

**With Simple Statement Batch Updates:**

**Per 1000 Queries = Req.Time+1000\*C.T+1000\*E.T+Resp.Time**

**= 1ms+1000\*1ms+1000\*1ms+1ms**

**= 2002ms**

**With PreparedStatement Batch Updates:**

**Per 1000 Queries = Req.Time+1000\*E.T+Resp.Time**

**= 1ms+1000\*1ms+1ms**

**= 1002ms**

**Hence the main advantages of Batch updates are**

**1.** **We can reduce network traffic**

**2.** **We can improve performance.**

**We can implement batch updates by using the following two methods**

**1.** **public void addBatch(String sqlQuery) To add query to batch**

**2.** **int[] executeBatch()**

**to execute a batch of sql queries**

**Application-21:We can implement batch updates either by simple Statement or by PreparedStatement**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class BatchUpdateExample {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**try {**

**// establish connection**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// create statement**

**statement = connection.createStatement();**

**// prepare the sql query;**

**String insertQuery = "insert into employee values(555,'malli',60000,'hyd')";**

**String updateQuery = "update employee set esal=esal+1000 where esal<50000";**

**String deleteQuery = "delete from employee where esal>=60000";**

**statement.addBatch(insertQuery);**

**statement.addBatch(updateQuery);**

**statement.addBatch(deleteQuery);**

**int[] updateCounts = statement.executeBatch();**

**int totalUpdateCount = 0;**

**for (int count : updateCounts) {**

**totalUpdateCount = totalUpdateCount + count;**

**}**

**System.*out*.println("The Number of Rows Updated : " + totalUpdateCount);**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-22:- Program to Demonstrate Batch Updates with PreparedStatement**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.SQLException;**

**public class BatchUpdateExample {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**PreparedStatement preparedStatement = null;**

**try {**

**// establish connection**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**String sql = "update employee set esal=? where eno=?";**

**preparedStatement = connection.prepareStatement(sql);**

**preparedStatement.setDouble(1, 72000);**

**preparedStatement.setInt(2, 5);**

**preparedStatement.addBatch();**

**preparedStatement.setDouble(1, 6000);**

**preparedStatement.setInt(2, 4);**

**preparedStatement.addBatch();**

**preparedStatement.setDouble(1, 12000);**

**preparedStatement.setInt(2, 6);**

**preparedStatement.addBatch();**

**int[] updateCounts = preparedStatement.executeBatch();**

**System.*out*.println("batch update is executed ");**

**for (int i = 0; i < updateCounts.length; i++) {**

**System.*out*.println("update count for statement " + i + " : " + updateCounts[i]);**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (preparedStatement != null) {**

**try {**

**preparedStatement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**ResultSet Types**

**Division-1:**

**Based on operations performed on ResultSet, we can divide ResultSet into 2 types**

**1.** **Read Only ResultSets (Static ResultSets)**

**2.** **Updatable ResultSets (Dynamic ResultSets)**

**1.** **Read Only ResultSets:**

**We can perform only read operations on the ResultSet by using corresponding getter methods and we cannot perform any updations.**

