**Please attend JDBC remaining topics from today[20-03-2023] at 9:PM in the workshop link.**

**Workshop Link:** [**https://attendee.gotowebinar.com/register/2348474801872643416**](https://attendee.gotowebinar.com/register/2348474801872643416)

Adv Java + Java Frameworks

—--------------------------------

Adv Java

Hibernate

Spring

Spring Boot

Microservices

Webservices

7:00AM to 8:30AM              8:30AM to 10:00AM

Adv JAVA

—--

—--                                          Spring Framework

—- Spring Boot

—-- Microservices

Hibernate

Webservices

Adv Java:

—----------

1. Pre-Requisite
2. JDBC
3. Servlets
4. JSPs
5. JSTL
6. EL

Prerequisite:

1. Class, abstract class, interface
2. Static Keyword
3. Class.forName() and newInstance()
4. Exception Handling

1. throws keyword

2. Try-catch-finally

3. try-with-resources

Containers In Java:

—-----------------------

Container is a top level component in Java, it is able to manage other programming elements like  variables, methods, blocks, constructors,....

There are three types of Containers in Java.

1. Class
2. Abstract class
3. Interface

Class:

—------

To represent all real world entities in java applications we have to use classes.

EX: Student, Employee, Account, Product,.....

Syntax:

[accessModifiers] class ClassName [extends SuperClassName] [implements InterfaceList]{

—-----

—-----

Variables

Constructors

Blocks

Methods

Classes

Abstract classes

Interfaces

Enums

—--

—--

}

Procedure to write classes in Java:

—-------------------------------------------

1. Declare a class with the “class” keyword.
2. Define variables and methods inside the class as per the requirement.
3. Declare the main class with main() method.
4. Inside the main class, inside the main() method , create an object for a user defined class.
5. Access class members.

EX:

—--

Employee.java

package com.durgasoft.app01.entities;

public class Employee {

int eno;

String ename;

float esal;

String eaddr;

public Employee(int eno, String ename, float esal, String eaddr) {// User defined constructor

this.eno = eno;

this.ename = ename;

this.esal = esal;

this.eaddr = eaddr;

}

public void displayEmpDetails() {

System.out.println("Employee Details");

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

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

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

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

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

}

}

Test.java

package com.durgasoft.app01.test;

import com.durgasoft.app01.entities.Employee;

public class Test {

public static void main(String[] args) {

Employee employee1 = new Employee(111, "AAA", 5000, "Hyd");

employee1.displayEmpDetails();

System.out.println();

Employee employee2 = new Employee(222, "BBB", 6000, "Hyd");

employee2.displayEmpDetails();

System.out.println();

Employee employee3 = new Employee(333, "CCC", 7000, "Hyd");

employee3.displayEmpDetails();

}

}

There are two types methods in java.

1. Concrete Methods
2. Abstract Methods

Create methods will have both method definition and method implementation.

void m1() —-> Method declaration/Method definition/Method prototype

{  —-----> Method Body / Method Implementation

}

Abstract method will have only method declaration  without method body.

abstract void m1();

Abstract class:

—-----------------

It is one type of class, it allows zero or more no of  concrete methods and zero or more no of abstract methods.

For abstract classes, we are able to declare reference variables, but, we are unable to create Objects.

Procedure to use abstract classes in Java applications:

—---------------------------------------------------------------------

1. Declare an abstract class with abstract keyword.
2. Provide variables and methods[concrete and abstract] as per the requirement.
3. Declare a subclass for the abstract class and provide implementation for all abstract methods.
4. In the main class, in mian() method , create an object for the subclass and declare a reference variable either for abstract class or for sub class.
5. Access abstract class members.

EX:

—--

package com.durgasoft.app02.test;

abstract class A{

void m1() {

System.out.println("m1-A");

}

abstract void m2();

abstract void m3();

}

class B extends A{

void m2() {

System.out.println("m2-B");

}

void m3() {

System.out.println("m3-B");

}

void m4() {

System.out.println("m4-B");

}

}

public class Test {

public static void main(String[] args) {

//A a = new A(); --> Error

A a = new B();

a.m1();

a.m2();

a.m3();

//a.m4(); ---> Error

System.out.println();

B b = new B();

b.m1();

b.m2();

b.m3();

b.m4();

}

}

OP:

m1-A

m2-B

m3-B

m1-A

m2-B

m3-B

m4-B

Interfaces:

—------------

1. Interface is a Java Feature, it allows only abstract methods.
2. For interfaces we are able to declare only reference variables, it is not possible to create Objects.
3. Inside the interfaces, by default, all variables are “public static final”, no need to declare explicitly.
4. Inside the interfaces, by default, all methods are “public abstract”, no need to declare explicitly.

Steps to use Interfaces in Java applications:

1. Declare an interface with the “interface” keyword.
2. Declare variables and methods inside the interface.
3. Declare an implementation class [Same as subclass but not sub class] for interface.
4. Provide implementation for all abstract methods of the interface.
5. In the Main class, in the main() method, create an object for implementation class and declare reference variables either for interface or for implementation class.
6. Access interface members.

EX:

package com.durgasoft.app03.test;

interface I{

int x = 10;// public static final

void m1();// public abstract

void m2();// public abstract

void m3();// public abstract

}

class A implements I{

public void m1() {

System.out.println("m1-A");

}

public void m2() {

System.out.println("m2-A");

}

public void m3() {

System.out.println("m3-A");

}

public void m4() {

System.out.println("m4-A");

}

}

public class Test {

public static void main(String[] args) {

//I i = new I();--> Error

I i = new A();

i.m1();

i.m2();

i.m3();

//i.m4(); --> Error

System.out.println();

A a = new A();

a.m1();

a.m2();

a.m3();

a.m4();

System.out.println();

System.out.println(I.x);

System.out.println(A.x);

System.out.println(i.x);

System.out.println(a.x);

}

}

Q)What are the differences between classes, abstract classes and interfaces?

—----------------------------------------------------------------------------------------------

Ans:

—---

1. Classes are able to allow only concrete methods.

Abstract classes are able to allow both concrete methods and abstract methods.

Interfaces are able to allow only abstract methods.

1. To declare classes, a “class” keyword is required.

To declare abstract classes we have to use the “abstract” keyword along with class keyword.

To declare interfaces we need the “interface” keyword.

1. For classes only we are able to create both reference variables and objects.

For abstract classes and interfaces we are able to create only reference variables, we are unable to create objects.

1. In interfaces, by default all variables are public static final.

No default cases exist for the variables in classes and abstract classes.

1. In interfaces, by default all methods are public and abstract , no default cases exist for methods in classes and abstract classes.
2. Classes are able to provide less shareability.

Abstract classes are able to provide middle level shareability.

Interfaces are able to provide more shareability.

1. Classes and abstract classes are able to allow only one superclass, not allowing more than one superclass.

 Interfaces are able to allow one or more super interfaces.

1. Constructors are possible in classes and abstract classes.

Constructors are not possible in interfaces.

1. Static blocks and instance blocks  are possible in classes and abstract classes.

Static blocks and instance blocks are not possible in interfaces.

Static Keyword:

—------------------  
In Java applications “static” is a keyword, it is able to improve Shareability.

In Java applications, we are able to utilize static keyword in the following four ways.

1. Static variables
2. Static Methods
3. Static Blocks
4. Static Import

Static variables:

—----------------------

1. Static variables are normal java variables, these variables are recognized and initialized at the time of loading the respective class bytecode.
2. In Java , static variables are able to share their latest values [last modified values] to the future objects which we are going to create and to the previous objects which we have created already.
3. In Java applications , always static variables must be declared as class level variables, not as local variables, if we declare static variables as local variables then the compiler will raise an error.
4. In Java, static variables data will be stored in Method Area.
5. In Java applications, we are able to access static variables either by using class names directly or by using reference variables of the respective class, where it is always recommended to use class name to access static variables.

Note: In Java , if we access any instance variable by using a reference variable contains null value then JVM will raise an exception like java.lang.NullPointerException, but if we access any static variable by using a reference variable contains null value then JVM will not raise any exception rather JVM access static variable value.

EX:

package com.durgasoft.app04.test;

class A{

static int i = m1();

int j = m2();

A(){

System.out.println("Object Creating.....");

}

static int m1() {

System.out.println("m1-A");

return 10;

}

int m2() {

//static int k = 30; ---> Error

System.out.println("m2-A");

return 20;

}

}

public class Test {

public static void main(String[] args)throws Exception {

A a = new A();

//Class.forName("com.durgasoft.app04.test.A");

System.out.println(a.i);

System.out.println(a.j);

System.out.println(A.i);

//System.out.println(A.j);--> Error

A a1 = null;

//System.out.println(a1.j);---> java.lang.NullPointerException

System.out.println(a1.i);

}

}

A diagram of a method area

Description automatically generated

Static Methods:

—-----------------

Static method is a java method , it will be recognized and executed at  the moment when we access that method.

Static methods are able to allow only static members of the current class, static methods are not allowing instance members of the current class.

Note: Inside the static method, if we want to use instance members then we must create an object for the respective class and we must use the generated reference variable.

In Java applications, we are able to access static methods either by using class names directly or by using reference variables of the respective class .

Note: In Java applications, if we access any instance method by using a reference variable contains null value there JVM will raise an exception like java.lang.NullPointerException, but if we access static method by using a reference variable contains null value then JVM will not raise any exception, here JVM will execute the respective static method.

Static methods are not allowing “this” keyword in their body, but to access current class static methods from some other method we are able to use “this” keyword.