**By default ResultSet is Read Only.**

**We can specify explicitly ResultSet as Read only by using the following constant of ResultSet.**

**public static final int CONCUR\_READ\_ONLY ➔ 1007**

**2.** **Updatable ResultSets:**

**The ResultSet which allows programmer to perform updations, such type of ResultSets are called Updatable ResultSets.**

**In this case we can perform select, insert, delete and update operations.**

**We can specify ResultSet explicitly as Updatable by using the following constant of ResultSet.**

**public static final int CONCUR\_UPDATABLE ➔ 1008**

**Division-2:**

**Based on Cursor movement, ResultSets will be divided into the following 2 types.**

**1.** **Forward only (Non-Scrollable) ResultSet**

**2.** **Scrollable ResultSets**

**1.** **Forward Only ResultSets:**

**It allows the programmers to iterate records only in forward direction ie from top to bottom sequentially.**

**By default every ResultSet is forward only.**

**We can specify explicitly ResultSet as Forward only by using the following constant of ResultSet**

**public static final int TYPE\_FORWARD\_ONLY ➔ 1003**

**2.** **Scrollable ResultSets:**

**It allows the programmers to iterate in both forward and backward directions.**

**We can also jump to a particular position randomly, or relative to current position. Here we can move to anywhere.**

**There are two types of Scrollable ResultSets.**

**1.** **Scroll Insensitive ResultSet**

**2.** **Scroll Sensitive ResultSet**

**1.** **Scroll Insensitive ResultSet:**

**After getting ResultSet if we are performing any change in Database and if those changes are not reflecting to the ResultSet, such type of ResultSets are called scroll insensitive ResultSets.**

**i.e ResultSet is insensitive to database operations.**

**We can specify explicitly ResultSet as Scroll insensitive by using the following constant**

**public static final int TYPE\_SCROLL\_INSENSITIVE ➔ 1004**

**2.Scroll sensitive ResultSets:**

**After getting the ResultSet if we perform any change in the database and if those changes are visible to ResultSet, such type of ResultSet is called Scroll sensitive ResultSet.**

**i.e ResultSet is sensitive to database operations**

**We can specify explicitly ResultSet as scroll sensitive by using the following constant..**

**public static final int TYPE\_SCROLL\_SENSITIVE ➔ 1005**

**Differnences Between Scroll Insensitive And Scroll Sensitive ResultSets**

| **Scroll Insensitive** | **Scroll Sensitive** |
| --- | --- |
| **After getting ResultSet if we perform any updation in the DB then those updation are not visible to the ResultSet i.e. ResultSet is**  **insensitive to DB updations.** | **After getting ResultSet if we perform any updation in the DB then those updation are by default available to the to the ResultSet**  **i.e. ResultSet is sensitive to DB updations.** |
| **As insensitive ResultSet is like snapshot of Data in DB when Query will be executed.** | **A Sensitive ResultSet doesn't represent snap shot of Data. It contains Pointers to**  **Rows of DB directly, which satisfy Query Condition.** |
| **Relatively Performance is High.** | **Relatively Performance is low because for**  **get Operation a Trip is required to DB.** |

**Differnences between Forward only and Scrollable ResultSets**

| **Non Scrollable (Forward only)** | **Scrollable** |
| --- | --- |
| **Cursor can move only in Forward Direction.** | **Cursor can move in both Forward and Backward Direction.** |
| **This Cursor can't move randomly.** | **Scrollable ResultSet Cursor can move randomly.** |
| **By using Non Scrollable ResultSet Cursor if we want to move Nth Record (N + 1) Iterations are required.** | **Performance is high because Cursor can move randomly to any Record.** |

**How to get Required ResultSet:**

**We can create Statement objects as follows to get desired ResultSets.**

| **Statement st =con.createStatement(int type,int mode); PreparedStatement pst=con.prepareStatement(query,int type,int mode);** |
| --- |

**Allowed values for type are: ResultSet.TYPE\_FORWARD\_ONLY ➔ 1003**

**ResultSet.TYPE\_SCROLL\_INSENSITIVE ➔ 1004**

**ResultSet.TYPE\_SCROLL\_SENSITIVE ➔ 1005**

**Allowed values for mode are: ResultSet.CONCUR\_READ\_ONLY ➔ 1007**

**ResultSet.CONCUR\_UPDATABLE ➔ 1008**

**List of allowed methods on Non-Scrollable ResultSets(Forward only):**

**1.** **rs.next()**

**it checks whether next record is available. If it is available then cursor will move to that position**

**2.** **rs.getXxx()**

**Read column values from record either with column index or with column name**

**3.** **rs.getRow()**

**It returns current position of cursor in the ResultSet i.e row number**

**List of allowed methods on Scrollable ResultSets:**

**1.** **rs.next()**

**2.** **rs.getXxx()**

**3.** **rs.getRow()**

**4.** **rs.previous()**

**It checks whether previous record is available or not. If it is available then the cursor will move to that record position**