 EX:

package com.durgasoft.app04.test;

class A{

int i = 10;

static int j = 20;

static void m1() {

System.out.println("m1-A");

//System.out.println(i);--> Error

System.out.println(j);

//System.out.println(this.j); --> Error

}

void m2() {

System.out.println("m2-A");

System.out.println(i);

System.out.println(j);

this.m1();

}

}

public class Test {

public static void main(String[] args){

A a = new A();

a.m1();

a.m2();

A.m1();

A a1 = null;

//a1.m2(); ---> NullPointerException

a1.m1();

}

}

Static Blocks:

—--------------

Static block is a set of instructions , it will recognize and execute at the time of loading the respective class bytecode to the memory.

Static blocks are able to allow only static members of the current class, not instance members of the current class.

Note: If we want to access instance members from a static block then we must create an object for the respective class and we must use the generated reference variable.

Static Blocks are not allowing this keyword.

EX:

package com.durgasoft.app04.test;

class A{

int i = 10;

static int j = 20;

static {

System.out.println("SB-A");

//System.out.println(i);---> Error

System.out.println(j);

//System.out.println(this.j); ---> Error

}

}

public class Test {

public static void main(String[] args){

A a = new A();

}

}

EX:

—-

package com.durgasoft.app04.test;

public class Test {

static int i = m1();

static int m1() {

System.out.println("Welcome To Durgasoft!");

System.exit(0);

return 10;

}

}

EX:

package com.durgasoft.app04.test;

public class Test {

static {

System.out.println("Welcome to Drgasoft!");

System.exit(0);

}

}

EX:

package com.durgasoft.app04.test;

public class Test {

static Object obj = new Object() {

{

System.out.println("Welcome To Durgasoft!");

System.exit(0);;

}

};

}

Class.forName() and newInstance()

—--------------------------------------

class A{

static{

System.out.println(“Class Loading….”);

}

A(){

System.out.println(“Object Creating……”);

}

}

A a = new A();

Where JVM will perform two actions.

1. JVM will load the specified class bytecode to the memory.
2. JVM will create an object for the loaded class.

In Java applications, we want to load a particular class bytecode to the memory not to create an object for the respective class. To achieve this requirement we have to use the following method from java.lang.Class .

public static Class forName(String className)throws ClassNotFoundException

EX: Class c = Class.forName(“com.durgasoft.app23.Employee”);

When Jvm executes the above instruction, JVM will perform the following actions.

1. JVM will take class name from the forName() method .
2. JVM will search for the respective class .class file at the following locations.

1. At current Location

2. At Java predefined Library

3. At the locations referred by “classpath” environment variable.

  3. If the required .class file is not identified at all the above locations

    then JVM will raise an exception like

    java.lang.ClassNotFoundfException.

4. If the required .class file exists at either of the above locations

then JVM will load its bytecode to the memory.

5.  After loading class bytecode, JVM will store metadata of the

 loaded class by creating an object for java.lang.Class .

EX:

package com.durgasoft.app05.test;

class A{

static {

System.out.println("Class Loading......");

}

public A() {

System.out.println("Object Creating.....");

}

}

public class Test {

public static void main(String[] args)throws Exception {

Class cls = Class.forName("com.durgasoft.app05.test.A");

}

}

newInstance() method:

—------------------------

If we load a particular class bytecode by using Class.forName() method and if we want to create an object for the loaded class then we have to use the following method from java.lang.Class .

public Object newInstance()throws InstantiationException, IllegalAccessException

EX: Object obj = cls.newInstance();

When JVM executes the above code , JVM will perform the following actions.

1. JVM will search for a non private and 0- arg constructor in the loaded class.
2. If the required constructor exists in the loaded class then JVM will execute the required constructor and JVM will create an object for the loaded class.
3. If the constructor is a parameterized constructor then JVM will raise an exception like java.lang.InstantiationException.
4. If the constructor is private constructor then JVM will raise an exception like java.lang.IllegalAccessException.
5. If the constructor is both private and parameterized

EX:

package com.durgasoft.app05.test;

class A{

static {

System.out.println("Class Loading......");

}

public A() {

System.out.println("Object Creating.....");

}

}

public class Test {

public static void main(String[] args)throws Exception {

Class cls = Class.forName("com.durgasoft.app05.test.A");

Object obj = cls.newInstance();

}

}

Exception:

Exception is an unexpected event occurred at runtime in our program , it may be provided by the users while entering dynamic input in java applications, it may be provided by the Database while executing sql queries in JDBC Applications , it may be provided by the network when we establish connection between Client and Server in Distributed applications,....... Causes abnormal termination to the applications.

Types of Exception :

1. Predefined Exceptions

1. CheckedException

SQLException, ClassNotFoundException,.....

1. Pure Checked Exceptions:

IOException

1. Partially Checked Exceptions

2.UncheckedException

ArithmeticException, NullPointerException,....

1. User defined Exceptions

There are two ways to handle exceptions in java.

1. By using throws keyword.
2. By Using try-catch-finally

throws Keyword:

It is able to throws out the generated exception from the present method to the caller method in order to handle exception.

EX:

—-

import java.io.\*;

class A{

void m1()throws Throwable{

m2();// Exception

}

void m2()throws Exception{

m3();// IOException

}

void m3()throws IOException{

throw new IOException("MyIOException");

}

}

class Test {

public static void main(String[] args)throws Throwable{

A a = new A();

a.m1();// Throwable

}

}

try-catch-finally:

—------------------

In Java applications, ‘throws’ keyword is not the real exception handler, it able to bypass exception from present method to the caller method in order to handle exception, but in java applications if we want to handle the exceptions where the exceptions are generated then we have to use “try-catch-finally”.

Syntax:

try{

—-----

—------

}catch(ExceptionName refvar){

—-----

—-----

}finally{

—----

—----

}

try block:

—----------

It is able to hold the instructions which may generate an exception.

In general, we will provide doubtful code inside the try block which may generate an exception or which may not generate an exception.

If we have any exception in the try block then JVM will perform the following actions.

1. JVM will recognize the exceptional situation in the try block.
2. JVM will create an object for the respective Exception class.
3. JVM will bypass the flow of execution to catch block by skipping all the remaining instructions in try block.
4. At the time of bypassing the flow of execution to catch block, JVM will pass the generated Exception object as parameter to catch block.
5. JVM will execute a catch block.
6. After executing the catch block, JVM will execute the finally block then all the remaining instructions after the finally block.

A person holding a circle

Description automatically generated

If no exception is identified in the try block then JVM will perform the following actions.

1. JVM will execute the complete try block.
2. At the end of try block, JVM will bypass flow of execution to finally block without executing catch block.
3. JVM will execute the finally block and all the remaining instructions after the finally block.

A white background with black text

Description automatically generated

Catch Block:

—-------------

It will be executed when we have an exception in the try block.

The main purpose of the catch block is to catch exceptions from try block and to display exceptions details to the users.

There are three approaches to display exception details.

1. By using printStackTrace() method
2. By Using System.out.println(e); that is toString() method.
3. By Using getMessage() method

By using printStackTrace() method

It is able to display the exception details like exception name, exception description and exception location.

By Using System.out.println(e);

It will execute the toString() method internally, it will return a string containing the exception details like Exception name and Exception description.

By Using getMessage():

It will return a string containing only an Exception description.

EX:

—-

class Test {

public static void main(String[] args){

try

{

int i = 10;

int j = 0;

float f = i/j;

}catch (Exception e){

e.printStackTrace();

System.out.println();

System.out.println(e);

System.out.println();

System.out.println(e.getMessage());

}finally{

}

}

}

OP:

D:\advjava10AM>javac Test.java

D:\advjava10AM>java Test

java.lang.ArithmeticException: / by zero

        at Test.main(Test.java:8)

java.lang.ArithmeticException: / by zero

/ by zero

finally:

It is able to hold a set of instructions which must be executed by JVM.

In try-catch-finally, only the finally block is giving guarantee for execution.

EX: We will keep resources closing operations in the finally block, because in applications resources must be closed if we open them in try block in order to avoid security problems and optimize memory utilization ,  where resources may be streams, Database Connections, Network connections,......

IN Java applications , if we want to use resources along with try-catch-finally then we have to use the following conventions.

1. Declare the resources before the try block.
2. Create the resources inside the try block.
3. Close the resources inside finally the block.

EX:

package com.durgasoft.app06.test;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Test {

public static void main(String[] args) {

BufferedReader bufferedReader = null;

//Connection connection = null;

//Socket socket= null;

try {

bufferedReader = new BufferedReader(new InputStreamReader(System.in));

//connection = DriverManager.getConnection(---,--,--);

// socket = new Socket("localhost", 4040);

System.out.print("Enter Data : ");

String data = bufferedReader.readLine();

System.out.println("Your Entered Data  : "+data);

} catch (Exception e) {

e.printStackTrace();

}finally {

try {

bufferedReader.close();

//connection.close();

//socket.close();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

If we use the above conventions to handle the exceptions from resources then we are able to get the following problems.

1. It is a bit of code confusion.
2. In this approach, developers must close the resources explicitly, it may not be guaranteed , if the resources are not closed then security problems will come.

To overcome the above problems , JDK1.7 version has provided an enhancement in try-catch-finally that is try-with-resources or Auto-Closeable resources.

1. try-with-resources is able to make JVM to close all the resources automatically when Flow of execution is coming out from try block, no need to close the resources explicitly, resources closing is guaranteed operation, with this option we are able to remove finally block in try-with -resources.
2. Its syntax is very simple.

Syntax:

try(

Resource-1;

Resource-2;

—---

){

—--

}catch(Exception e){

—--

}

Where the Resource-1, Resource-2,... must implement java.lang.AutoCloseable marker interface.

EX:

package com.durgasoft.app06.test;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Test {

public static void main(String[] args) {

try(

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

//Connection connection = DriverManager.getConnection(---,--,--);

//Socket socket = new Socket(--,---);

) {

System.out.print("Enter Data  : ");

String data = bufferedReader.readLine();

System.out.println("Your Eneterd Data   : "+data);

} catch (Exception e) {

e.printStackTrace();

}

}

}

ADV Java With Frameworks:

—-------------------------------

1. Adv Java
2. Hibernate
3. Spring
4. Spring Boot
5. Microservices
6. Web Services

Advanced Java:

—-----------------

1. JDBC
2. Servlets
3. JSPs
4. JSTL
5. EL

JDBC:

—------

In enterprise applications, it is the convention to manage organizations data like Employees details, Customers details, Products details,..... To manage all these details we have to use “Storage Areas”.

There are 2 types of Storage Areas.

1. Temporary STorage Areas
2. Permanent Storage Areas.

Temporary Storage AReas:

These are memory elements, they are able to store data temporarily.

EX: Buffers, Java Objects

Permanent Storage Areas:

These are able to store data permanently.

EX: File System

Database Management System

Data Warehouses

Q)What are the differences between File System and Database Management System?

—-----------------------------------------------------------------------

Ans:

—--

1. File is platform dependent, it is not suitable for the platform Independent Programming languages.

Database Management Systems are platform Independent, it is very much suitable for the platform independent programming languages.

1. File Systems are able to store less data.

Database Management Systems are able to store more data.

1. File Systems are able to provide less security.

Database Management SYstems are able to provide more security.

1. Filesystems are able to increase data redundancy.

DBMS are able to reduce Data Redundancy.

1. File Systems are not having Query languages support.

DBMS has Query Languages Support.

Q)What are the differences between Database Management Systems and Data Warehouses?

—----------------------------------------------------------------------------

Ans:

—--

1. DBMS is able to store less data when compared with Data Warehouses.

Data warehouses are able to store more data when compared with

DBMS.

1. DBMS do not have good data retrieval mechanisms.

Data warehouses are having data retrieval mechanisms in the form of Data Mining tech.

Q)What is the difference between Database and Database Management SYstem:

—----------------------------------------------------------------------

ANs:

—----

1. Database is a memory block to store data.

DBMS is a software system, it can be used to manage the data in the database by storing it in Database and by retrieving it from Database.

1. Database is the collection of interrelated data as a single unit.

DBMS is a collection of interrelated data and a set of programs to perform all the database operations.

There are three types of Database Management Systems.

1. RDBMS
2. OODBMS
3. ORDBMS

RDBMS:

It is able to represent data in the form of Tables.

To perform database operations with tables we have a Query Language that is SQL2.

OODBMS:

It is able to represent data in the form of Objects.

To perform database operations with objects then we have to use a separate query language that is “OQL”.

ORDBMS:

It is able to represent data in the form of tables and Objects.

To perform database operations we need a separate query language that is SQL3.

SQL3 = SQL2 + OQL.

**A diagram of data mining

Description automatically generated**

Query Processing System:

—----------------------------

In general, when we send an SQL query to a database, Database Engine will  execute the provided sql query by using the following phases.

1. Query Tokenization
2. Query Parsing
3. Query Optimization
4. Query Execution

Query Tokenization:

It is able to divide the provided sql query into a number of pieces called Tokens.

It will provide a stream of tokens as an output.

Query Parsing:

It will take a stream of tokens as input and it will create a tree representation called Query Tree, if Query Tree is successful then there are no errors in the provided query, if query tree is not successful then there are some errors in the provided sql query.

Query Optimization:

It is able to apply a number of optimization algorithms over the query tree in order to reduce memory utilization and in order to reduce query execution time and it will provide optimized query tree.

Query Execution:

It is able to have number interpreters to execute optimized query tree and it will perform the required database operation.

A diagram of a server

Description automatically generated

JDBC:

Def1:  JDBC is a step by step procedure to interact with the database from Java applications in order to perform database operations from java applications.

Def2: JDBC is a technology, not a programming language, it can be used to connect with databases from java applications in order to perform database operations from java applications.

Def3:  JDBC is an API [set of predefined classes and interfaces] , it can be used to interact with databases from java applications in order to perform database operations from java applications.

Def4: JDBC is an abstraction provided by SUN Microsystems and implemented by all the database vendors in order to interact with databases from java applications in order to perform database operations from java applications.

In general, in Jdbc applications, we will  perform the following actions.

1. We will prepare database logic in a Java application.
2. We will send database logic from a java application to the database.
3. At Database, Database Engine must execute database logic .
4. Database Engine must perform database operations on data.
5. Database Engine must send some response back to the java application.

The problem in the above approach is ,  
1. Database logic must be provided in java applications as per Java

representations.

2. At Databases, Database Engine is unable to execute java

represented database logic, because Database Engines are able to

understand only query language representations.

In this context, Database Engines required Database logic as per Query Languages representations, but, Java application is providing database logic in Java representations.

In the above context, to execute Jdbc applications we have to use the following two solutions.

1. Make the database Engine to understand Java language directly, it is not possible for every database Vendor.
2. Having converters to map Java representations to Query Language representations and from Query representations to java Representations.

To Execute Java applications, some translators or converters exist in outside called “Jdbc Drivers”.

A diagram of a driver's code

Description automatically generated

 Driver is an interface between Java applications and database, it is able to map Java representations with Database representations and Database representations with Java Representations.

To provide Driver, SUN Microsystems has provided an interface in the form of java.sql.Driver with some abstract methods and SUN Microsystems has given an intimation to all the database vendors to provide their own implementation classes for java.sql.Driver interface.

With the above convention, almost all the Database Vendors have provided their own implementation classes for java.sql.Driver interface in their database product softwares.

EX:

1. oracle.jdbc.OracleDriver provided by Oracle.
2. com.mysql.cj.jdbc.Driver provided by MySQL.

A diagram of a software system

Description automatically generated

With the above conventions, All the database vendors have provided their own Drivers , but, all these drivers are divided into the following five types.

1. Type-1 Driver
2. Type-2 Driver
3. Type-3 Driver
4. Type-4 Driver
5. Type-5 Driver

Type-1 Driver:

—----------------

1. Type-1 Driver is a model implementation or showcase or sample implementation for all the Drivers which are provided by the database vendors.
2. SUN Microsystems has provided Type-1 Driver in the form of sun.jdbc.odbc.JdbcOdbcDriver .
3. Type-1 driver is also called Jdbc-Odbc Driver or Bridge Driver.
4. Type-1 Driver was provided by SUN Microsystems with the interdependent on the Microsoft product ODBC Driver.
5. ODBC Driver is an open specification, it can be used to connect with any type of Database from JdbcOdbcDriver.
6. The Microsoft product ODBC Driver is very much compatible with the Microsoft products only like Windows operating system.
7. Due to the above reason, Type-1 Driver is very much compatible with the Microsoft products that are Windows Operating System only.
8. Type-1 Driver is less portable Driver, it will make Jdbc application less portable.
9. If we want to use Type-1 Driver in Jdbc applications then we must install an ODBC Driver in our system.
10. Type-1 Driver is slower Driver , because, to connect with Database from Java applications it required two times Conversions.

1. Java Representations to ODBC representations.

2. ODBC Representations to Database Representations

1. It will increase Jdbc applications execution time.
2. It will reduce Jdbc applications performance.
3. It is suggestible for simple Jdbc applications only as it is a model implementation for all the Drivers.
4. Type-1 Driver is suggestible for Standalone applications only, not suggestible for Server side applications like Web applications and Distributed Applications.
5. Type-1 Driver class [sun.jdbc.odbc.JdbcOdbcDriver] supported by JAVA up to its JDK1.7 version, not supported from JDK1.8 version. If we use Type-1 Driver in JDBC applications using JDK1.8 version or above versions then JVM will raise an exception like java.lang.ClassNotFoundException.

A diagram of a car wiring

Description automatically generated

Type-2 Driver:

—---------------

1. Type-2 Driver is also called “Part Java , Part native Driver” or simply Native Driver, because Type-2 Driver was designed on the basis of Java implementations and the native library provided by Database Vendor.
2. If we want to use Type-2 Driver in Jdbc applications then we have to install a database vendor provided native library.
3. Type-2 Driver is a more portable Driver when compared with Type-1 Driver, because Type-2 Driver does not require the Microsoft product ODBC Driver.
4. Type-2 Driver is faster when compared with Type-1 Driver, because Type-2 Driver does not require two times conversions in order to connect with Database.
5. Type-2 Driver will increase Jdbc applications performance.
6. Type-2 Driver is suggestible for Standalone Applications, it is not suggestible for Web applications and Distributed Applications.
7. Type-2 Driver is costfull Driver.

A diagram of a computer

Description automatically generated

Type-3 Driver:

—---------------

1. Type-3 Driver is also called “Middleware Database Access Server Driver” or SImply “Network Driver”.
2. Type-1 and Type-2 Drivers are suitable for Standalone Applications only, but Type-3 Driver was prepared exclusively  for Server side applications like Web applications and Distributed applications.
3. Type-3 Driver is a more portable Driver when compared with Type-1 driver and Type-2 Driver, because Type-3 Driver is not dependent on the Microsoft Product ODBC Driver and it is not dependent on the Database vendor provided native library.
4. Type-3 Driver is faster when compared with Type-1 and Type-2 Drivers, because Type-3 Driver is not required two times conversions to connect with databases and it is not required to load the database vendor provided native library.
5. Type-3 Driver will connect with the database on the basis of Database Server IP address port number.
6. Type-3 Driver is able to provide a very good environment to connect with multiple databases from multiple java applications at a time.

A diagram of a computer

Description automatically generated

Type-4 Driver:

—----------------

1. Type-4 Driver is called “Pure Java Driver” or “Thin Driver”, because Type-4 Driver was prepared completely on the basis of Java implementations.
2. Type-4 Driver is suitable for both Standalone applications and Distributed applications.
3. Type-4 Driver is a more portable Driver, it will work with or without the application server, it does not require the Microsoft product ODBC driver and Database vendor  provided native library.
4. It is a faster driver when compared with Type-1, Type-2 AND Type-3 drivers.
5. It is the most economical Driver, it is not costfull driver.