**5.** **rs.beforeFirst();**

**the cursor will be moved to before first record position**

**6.** **rs.afterLast()**

**moves the cursor after last record position**

**7.** **rs.first()**

**moves the cursor to the first record position**

**8.** **rs.last()**

**moves the cursor to the last record position**

**9.** **rs.absolute(int x) The argument can be either positive or negative.**

**If it is positive then the cursor will be moved to that record position from top of ResultSet. If the argument is negative then it will be moved to the specified record position from last.**

**10.** **rs.relative(int x) The argument can be either positive or negative**

**If the argument is positive then the cursor will move to forward direction to specified number of records from the current position. If the argument is negative then the cursor will move to backward direction to the specified number of records from the current position.**

**11.** **rs.isFirst() returns true if the cursor is locating at first record position**

**12.** **rs.isLast()**

**13.** **rs.isBeforeFirst()**

**14.** **rs.isAfterLast()**

**15.** **rs.refreshRow() We can use this method in scroll sensitive ResultSets to update row with latest values from**

**Database.**

**Q. What is the difference Between absolute() and relative()methods?**

**absolute() method will always work either from BFR or from ALR. relative() method will work wrt to current position**

**In both methods +ve number means we have to move forward direction and -ve number means we have to move backward direction.**

**Note:**

**1.** **rs.last() and rs.absolute(-1) both are equal**

**2.** **rs.first() and rs.absolute(1) both are equal**

**Application-1: Iterating records in both forward and backward direction by using SCROLLABLE ResultSet**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class ResultSetDemo {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**ResultSet resultSet = null;**

**try {**

**// establish connection**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// CREATE THE STATEMENT Object**

**statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);**

**// prepare SQLQuery.**

**String sqlQuery = "select\*from employee";**

**// Process results from resultSet**

**resultSet = statement.executeQuery(sqlQuery);**

**System.*out*.println("Records in Forward Direction : ");**

**System.*out*.println();**

**System.*out*.println("ENO\tENAME\teSAL\tEaddr");**

**System.*out*.println("=========================================================");**

**while (resultSet.next()) {**

**System.*out*.println(resultSet.getInt(1) + "\t" + resultSet.getString(2) + "\t" + resultSet.getDouble(3)**

**+ "\t" + resultSet.getString(4));**

**}**

**System.*out*.println("Records in Backward Direction : ");**

**System.*out*.println();**

**System.*out*.println("ENO\tENAME\teSAL\tEaddr");**

**System.*out*.println("=========================================================");**

**while (resultSet.previous()) {**

**System.*out*.println(resultSet.getInt(1) + "\t" + resultSet.getString(2) + "\t" + resultSet.getDouble(3)**

**+ "\t" + resultSet.getString(4));**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (resultSet != null) {**

**try {**

**resultSet.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Application-2: Navigating Records by using SCROLLABLE ResultSet**

**package com.codegnan.jdbcexamples;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**public class ResultSetDemo {**

**static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/adjava";**

**static final String *USERNAME* = "root";**

**static final String *PASSWORD* = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**ResultSet resultSet = null;**

**try {**

**// establish connection**

**connection = DriverManager.*getConnection*(*JDBC\_URL*, *USERNAME*, *PASSWORD*);**

**// CREATE THE STATEMENT Object**

**statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);**

**// prepare SQLQuery.**

**String sqlQuery = "select\*from employee";**

**// Process results from resultSet**

**resultSet = statement.executeQuery(sqlQuery);**

**System.*out*.println("Records in Forward Direction : ");**

**System.*out*.println();**

**System.*out*.println("ENO\tENAME\teSAL\tEaddr");**

**System.*out*.println("=========================================================");**

**while (resultSet.next()) {**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**}**

**System.*out*.println("Records in Backward Direction : ");**

**System.*out*.println();**

**System.*out*.println("ENO\tENAME\teSAL\tEaddr");**

**System.*out*.println("=========================================================");**

**while (resultSet.previous()) {**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**}**

**resultSet.first();**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**resultSet.last();**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**resultSet.relative(-4);**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**resultSet.absolute(2);**