A diagram of a driver

Description automatically generated

Steps to prepare Jdbc Applications:

—----------------------------------------

1. Download And Install Database Software[Oracle 21xe, MySQL8.3].
2. Download and Install JAVA Software[JAVA7 and JAVA8 or JAVA17]
3. Download And Install  latest IDE[Eclipse, Intellij Idea]
4. Create Java Project with main class,  main() method in IDE.
5. Provide JDBC Application Logic in main() method,  in Main Class.
6. Load And Register Driver
7. Establish Connection between Java applications and Database.
8. Create Statement or PreparedStatement or CallableStatement as per the requirement.
9. Write and Execute SQL Query.
10. Close the Resources.

Oracle Database Installation:

—--------------------------------

MySQL Installation:

—-----------------------

JDBC Logic:

Load and Register Driver :

—----------------------------

1. Driver is an interface provided by SUN Microsystems and its implementation classes are provided by all the Database Vendors in their Database Softwares.
2. If we want to load Driver class to the memory, first we have to make available Driver class to the JDBC application , for this we have to set the “classpath” environment variable.

set classpath=C:\Oracle21XE\dbhomeXE\jdbc\lib\ojdbc11.jar;

Note: If we use IDE like Eclipse then we will add ojdbc11.jar file to the project build path.

1. If we use Type-1 Driver in Jdbc applications then it is not required to set the “classpath” environment variable, because Type-1 Driver class was provided by SUN Microsystems in the same Java software along with the Java predefined library.
2. If we want to use Type-1 Driver in a Jdbc application then we must Configure ODBC Driver in our computer.
3. Go to Control Panel
4. Select System And Security
5. Select Administrative Tools
6. Double Click on ODBC data sources 64 bit
7. Select User DSN tab
8. Click on Add Button
9. Select the required Database[Oracle DB21Home]
10. Click on Finish button
11. Provide DSN name[nag]
12. Click on Ok Button
13. Click on Ok Button

If we select MySQL Database then we have to provide the following details.

1. Go to Control Panel
2. Select System And Security
3. Select Administrative Tools
4. Double Click on ODBC data sources 64 bit
5. Select User DSN tab
6. Click on Add Button
7. Select MySQL ODBC8.0 UNICODE Driver.
8. Click on Finish button
9. Provide the following details
10. Data Source Name : durga
11. User : root
12. Password : root
13. Database Name : durgadb

    d. Click on Ok Button

    E. Click on Ok Button

5. When Driver class is available in Jdbc applications we have to load and Register Driver class, for this we have to use the following method from java.lang.Class.

public static Class forName(String DriverClass)throws ClassNotFoundException

EX: Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

EX: Class.forName(“oracle.jdbc.OracleDriver”);

EX: Class.forName(“com.mysql.cj.jdbc.Driver”);

6. When we execute the above instructions, JVM will perform the following actions.

1. JVM will take the driver class from forName() and JVM will search for the driver class .class file at the following locations.
2. At current Location
3. At Java predefined Library
4. AT the locations referred by classpath environment variable.

    2. If the required .class file is not available at all the above locations

then JVM will raise an exception like

java.lang.ClassNotFoundException.

    3. If the required Driver .class file exists at either of the above

locations then JVM will load Driver class bytecode to the

memory.

    4. At the time of loading driver class bytecode to the memory, JVM

will execute a static block, where inside the static block JVM will

execute DriverManager.registerDriver() method,  this method will register JDBC Driver with our application.

   5. After loading Driver class bytecode to the memory, JVM will

create java.lang.Class object with the metadata of the driver

class in Heap Memory.

Note1: Type-1 Driver is available in Java up to JDK1.7 version, not existing from jdk1.8 version onwards.

Note: “Load And Register Driver” step is optional when we use JDK1.8  or JDBC 3.0 version and Oracle11xe.

Note: “Load And Register Driver” step is optional when WE Use JDK1.8 version, JDBC3.0 version and MySQL 5.8 version..

Note: Load and Register Driver step is optional in the case of Type-1 Driver also.

In all the above cases, JVM will load and Register the driver when we establish Connection between java application and Database.

public class DriverManager{// SUN Microsystems

—----

public static void registerDriver(-----){

—-----Logic To register the Driver—----

}

—---

}

public interface Driver{// SUN Microsystems

—----

}

public class OracleDriver implements Driver{// Oracle

—----

static{

DriverManager.registerDriver();

}

—---

}

class JdbcApp{// By Developer

public static void main(String[] args)throws SQLException{

Class.forName(“oracle.jdbc.OracleDriver”);

}

}

2. Establish the Connection between Java application and database:

—---------------------------------------------------------------------------

To establish Connection between Java application and Database , we have to use the following method from java.sql.DriverManager class.

public static Connection getConnection(

String driverURL,

String dbUserName,

String dbPassword

)throws SQLException

Where dbUserName and dbPassword are the database username and database password which we provided at the time of database installation.

Where driverURL is an url to establish connection from java application and database.

IN JDBC, Every driver will have its own Driver class and driver URL.

Where Every driver URL will be in the following format.

MainProtocolName: SubProtocolName: DBName

Where MainProtocolName should be “jdbc”.

EX1: Type-1 Driver Provided by SUN Microsystems:

Driver Class : sun.jdbc.odbc.JdbcOdbcDriver

Driver URL  : jdbc:odbc:nag

EX2: Type-4 Driver provided by Oracle:

Driver Class : oracle.jdbc.OracleDriver[new]

     oracle.jdbc.driver.OracleDriver[Deprecated]

Driver URL   : jdbc:oracle:thin:@localhost:1521:xe

EX3: Type-4 Driver provided by MySQL:

Driver Class : com.mysql.cj.jdbc.Driver

Driver URL  : jdbc:mysql://localhost:3306/durgadb

EX:

Connection con = DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”, “system”, “durga”);

When we execute the above code, JVM will perform the following actions.

1. JVM will access the getConnection() method.
2. Where in the getConnection() method , JVM will access connect() method.
3. By the execution of connect() method, JVM will create a socket connection between java application and database  on the basis of the provided Driver URL.
4. W.r.t the Socket Connection , getConnection() method will create and return Connection object.

Q) In general, in Java applications, we are unable to create objects for interfaces, In JDBC API, java.sql.Connection is an interface,for which we are unable to create objects but how it is possible for getConnection() method to create an object for java.sql.Connection interface?

—----------------------------------------------------------------------------

Ans:

JDBC is an abstraction that is the collection of interfaces provided by SUN Microsystems and its implementation was provided by all the database vendors.

java.sql.Connection is an interface provided by SUN Microsystems as part of JDBC Abstraction , but its implementation classes are provided by almost all the database vendors as part of their own JDBC abstraction implementation.

In Jdbc applications , getConnection() method is not creating object for Connection interface directly, getConnection() method is creating object for the implementation class of the java.sql.Connection interface and getConnection() method is returning that Connection interfaces implementation class object.

 public interface Connection{// SUN Microsystems

}

public class OracleConnection implements Connection{// Oracle

}

public class DriverManager{// SUN Microsystems

public static Connection getConnection(---,--,--){

—connect()------

Connection con = new OracleConnection();

return con;

}

}

class JdbcApp{// Developer

public static void main(String[] args){

—--------

Connection con = DriverManager.getConnecgtion(--,--,--);

—--

}

}

3. Create Statement or PreparedStatement or CallableStatement as per the requirement:

—------------------------------------------------------------------------------

In Jdbc applications, After creating the Connection , Developers have to prepare sql queries and Developers have to execute sql queries, here in Jdbc applications to write and execute sql queries we need some predefined methods , here JDBC API has provided the requirement methods in the form of

1. Statement
2. PreparedStatement
3. CallableStatement

Q)What is the difference between Statement, PreparedStatement and CallableStatement?

—------------------------------------------------------------------------------

Ans:

—---

In Jdbc applications, when we want to execute all the sql queries as independent sql queries then we  will use Statement.

In Jdbc applications, if we want to execute the same sql query in the next sequence there to improve Jdbc applications performance we will use PreparedStatement.

In Jdbc applications, to access database Managed Stored Procedures and functions from Java applications we have to use CallableStatement.

To create Statement in Jdbc applications we have to use the following method from java.sql.Connection .

public Statement createStatement()throws SQLException

EX: Statement st = con.createStatement();

Q)In the Jdbc API, java.sql.Statement is an interface, for which we are unable to create an Object , but how is it possible for the createStatement()  method to create an object for Statement interface?

—-----------------------------------------------------------------------

Ans:

—----

JDBC is an abstraction provided by SUN Microsystems but its implementations are provided by all the Database vendors.

java.sql.Statement is an interface provided by SUN Microsystems as part of Jdbc abstraction, but its implementation classes are provided by the Database vendor as part of Jdbc abstraction implementation.

In Jdbc applications, createStatement() method is not creating an object for java.sql.Statement interface, it is creating an object for the implementation class of java.sql.Statement interface.

public interface Statement{// SUN

}

public class OracleStatementImpl implements Statement{// Oracle

}

public interface Connection{// SUN Microsystems

public Statement createStatement();

}

public class OracleConnectionImpl implements Connection{// Oracle

public Statement createStatement(){

—---

Statement st = new OracleStatementImpl();

return st;

}

}

class JdbcApp{// Developers

public static void main(String[] args){

—----

Statement st = con.createStatement();

—--

}

}

4. Write and Execute SQL Queries:

—---------------------------------------

In Jdbc applications, to write and execute sql queries we have to use the following methods.

1. executeQuery()
2. executeUpdate()
3. execute()

Types of SQL Queries:

1. Selection group : select

2. Non Selection Group: create, alter, drop, insert update,delete

Q)What is the difference between executeQuery() method, executeUpdate() method and execute() method?

—-------------------------------------------------------------------

Ans:

—---

In Jdbc applications, executeQuery() method can be used to execute only selection group sql queries like select sql query inorder to fetch the data from Database.

public ResultSet executeQuery(String query)throws SQLException

EX: ResultSet rs = st.executeQuery(“select \* from emp1”);

When we execute the above instruction, JVM will perform the following actions.

1. JVM will take the provided sql query from executeQuery() method.
2. JVM will send that select sql query to database Engine through Jdbc Driver and Connection.
3. At Database, database Engine will execute that select sql query by following Query Tokenization, Query Parsing , Query Optimization and Query Execution.
4. By executing the select sql query , Database Engine will fetch the data from the database.
5. Database Engine will send data to java application through the same connection and Jdbc Driver.
6. In The Java application, the fetched data will be stored in the form of an object called “ResultSet” object.
7. At Java application, the executeQuery() method will return the generated ResultSet object reference value.

In Jdbc applications, executeUpdate() method can be used to execute non select sql queries like create, alter, drop, insert, update, delete in order to perform manipulations in the database table.

public int executeUpdate(String query)throws SQLException

EX:

int rowCount = st.executeUpdate(“update emp1 set ESAL = ESAL + 500 where ESAL < 10000”);

When we execute the above instruction, JVM will perform the following actions.

1. JVM will take the provided non select sql query from executeUpdate() method.
2. JVM will send that non select sql query to database Engine through Jdbc Driver and Connection.
3. At Database, Database Engine will execute that non select sql query by following Query Tokenization, Query Parsing , Query Optimization and Query Execution.
4. By executing the non select sql query , Database Engine will do some manipulations on the number of records in the database table.
5. Database Engine will identify the number records which are affected[RowCount] with the provided non select sql query and Database Engine will  send the generated RowCount value to  java application through the same connection and Jdbc Driver.
6. In The Java application, the RowCount will be stored in the form of an integer value.
7. At Java application, the executeUpdate() method will return the generated RowCount value.

In Jdbc applications, the execute() method can be used to execute both the types of SQL queries. If we execute select sql qwerty with execute() method then execute() method will return true value. If we execute non select sql qwerty with execute() method then execute() method will return false value.

public boolean execute(String query)throws SQLException

EX:boolean b1 = st.execute(“select \* from emp1”);

System.out.println(b1); // true

EX:boolean b2 = st.execute(“update emp1 set ESAL = ESAL + 500 where ESAL < 10000”);

System.out.println(b2);// false

In Jdbc applications, if we execute select sql query with execute() method then JVM will send select sql query to database Engine, where Database Engine will fetch the data from database table and sends that data to Java application in the from of ResultSet, but execute() method will return true value.

In Jdbc applications, if we execute non select sql query with execute() method then JVM will send non select sql query to database Engine, where Database Engine will perform manipulations on  the data , identify the rowCount value and sends that RowCount value to Java application in the from of int value, but execute() method will return false value.

Note: In general, in jdbc applications, if we know the type of  sql query at the time of writing program then we are able to use either executeQuery() method or executeUpdate() method  depending on the type of sql query,   If we don't know the type of sql query at the time of writing jdbc application then we have to use execute() method.

6. Close the resources:

—---------------------------

In Jdbc applications , it is convention to close the resources like Jdbc COnnection, Statement,..... At the end of the Jdbc applications in order to avoid security problems and to utilize memory in an effective way.

rs.close();

st.close();

con.close();

Note: IN Jdbc applications, Closing connection is sufficient, because, if we close the COnnection then automatically all Statements and ResultSets will be closed.

con.close();

Note: If we use try-with-resources then it is not required to close any resource explicitly, automatically JVM will close all the resources which are declared in try-with-resources.

A screenshot of a computer

Description automatically generated

App01:

—------

Objective: Creating Table in database.

Database: Oracle

Driver: Type-1 Driver

JAVA Version: JDK1.7 version

EX:

**package** com.durgasoft.app01.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

//Load and Register Driver

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

// Establish COnnection between Java application and Database

Connection connection = DriverManager.*getConnection*("jdbc:odbc:nag", "system", "durga");

// Create Statement

Statement statement = connection.createStatement();

// Write and Execute SQL Query

String query = "create table emp1(ENO number(5) primary key, ENAME varchar2(10), ESAL float(5), EADDR varchar2(10))";

statement.executeUpdate(query);

System.***out***.println("Table emp1 Created Successfully");

// Close the resources

statement.close();

connection.close();

}

}

App02:

Objective: Creating Table in database.

Database: MySQL

Driver: Type-1 Driver

JAVA Version: JDK1.7 version

EX:

package com.durgasoft.app02.test;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

public class Test {

public static void main(String[] args)throws Exception {

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

Connection connection = DriverManager.getConnection("jdbc:odbc:durga", "root", "root");

Statement statement = connection.createStatement();

String query = "create table emp1(ENO int(5) primary key, ENAME char(10), ESAL float(5), EADDR char(10))";

statement.executeUpdate(query);

System.out.println("Table emp1 Createrd Successfully");

statement.close();

connection.close();

}

}

If we want to use the Type-4 driver provided by Oracle in JDBC Applications then we have to use the following driver class name and Driver URL.

Driver Class :  oracle.jdbc.OracleDriver

Driver URL  : jdbc:oracle:thin:@localhost:1521:xe [Oracle XE versions]

                 jdbc:oracle:thin:@locaslhost:1521:ORCL[OracleG/C version]

Note: To use the above driver class we have to keep ojdbc11.jar file in build path.

App03:

Objective: Creating Table in Database by taking table name as Dynamic input.

Driver : Type-4 Driver provided by Oracle .

EX:

**package** com.durgasoft.app03.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Class.*forName*("oracle.jdbc.OracleDriver");

Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

Statement statement = connection.createStatement();

BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

System.***out***.print("Table Name[Employee Related Name] : ");

String tname = bufferedReader.readLine();

statement.executeUpdate("create table "+tname+"(ENO number(3) primary key, ENAME varchar2(10), ESAL float(5), EADDR varchar2(10))");

System.***out***.println("Table "+tname+" Is Created Successfully");

bufferedReader.close();

statement.close();

connection.close();

}

}

If we want to use Type-4 Driver provided by MySQL then we have to use the following Driver class name and Driver URL.

Driver Class Name : com.mysql.cj.jdbc.Driver

Driver URL :  jdbc:mysql://localhost:3306/durgadb

If we want to use the above driver we must add the below jar file in Project Buildpath.

C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-j-8.0.31.jar

App04:

**package** com.durgasoft.app04.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

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

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

Statement statement = connection.createStatement();

BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

System.***out***.print("Table Name [Employee Related Name ] : ");

String tname = bufferedReader.readLine();

statement.executeUpdate("create table "+tname+"(ENO int(3) primary key, ENAME char(10), ESAL float(5), EADDR char(10))");

System.***out***.println("Table "+tname+" Created Succesfully");

bufferedReader.close();

statement.close();

connection.close();

}

}

EX:

—--

**package** com.durgasoft.app05.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

String primaryKeys = "";

**int** pkColumnCount = 0;

BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

Class.*forName*("oracle.jdbc.OracleDriver");

Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

Statement statement = connection.createStatement();

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

String tableName = bufferedReader.readLine();

String query = "create table "+tableName+"(";

**while**(**true**) {

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

String columnName = bufferedReader.readLine();

System.***out***.print("Column Type : ");

String columnType = bufferedReader.readLine();

System.***out***.print("Column Size : ");

**int** columnSize = Integer.*parseInt*(bufferedReader.readLine());

System.***out***.print("Is Primary Key[Yes/No]? : ");

String pkOption = bufferedReader.readLine();

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

pkColumnCount = pkColumnCount + 1;

**if**(pkColumnCount == 1) {

primaryKeys = primaryKeys + columnName;

}**else** {

primaryKeys = primaryKeys + "," + columnName;

}

}

query = query + columnName + " " + columnType + "(" + columnSize + "),";

System.***out***.print("Onemore Column?[Yes/No] : ");

String nextColumnOption = bufferedReader.readLine();

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

**continue**;

}**else** {

query = query + "primary key("+primaryKeys+"))";

**break**;

}

}

//System.out.println(query);

statement.executeUpdate(query);

System.***out***.println("Table "+tableName+" Created Successfully");

bufferedReader.close();

statement.close();

connection.close();

}

}

In Jdbc applications, we will use some resources like Connection, Streams, Statements, ResultSets,...... these resources may raise some exceptions like SQLException, IOException when we perform operations  like creating resources , closing resources ……..

In the above context, to handle the resources generated exceptions it is suggestible to use try-catch-finally instead of throws keyword.

If we want to use try-catch-finally in JDBC applications then we have to use the following conventions.

1. Declare the resources before try block.
2. Create the resources inside the try block.
3. Close the resources in the finally block.

EX:

BufferedReader br = null;

Connection con = null;

Statement st = null;

—--

—--

try{

br = new BufferedReader(new INputStreamReader(System.in));

con=DriverManager.getConnection(--,--,--);

st=con.createStatement();

}catch(Exception e){

e.printStackTrace();

}finally{

try{

con.close();

br.close();

—-

}catch(Exception e){

e.printStackTrace();

}

}

With the above approach we are able to get the following problems.

1. There is no guarantee for closing the resources as we are closing the resources explicitly.
2. The above approach is confusing code , because we have to write try-catch-finally inside finally block to handle the exceptions which are generated in close operations.

To overcome the above problems we have to use the try-with-resources option.