**System.*out*.println(resultSet.getRow() + "------------>" + resultSet.getInt(1) + "\t"**

**+ resultSet.getString(2) + "\t" + resultSet.getDouble(3) + "\t" + resultSet.getString(4));**

**System.*out*.println(resultSet.isFirst());**

**System.*out*.println(resultSet.isLast());**

**System.*out*.println(resultSet.isBeforeFirst());**

**System.*out*.println(resultSet.isAfterLast());**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (resultSet != null) {**

**try {**

**resultSet.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Transaction Management in JDBC**

**Process of combining all related operations into a single unit and executing on the rule "either all or none", is called transaction management.**

**Hence transaction is a single unit of work and it will work on the rule "either all or none".**

**Case-1: Funds Transfer**

**1.** **debit funds from sender's account**

**2.** **credit funds into receiver's account**

**All operations should be performed as a single unit only. If debit from sender's account completed and credit into receiver's account fails then there may be a chance of data inconsistency problems.**

**Types of Transactions:**

**There are two types of Transactions**

**1.** **Local Transactions**

**2.** **Global Transactions**

**1. Local Transactions:**

**All operations in a transaction are executed over same database.**

**Eg: Funds transfer from one accoun to another account where both accounts in the same bank.**

**2. Global Transactions:**

**All operations is a transaction are expected over different databases.**

**Eg: Funds Transfer from one account to another account and accounts are related to different banks.**

**Note:**

**JDBC can provide support only for local transactions.**

**If we want global transactions then we have to go for EJB or Spring framework.**

**Process of Transaction Management in JDBC:**

**1.** **Disable auto commit mode of JDBC**

**By default auto commit mode is enabled. i.e after executing every sql query, the changes will be committed automatically in the database.**

**We can disable auto commit mode as follows**

**con.setAutoCommit(false);**

**2.** **If all operations completed then we can commit the transaction by using the following method.**

**con.commit();**

**3.** **If any sql query fails then we have to rollback operations which are already completed by using rollback() method.**

**con.rollback();**

**Program: To demonstrate Transactions**

**package com.codegnan.jdbctransactions;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**import java.util.Scanner;**

**public class TransactionDemo {**

**static final String JDBC\_URL = "jdbc:mysql://localhost:3306/adjava";**

**static final String USERNAME = "root";**

**static final String PASSWORD = "root";**

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

**Connection connection = null;**

**Statement statement = null;**

**ResultSet resultSet = null;**

**try {**

**// establish connection**

**connection = DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD);**

**// create the statement object**

**statement = connection.createStatement();**

**// display data before transaction.**

**System.out.println("DATA before Transaction ");**

**ResultSet rsBefore = statement.executeQuery("select\*from accounts");**

**while (rsBefore.next()) {**

**System.out.println(rsBefore.getString(1) + "\t" + rsBefore.getDouble(2));**

**}**

**// start transaction**

**System.out.println("\n Transaction Begins ");**

**// disable autocommit mode**

**connection.setAutoCommit(false);**

**statement.executeUpdate("update accounts set balance=balance-10000 where name='malli'");**

**statement.executeUpdate("update accounts set balance=balance+10000 where name='sunny'");**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("can you please confirm this transaction of 10000[Yes/No]");**

**// confirm the transaction**

**String option = scanner.next();**

**if (option.equalsIgnoreCase("yes")) {**

**connection.commit();**

**System.out.println("Transaction Committed");**

**} else {**

**connection.rollback();**

**System.out.println("transaction rollbacked");**

**}**

**System.out.println("Data After Transaction ");**

**ResultSet rsAfter = statement.executeQuery("select\*from accounts ");**

**while (rsAfter.next()) {**

**System.out.println(rsAfter.getString(1) + "\t" + rsAfter.getDouble(2));**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**} finally {**