Syntax:

try(

Resource-1;

Resource-2;

—----

Resource-n;

){

}catch(Exception e){

}

To declare any resource with try-with-resources then the provided resource must be an implementation of java.lang.AutoCloseable marker interface.

If we declare the resources in try-with-resources then that resources will be closed by JVM automatically when flow of execution is coming out from try block.

EX:

—--

**package** com.durgasoft.app06.test;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

BufferedReader bufferedReader = **null**;

Connection connection = **null**;

Statement statement = **null**;

**try** {

Class.*forName*("oracle.jdbc.OracleDriver");

connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

statement = connection.createStatement();

bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

**while**(**true**) {

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

**int** eno = Integer.*parseInt*(bufferedReader.readLine());

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

String ename = bufferedReader.readLine();

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

**float** esal = Float.*parseFloat*(bufferedReader.readLine());

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

String eaddr = bufferedReader.readLine();

**int** rowCount = statement.executeUpdate("insert into emp1 values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");

**if**(rowCount == 1) {

System.***out***.println("Employee "+eno+" inserted Successfully");

}**else** {

System.***out***.println("Employee "+eno+" Insertion Failure");

}

System.***out***.print("Onemore Employee?[Yes/No] : ");

String userOption = bufferedReader.readLine();

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

**continue**;

}**else** {

**break**;

}

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

bufferedReader.close();

statement.close();

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

EX:

—--

**package** com.durgasoft.app07.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

BufferedReader bufferedReader = **null**;

**try** {

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

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

statement = connection.createStatement();

bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

System.***out***.print("Bonus Amount : ");

**int** bonus = Integer.*parseInt*(bufferedReader.readLine());

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

**float** salaryRange = Float.*parseFloat*(bufferedReader.readLine());

**int** rowCount = statement.executeUpdate("update emp1 set ESAL = ESAL + "+bonus+" where ESAL < "+salaryRange);

System.***out***.println("Employees Updated : "+rowCount);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

bufferedReader.close();

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

EX:

—-

**package** com.durgasoft.app08.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

BufferedReader bufferedReader = **null**;

**try** {

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

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb","root","root");

statement = connection.createStatement();

bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

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

**float** salaryRange = Float.*parseFloat*(bufferedReader.readLine());

**int** rowCount = statement.executeUpdate("delete from emp1 where ESAL < "+salaryRange);

System.***out***.println("Employees Deleted : "+rowCount);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

bufferedReader.close();

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

Q)IN Jdbc applications, if we execute the DML sql queries like insert, update and delete by using executeUpdate() method then Database will perform really manipulations on the number of records so executeUpdate() method will return the generated rowCount value. In Jdbc applications, if we execute DDL sql queries like create, alter and drop by using executeUpdate() method then database will create table or alter table or drop table , where Database Engine will not perform manipulations on the number of records, so Database Engine will not rowCount value to executeUpdate() method, here what value would be returned by executeUpdate() method?

—---------------------------------------------------------------------------

Ans:

—---

When we execute DDL sql queries like create, alter and drop by using executeUpdate() method then Database Engine will not perform records manipulations at database, Database Engine will not send rowCount value to Java application,  in this context , return a value from executeUpdate() method is completely depending on the driver which we used.

case#1: If we use Type-1 Driver provided by SUN Microsystems with Oracle Database then executeUpdate() method return ‘-1’ value.

**package** com.durgasoft.app09.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

**try** {

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

connection = DriverManager.*getConnection*("jdbc:odbc:nag", "system", "durga");

statement = connection.createStatement();

**int** rowCount1 = statement.executeUpdate("create table emp50(ENO number primary key, ENAME varchar2(10))");

**int** rowCount2 = statement.executeUpdate("drop table emp50");

System.***out***.println("create RowCount : "+rowCount1);

System.***out***.println("drop RowCount : "+rowCount2);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

Case#2:If we use Type-1 Driver provided by SUN Microsystems with mySQL Database then executeUpdate() method will return ‘0’ value.

EX:

**package** com.durgasoft.app09.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

**try** {

/\*

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

connection = DriverManager.getConnection("jdbc:odbc:nag", "system", "durga");

\*/

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

connection = DriverManager.*getConnection*("jdbc:odbc:durga", "root", "root");

statement = connection.createStatement();

//int rowCount1 = statement.executeUpdate("create table emp50(ENO number primary key, ENAME varchar2(10))");

**int** rowCount1 = statement.executeUpdate("create table emp50(ENO int(5) primary key, ENAME char(10))");

**int** rowCount2 = statement.executeUpdate("drop table emp50");

System.***out***.println("create RowCount : "+rowCount1);

System.***out***.println("drop RowCount : "+rowCount2);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

Case#3: If we use Type-4 Driver provided by Oracle and MySQL then executeUpdate() method will return  ‘0’ value.

EX:

**package** com.durgasoft.app09.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

**try** {

/\*

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

connection = DriverManager.getConnection("jdbc:odbc:nag", "system", "durga");

\*/

/\*

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

connection = DriverManager.getConnection("jdbc:odbc:durga", "root", "root");

\*/

/\*

Class.forName("oracle.jdbc.OracleDriver");

connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

\*/

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

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

statement = connection.createStatement();

//int rowCount1 = statement.executeUpdate("create table emp50(ENO number primary key, ENAME varchar2(10))");

**int** rowCount1 = statement.executeUpdate("create table emp50(ENO int(5) primary key, ENAME char(10))");

**int** rowCount2 = statement.executeUpdate("drop table emp50");

System.***out***.println("create RowCount : "+rowCount1);

System.***out***.println("drop RowCount : "+rowCount2);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

**ResultSet**:

ResultSet is an object, it is able to store the fetched data from Database.

In Jdbc applications, when we execute select sql query , Database Engine will fetch data from Database table and Database Engine will send that data to Java application, at Java application the fetched data will be stored in the form of ResultSet object.

ResultSet object is represented in the form of an interface java.sql.ResultSet provided by JDBC API as part of JDBC Abstraction and its implementation classes are provided by all the Database vendors as part of their JDBC abstraction implementation.

In Jdbc , to execute select sql query and to get ResultSet with the fetched data we will use the following method from java.sql.Statement.

public ResultSet executeQuery(String query)throws SQLException

EX: ResultSet rs = st.executeQuery(“select \* from emp1”);

After getting the ResultSet object , if we want to get data from the ResultSet object we have to use the following steps.

1. Check whether the next record exists or not from the current cursor position by using the following method from java.sql.ResultSet.

Public boolean next()throws SQLException

It will check whether the next record exists or not, if the next record exists then move the resultset cursor to next record position and return true value.

If no next record exists then return false value.

1. After getting the resultSet cursor to the record position then read data from columns individually.

public xxx getXxx(int columIndex or String columnName)throws SQLException

Xxx may be byte, short, int,....

EX: int eno = rs.getInt(1); or rs.getInt(“ENO”);

EX: String ename = rs.getString(2); or rs.getString(“ENAME”);

Repeat the above steps for all the records which exist in ResultSet.

A screenshot of a computer

Description automatically generated

EX:

**package** com.durgasoft.app10.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

ResultSet resultSet = **null**;

**try** {

Class.*forName*("oracle.jdbc.OracleDriver");

connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

statement = connection.createStatement();

resultSet = statement.executeQuery("select \* from emp1");

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

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

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

System.***out***.print(resultSet.getInt("ENO")+"\t");

System.***out***.print(resultSet.getString("ENAME")+"\t");

System.***out***.print(resultSet.getFloat("ESAL")+"\t");

System.***out***.print(resultSet.getString("EADDR")+"\n");

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

In Jdbc applications, we are able to use the execute() method to execute both the types of sql queries like select and non select sql queries.

public boolean execute(String query)throws SQLException

If we execute select sql query then execute() method will return true value and if we execute non select sql query then execute() method will return false value.

If we execute select sql query with execute() method then JVM will send the provided select sql query to database engine , where Database Engine will fetch the data from database table and It will send that fetched data to java application , At Java application the fetched data will be stored in the form of ResultSet object but execute() method will return true value.

In the above context, if we want to get the generated ResultSet object then we have to use the following method from java.sql.Statement.

public ResultSet getResultSet()throws SQLException

EX: ResultSet rs = st.getResultSet();

EX: boolean b = st.execute(“select \* from emp1”);

ResultSet rs = st.getResultSet();

EX:

**package** com.durgasoft.app11.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

ResultSet resultSet = **null**;

**try** {

Class.*forName*("oracle.jdbc.OracleDriver");

connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system","durga");

statement = connection.createStatement();

**boolean** b = statement.execute("select \* from emp1");

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

resultSet = statement.getResultSet();

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

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

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

System.***out***.print(resultSet.getInt("ENO")+"\t");

System.***out***.print(resultSet.getString("ENAME")+"\t");

System.***out***.print(resultSet.getFloat("ESAL")+"\t");

System.***out***.print(resultSet.getString("EADDR")+"\n");

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

If we execute non select sql query with execute() method then JVM will send that non select sql query to Database Engine, where Database Engine will execute non select sql query, perform database operations , find RowCount value and send that rowCount value to Java application as an integer value, but execute() method will return false value.

In the above context, to get the generated RowCount value we have to use the following method.

public int getUpdateCount()throws SQLException

EX:

boolean b = st.execute(“update emp1 set ESAL = ESAL + 500 where ESAL < 10000”);

int rowCount = st.getUpdateCount();

EX:

—--

**package** com.durgasoft.app12.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** Test {

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

Connection connection = **null**;

Statement statement = **null**;

**try** {

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

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

statement = connection.createStatement();

**boolean** b = statement.execute("update emp1 set ESAL = ESAL + 500 where ESAL < 10000");

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

**int** rowCount = statement.getUpdateCount();

System.***out***.println("Row Count : "+rowCount);

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**try** {

connection.close();

} **catch** (Exception e2) {

e2.printStackTrace();

}

}

}

}

EX: On try-with-resources:

**package** com.durgasoft.app13.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**public** **class** Test {

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

**try**(

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("select \* from emp1");

){

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

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

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

System.***out***.print(resultSet.getInt("ENO")+"\t");

System.***out***.print(resultSet.getString("ENAME")+"\t");

System.***out***.print(resultSet.getFloat("ESAL")+"\t");

System.***out***.print(resultSet.getString("EADDR")+"\n");

}

}**catch** (Exception e) {

e.printStackTrace();

}

}

}

**Metadata:**

Data about the data is called Metadata.

In Jdbc , there are two types of Metadata.

1. Database Metadata
2. ResultSetMetadata

Database Metadata:

Data about the database is called Database Metadata, where database Metadata includes database details like

1. Database Product Name
2. Database Product Version
3. Database Driver Minor and Major Versions
4. Database Supported SQL Keywords
5. Database Supported String functions
6. Database Supported Numeric Functions

—----

—----

To represent Database Metadata in Jdbc applications, JDBC has provided an interface in the form of “java.sql.DatabaseMetaData” and its implementation classes are provided by all the database vendors.

To get a DatabaseMetaData object we have to use the following method from java.sql.Connection.

public DatabaseMetaData getMetaData()throws SQLException

EX: DatabaseMetaData md = con.getMetaData();

To get the details of the Databases from databaseMetaData we have to use the following methods.

1. public String getDatabaseProductName()
2. public String getDatabaseProductVersion()
3. public int getDriverMinorVersion()
4. public int getDriverMajorVersion()
5. public String getSQLKeywords()
6. public String getNumericFunctions()
7. public String getStringFunctions()
8. public boolean supportsBatchUpdates()
9. public boolean supportsStoredProcedures()

—-----

—-----

Note: It will be very much useful to know some features of the Database when we are working with unknown databases.

EX:

—--

**package** com.durgasoft.app14.test;

**import** java.sql.Connection;

**import** java.sql.DatabaseMetaData;

**import** java.sql.DriverManager;

**public** **class** Test {

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

**try**(

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

) {

DatabaseMetaData databaseMetaData = connection.getMetaData();

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

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

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

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

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

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

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

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

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

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

ResultSetMetaData:

Data about the ResultSet is called “ResultSetMetaData”.

ResultSetMetaData includes the following details

1. Number of columns in ResultSet.
2. Column names
3. Column data Types
4. Column Sizes

—---

—---

To represent ResultSetMetaData , JDBC has provided an interface in the form of java.sql.ResultSetMetaData and its implementation classes are provided by all the Database vendors.

To get the ResultSetMetaData object we have to use the following method from java.sql.ResultSet.

public ResultSetMetaData getMetaData()throws SQLException

EX: ResultSetMetaData md = rs.getMetaData();

To get the details of ResultSet we have to use the following methods from ResultSetMetaData.

1. public int getColumnCount()
2. public String getColumnName(int columnIndex)
3. public String getColumnTypeName(int  columnIndex)
4. public int getColumnDisplaySize(int columnIndex)

—----

—----

EX:

—--

**package** com.durgasoft.app15.test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.Statement;

**public** **class** Test {

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

**try**(

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb","root","root");

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("select \* from emp1");

) {

ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

**int** columnCount = resultSetMetaData.getColumnCount();

System.***out***.println("No Of Columns : "+columnCount);

**for**(**int** columnIndex = 1; columnIndex <= columnCount; columnIndex++) {

System.***out***.println("Column Name : "+resultSetMetaData.getColumnName(columnIndex));

System.***out***.println("Column Type : "+resultSetMetaData.getColumnTypeName(columnIndex));

System.***out***.println("Column Size : "+resultSetMetaData.getColumnDisplaySize(columnIndex));

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

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

EX:

—--

**package** com.durgasoft.app16.test;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.Statement;

**public** **class** Test {

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

**try**(

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

Statement statement = connection.createStatement();

BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));

) {

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

String tableName = bufferedReader.readLine();

ResultSet resultSet = statement.executeQuery("select \* from "+tableName);

ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

**int** columnCount = resultSetMetaData.getColumnCount();

**for**(**int** columnIndex = 1; columnIndex <= columnCount; columnIndex++) {

System.***out***.print(resultSetMetaData.getColumnName(columnIndex)+"\t");

}

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

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

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

**for**(**int** columnIndex = 1; columnIndex <= columnCount; columnIndex++) {

System.***out***.print(resultSet.getString(columnIndex)+"\t");

}

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

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

IntelliJ Idea:

—---------------

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               Statement statement = connection.createStatement();

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               ){

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

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

           while (resultSet.next()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

EX:

—---

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

      try(

              BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

              Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

              Statement statement = connection.createStatement();

              ){

          System.*out*.print("Enter SQL Query : ");

          String query = bufferedReader.readLine();

          boolean b = statement.execute(query);

          if(b == true){

              ResultSet resultSet = statement.getResultSet();

              ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

              int coulumnCount = resultSetMetaData.getColumnCount();

              for (int colIndex = 1; colIndex <= coulumnCount; colIndex++){

                  System.*out*.print(resultSetMetaData.getColumnName(colIndex)+"\t");

              }

              System.*out*.println();

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

              while (resultSet.next()){

                  for (int colIndex = 1; colIndex <= coulumnCount; colIndex++){

                      System.*out*.print(resultSet.getString(colIndex)+"\t");

                  }

                  System.*out*.println();

              }

          }else{

              int rowCount = statement.getUpdateCount();

              System.*out*.println("RowCount : "+rowCount);

          }

      }catch(Exception e){

          e.printStackTrace();

      }

   }

}

Q)In Jdbc Applications, in general we will use executeUpdate() method to execute non select sql queries, but if execute select  sql query with executeUpdate() method then what will be the result in Jdbc applications?

—-----------------------------------------------------------------------------

Ans:

—---

If we provide select sql query as parameter to executeUpdate() method then JVM will send the provided select sql query to Database Engine, at database, Database engine will execute the provided select sql query , Database Engine will fetch the data from database , database Engine will send the fetched data to java application in the form of ResultSet object, but executeUpdate() method is expecting rowCount value from database , in this context generating an exception or not is completely depending on the Driver which we have used.

Case#1:

If we use Type-1 Driver with Oracle DSN and MySQL DSN then JVM will raise an exception like java.sql.SQLException: No row count was produced

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

      try(

              Connection connection = DriverManager.*getConnection*("jdbc:odbc:nag", "system", "durga");

              Statement statement = connection.createStatement();

              ){

           int rowCount = statement.executeUpdate("select \* from emp1");

      }catch(Exception e){

          e.printStackTrace();

      }

   }

}

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

       Statement statement = null;

       ResultSet resultSet = null;

       try {

           connection = DriverManager.*getConnection*("jdbc:odbc:durga", "root", "root");

           statement = connection.createStatement();

           int rowCount = statement.executeUpdate("select \* from emp1");

       }catch(Exception e){

           try {

               e.printStackTrace();

               resultSet = statement.getResultSet();

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

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

               while(resultSet.next()){

                   System.*out*.print(resultSet.getInt("ENO")+"\t");

                   System.*out*.print(resultSet.getString("ENAME")+"\t\t");

                   System.*out*.print(resultSet.getFloat("ESAL")+"\t");

                   System.*out*.print(resultSet.getString("EADDR")+"\n");

               }

           }catch(Exception e1){

               e1.printStackTrace();

           }

       }finally{

           try{

               connection.close();

           }catch(Exception e){

               e.printStackTrace();

           }

       }

   }

}

Output:

java.sql.SQLException: No row count was produced

at sun.jdbc.odbc.JdbcOdbcStatement.executeUpdate(JdbcOdbcStatement.java:295)

at Main.main(Main.java:15)

ENO ENAME ESAL EADDR

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

111 AAA 5500.0 Hyd

222 BBB 6500.0 Hyd

333 CCC 7500.0 Hyd

444 DDD 8500.0 Hyd

555 EEE 9500.0 Hyd

Case#2:

If we use Type-4 Driver provided by Oracle the JVM will not raise any exception, in this case executeUpdate() method will return an integer value that is same as the number of records which are existed in jthe generated ResultSet object.

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

      try(

              Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system","durga");

              Statement statement = connection.createStatement();

              ){

          int rowCount = statement.executeUpdate("select \* from emp1");

          System.*out*.println("RowCount : "+rowCount);

          ResultSet resultSet = statement.getResultSet();

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

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

          while(resultSet.next()) {

              System.*out*.print(resultSet.getInt("ENO") + "\t");

              System.*out*.print(resultSet.getString("ENAME") + "\t\t");

              System.*out*.print(resultSet.getFloat("ESAL") + "\t");

              System.*out*.print(resultSet.getString("EADDR") + "\n");

          }

      }catch(Exception e){

          e.printStackTrace();

      }

   }

}

Output:

RowCount : 5

ENO ENAME ESAL EADDR

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

111 AAA 5000.0 Hyd

222 BBB 6000.0 Hyd

333 CCC 7000.0 Hyd

444 DDD 8000.0 Hyd

555 EEE 9000.0 Hyd

Case#3:

If we use Type-4 Driver provided by MySQL then JVM will raise an exception like “java.sql.SQLException: Statement.executeUpdate() or Statement.executeLargeUpdate() cannot issue statements that produce result sets.”. In this case ResultSet is also not created.

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

      Connection connection = null;

      Statement statement = null;

      ResultSet resultSet = null;

       try{

*//Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga");*

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb","root","root");

           statement = connection.createStatement();

          int rowCount = statement.executeUpdate("select \* from emp1");

      }catch(Exception e){

           e.printStackTrace();

      }

   }

}

Output:

java.sql.SQLException: Statement.executeUpdate() or Statement.executeLargeUpdate() cannot issue statements that produce result sets.