**if (statement != null) {**

**try {**

**statement.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (resultSet != null) {**

**try {**

**resultSet.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**if (connection != null) {**

**try {**

**connection.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**}**

**Jdbc curd operations by using switch statemenmt and DAO classes and pojo classes?**

**------------------------------------------------------------------------------------**

**CREATE TABLE users (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**email VARCHAR(100) NOT NULL**

**);**

**INSERT INTO users (name, email) VALUES ('John Doe', 'john.doe@example.com');**

**INSERT INTO users (name, email) VALUES ('Jane Smith', 'jane.smith@example.com');**

**INSERT INTO uses (name, email) VALUES ('Alice Johnson', 'alice.johnson@example.com');**

**User.java**

**---------------**

**package com.codegnan.entity;**

**public class User {**

**private int id;**

**private String name;**

**private String email;**

**// Constructors**

**public User() {**

**}**

**public User(int id, String name, String email) {**

**this.id = id;**

**this.name = name;**

**this.email = email;**

**}**

**// Getters and Setters**

**public int getId() {**

**return id;**

**}**

**public void setId(int id) {**

**this.id = id;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getEmail() {**

**return email;**

**}**

**public void setEmail(String email) {**

**this.email = email;**

**}**

**// toString() method**

**@Override**

**public String toString() {**

**return "User{" +**

**"id=" + id +**

**", name='" + name + '\'' +**

**", email='" + email + '\'' +**

**'}';**

**}**

**}**

**UserDAO.java**

**----------------------**

**package com.codegnan.DAO;**

**import java.util.List;**

**import com.codegnan.entity.User;**

**public interface UserDAO {**

**void addUser(User user);**

**User getUserById(int id);**

**List<User> getAllUsers();**

**void updateUser(User user);**

**void deleteUser(int id);**

**}**

**JDBCUserDAO.java**

**------------------------------**

**package com.codegnan.DAO;**

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

**import java.util.ArrayList;**

**import java.util.List;**

**import com.codegnan.entity.User;**

**public class JDBCUserDAO implements UserDAO {**

**private String url;**

**private String username;**

**private String password;**

**public JDBCUserDAO(String url, String username, String password) {**

**this.url = url;**

**this.username = username;**

**this.password = password;**

**}**

**@Override**

**public void addUser(User user) {**

**try (Connection connection = DriverManager.getConnection(url, username, password)) {**

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

**try (PreparedStatement statement = connection.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {**

**statement.setString(1, user.getName());**

**statement.setString(2, user.getEmail());**

**statement.executeUpdate();**

**ResultSet resultSet = statement.getGeneratedKeys();**

**if (resultSet.next()) {**

**user.setId(resultSet.getInt(1));**

**}**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**@Override**

**public User getUserById(int id) {**

**User user = null;**

**try (Connection connection = DriverManager.getConnection(url, username, password)) {**

**String query = "SELECT \* FROM users WHERE id=?";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**statement.setInt(1, id);**

**ResultSet resultSet = statement.executeQuery();**

**if (resultSet.next()) {**

**user = new User();**

**user.setId(resultSet.getInt("id"));**

**user.setName(resultSet.getString("name"));**

**user.setEmail(resultSet.getString("email"));**

**}**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return user;**

**}**

**@Override**

**public List<User> getAllUsers() {**

**List<User> users = new ArrayList<>();**

**try (Connection connection = DriverManager.getConnection(url, username, password)) {**

**String query = "SELECT \* FROM users";**

**try (Statement statement = connection.createStatement();**

**ResultSet resultSet = statement.executeQuery(query)) {**

**while (resultSet.next()) {**

**User user = new User();**

**user.setId(resultSet.getInt("id"));**

**user.setName(resultSet.getString("name"));**

**user.setEmail(resultSet.getString("email"));**

**users.add(user);**

**}**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return users;**

**}**

**@Override**

**public void updateUser(User user) {**

**try (Connection connection = DriverManager.getConnection(url, username, password)) {**

**String query = "UPDATE users SET name=?, email=? WHERE id=?";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**statement.setString(1, user.getName());**

**statement.setString(2, user.getEmail());**

**statement.setInt(3, user.getId());**

**statement.executeUpdate();**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**@Override**

**public void deleteUser(int id) {**

**try (Connection connection = DriverManager.getConnection(url, username, password)) {**

**String query = "DELETE FROM users WHERE id=?";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**statement.setInt(1, id);**

**statement.executeUpdate();**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**Main.java**

**-------------------**

**package com.codegnan.DAO;**

**import java.util.List;**

**import java.util.Scanner;**

**import com.codegnan.entity.User;**

**public class Main {**

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

**String url = "jdbc:mysql://localhost:3306/advjava..,.,";**

**String username = "root";**

**String password = "root";**

**Scanner scanner = new Scanner(System.in);**

**UserDAO userDAO = new JDBCUserDAO(url, username, password);**

**while (true) {**

**System.out.println("Choose operation:");**

**System.out.println("1. Add User");**

**System.out.println("2. View User by ID");**

**System.out.println("3. View All Users");**

**System.out.println("4. Update User");**

**System.out.println("5. Delete User");**

**System.out.println("6. Exit");**

**int choice = scanner.nextInt();**

**switch (choice) {**

**case 1:**

**addUser(userDAO);**

**break;**

**case 2:**

**viewUserById(userDAO);**

**break;**

**case 3:**

**viewAllUsers(userDAO);**

**break;**

**case 4:**

**updateUser(userDAO);**

**break;**

**case 5:**

**deleteUser(userDAO);**

**break;**

**case 6:**

**scanner.close();**

**return;**

**default:**

**System.out.println("Invalid choice!");**

**}**

**}**

**}**

**private static void addUser(UserDAO userDAO) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter name:");**

**String name = scanner.nextLine();**

**System.out.println("Enter email:");**

**String email = scanner.nextLine();**

**User user = new User();**

**user.setName(name);**

**user.setEmail(email);**

**userDAO.addUser(user);**

**System.out.println("User added successfully!");**

**}**

**private static void viewUserById(UserDAO userDAO) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter user ID:");**

**int id = scanner.nextInt();**

**User user = userDAO.getUserById(id);**

**if (user != null) {**

**System.out.println("User found:");**

**System.out.println("ID: " + user.getId());**

**System.out.println("Name: " + user.getName());**

**System.out.println("Email: " + user.getEmail());**

**} else {**

**System.out.println("User not found!");**

**}**

**}**

**private static void viewAllUsers(UserDAO userDAO) {**

**List<User> users = userDAO.getAllUsers();**

**System.out.println("All users:");**

**for (User user : users) {**

**System.out.println("ID: " + user.getId() + ", Name: " + user.getName() + ", Email: " + user.getEmail());**

**}**

**}**

**private static void updateUser(UserDAO userDAO) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter user ID:");**

**int id = scanner.nextInt();**

**User existingUser = userDAO.getUserById(id);**

**if (existingUser != null) {**

**System.out.println("Enter new name:");**

**String name = scanner.next();**

**System.out.println("Enter new email:");**

**String email = scanner.next();**

**existingUser.setName(name);**

**existingUser.setEmail(email);**

**userDAO.updateUser(existingUser);**

**System.out.println("User updated successfully!");**

**} else {**

**System.out.println("User not found!");**

**}**

**}**

**private static void deleteUser(UserDAO userDAO) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter user ID:");**

**int id = scanner.nextInt();**

**User existingUser = userDAO.getUserById(id);**

**if (existingUser != null) {**

**userDAO.deleteUser(id);**

**System.out.println("User deleted successfully!");**

**} else {**

**System.out.println("User not found!");**

**}**

**}**

**}**