Q)In Jdbc applications, in general, we are able to execute select sql queries by using executeQuery() method, but if we pass non select sql query as parameter to executeQuery() method then what will be the result in Jdbc applications?

—-----------------------------------------------------------------------------

Ans:

—---

In Jdbc applications , if we provide non select sql query as parameter to executeQuery() method then JVM will send the provided non select sql query to Database , at database , Database Engine will execute the provided non select sql query and it returns rowCount  value in the form of an integer value, but at java application executeQuery() method is expecting ResultSet , in this context with the mismatched results , generating an exception or not is completely depending on the Driver which we used.

 Case#1:

If we use Type-1 Driver with Oracle DSN and MySQL DSN then JVM will raise an exception like “java.sql.SQLException: No ResultSet was produced”

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

       Statement statement = null;

       ResultSet resultSet = null;

       try{

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

           connection = DriverManager.*getConnection*("jdbc:odbc:nag", "system", "durga");

           statement = connection.createStatement();

           resultSet = statement.executeQuery("update emp1 set ESAL = ESAL + 500 where ESAL < 10000");

       }catch(Exception e){

           try{

               e.printStackTrace();

               int rowCount = statement.getUpdateCount();

               System.*out*.println("RowCount   : "+rowCount);

           }catch(Exception ex){

               ex.printStackTrace();

           }

       }finally{

           try{

               connection.close();

           }catch(Exception e){

               e.printStackTrace();

           }

       }

   }

}

EX:

—---

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

       Statement statement = null;

       ResultSet resultSet = null;

       try{

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

           connection = DriverManager.*getConnection*("jdbc:odbc:durga", "root", "root");

           statement = connection.createStatement();

           resultSet = statement.executeQuery("update emp1 set ESAL = ESAL - 500 where ESAL < 10000");

       }catch(Exception e){

           try{

               e.printStackTrace();

               int rowCount = statement.getUpdateCount();

               System.*out*.println("RowCount   : "+rowCount);

           }catch(Exception ex){

               ex.printStackTrace();

           }

       }finally{

           try{

               connection.close();

           }catch(Exception e){

               e.printStackTrace();

           }

       }

   }

}

 Case#2:

If we use Type-4 Driver provided by Oracle then JVM will not raise any exception and JVM will return a default ResultSet object.

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               Statement statement = connection.createStatement();

               ){

           ResultSet resultSet = statement.executeQuery("update emp1 set ESAL = ESAL - 500 where ESAL < 10000");

           int rowCount = statement.getUpdateCount();

           System.*out*.println("RowCount   : "+rowCount);

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Case#3:

If we use Type-4 Driver provided by MySQL then JVM will raise an exception like “java.sql.SQLException: Statement.executeQuery() cannot issue statements that do not produce result sets.”

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

       Statement statement = null;

       try{

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement();

           ResultSet resultSet = statement.executeQuery("update emp1 set ESAL = ESAL + 500 where ESAL < 10000");

       }catch(Exception e){

           try {

               e.printStackTrace();

               int rowCount = statement.getUpdateCount();

               System.*out*.println("RowCount   : "+rowCount);

           }catch(Exception ex){

               ex.printStackTrace();

           }

       }finally{

           try{

               connection.close();

           }catch(Exception e){

               e.printStackTrace();

           }

       }

   }

}

Q)Write a Jdbc application to read data from the database table and display that data in the form of a text file with , separated?

—-------------------------------------------------------------------------

ENO,ENAME,ESAL,EADDR

111,AAA,5000,Hyd

222,BBB,6000,Hyd

333,CCC,7000,Hyd

EX:

—-

import java.io.FileOutputStream;

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement();

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               FileOutputStream fileOutputStream = new FileOutputStream("E:/documents/emp.txt");

               ){

           String data = "";

           ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

           int columnCount = resultSetMetaData.getColumnCount();

           int count = 0;

           for(int index = 1; index <= columnCount; index++){

               count = count + 1;

               if(count == 1) {

                   data = data + resultSetMetaData.getColumnName(index);

               }else{

                   data = data + "," + resultSetMetaData.getColumnName(index);

               }

           }

           data = data + "\n";

           while(resultSet.next()){

               data = data + resultSet.getInt("ENO")+",";

               data = data + resultSet.getString("ENAME")+",";

               data = data + resultSet.getFloat("ESAL")+",";

               data = data + resultSet.getString("EADDR")+"\n";

           }

           fileOutputStream.write(data.getBytes());

           System.*out*.println("Employee Data Transferred to E:/documents/emp.txt file");

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Q)Write a Jdbc application to read data from the database table and display that data in the form of a csv file?

—-----------------------------------------------------------------------------

EX:

—--

import java.io.FileOutputStream;

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement();

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               FileOutputStream fileOutputStream = new FileOutputStream("E:/documents/emp.csv");

       ){

           String data = "";

           ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

           int columnCount = resultSetMetaData.getColumnCount();

           int count = 0;

           for(int index = 1; index <= columnCount; index++){

               count = count + 1;

               if(count == 1) {

                   data = data + resultSetMetaData.getColumnName(index);

               }else{

                   data = data + "," + resultSetMetaData.getColumnName(index);

               }

           }

           data = data + "\n";

           while(resultSet.next()){

               data = data + resultSet.getInt("ENO")+",";

               data = data + resultSet.getString("ENAME")+",";

               data = data + resultSet.getFloat("ESAL")+",";

               data = data + resultSet.getString("EADDR")+"\n";

           }

           fileOutputStream.write(data.getBytes());

           System.*out*.println("Employee Data Transferred to E:/documents/emp.csv file");

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Q)Write a Jdbc program to read data from the database and to display the fetched data in the form of an html file?

—--------------------------------------------------------------------------

import java.io.FileOutputStream;

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement();

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               FileOutputStream fileOutputStream = new FileOutputStream("E:/documents/emp.html");

       ){

           String data = "<html><body><table align='center' border='1'>";

           ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

           int columnCount = resultSetMetaData.getColumnCount();

          data = data + "<tr>";

           for(int index = 1; index <= columnCount; index++){

                   data = data + "<th>" + resultSetMetaData.getColumnName(index) + "</th>";

           }

           data = data + "</tr>";

           while(resultSet.next()){

               data = data + "<tr>";

               data = data + "<td>" + resultSet.getInt("ENO")+"</td>";

               data = data + "<td>" + resultSet.getString("ENAME")+"</td>";

               data = data + "<td>" + resultSet.getFloat("ESAL")+"</td>";

               data = data + "<td>" + resultSet.getString("EADDR")+"</td>";

               data = data + "</tr>";

           }

           data = data + "</table></body></html>";

           fileOutputStream.write(data.getBytes());

           System.*out*.println("Employee Data Transferred to E:/documents/emp.html file");

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Application:

1. Add Student
2. Search Student
3. Update Student
4. Delete Student
5. Exit

Your Option : 1

Student Id   : S-111

Student Name : AAA

Student Address : Hyd

Student Inserted Sucecssfully

1. Add Student
2. Search Student
3. Update Student
4. Delete Student
5. Exit

Your Option : 2

Student Id  : S-111

Student Id   : S-111

Student Name  : AAA

Student Address : Hyd

1. Add Student
2. Search Student
3. Update Student
4. Delete Student
5. Exit

Your Option : 3

Student Id   : S-111

Student Name : [Old : AAA] , new : BBB

Student Address : [Old : Hyd], new : Chennai

Student Updated Successfully

1. Add Student
2. Search Student
3. Update Student
4. Delete Student
5. Exit

Your Option : 4

Student Id   : S-111

Student Deleted Successfully

1. Add Student
2. Search Student
3. Update Student
4. Delete Student
5. Exit

Your Option : 5

Thank You for Using the Student Management System.

A diagram of a diagram

Description automatically generated with medium confidence

Dao Layer:

1. Create an interface for the Dao layer.
2. Declare the required methods[CRUD] in the Dao interface.
3. Declare an implementation class for DAO interface.
4. Provide implementation for Dao method in the implementation class.
5. Prepare a Factory class for supplying dao objects to the Service Layer.

public interface StudentDao{

public String add(Student student);

public Student search(String sid);

public String update(Student student);

public String delete(String sid);

}

public class StudentDaoImpl implements StudentDao{

—-implementation for all methods of StudentDao interface–

}

public class StudentDaoFactory{

private static StudentDao studentDao;

static{

studentDao = new StudentDaoImpl();

}

public static StudentDao getStudentDao(){

return studentDao;

}

}

Service Layer:

—---------------

1. Declare an interface with Service layer methods.
2. Declare an implementation class to the Service Layer interface.
3. Provide implementation for all methods of Service Layer interface.
4. Prepare Factory class with Factory method to supply Service Layer object to Controller layer.

public interface StudentService{

public String addStudent(Student student);

public Student searchStudent(String sid);

  public String updateStudent(Student student);

public String deleteStudent(String sid);

}

public class StudentServiceImpl implements StudentService{

public String addStudent(Student student){

—---

StudentDao studentDao = StudentDaoFactory.getStudentDao();

String status = studentDao.add(student);

Return status;

}

public Student searchStudent(String sid){

StudentDao studentDao = StudentDaoFactory.getStudentDao();

String status = studentDao.search(sid);

Return status;

}

 public String updateStudent(Student student){

}

public String deleteStudent(String sid){

}

}

public class StudentServiceFactory{

private static StudentService studentService;

static{

studentSevice = new StudentServiceImpl();

}

public static StudentService getStudentService(){

return studentService;

}

}

Controller Layer:

—-----------------

Main Class and main() method.

Presentation Layer is a command prompt.

Main.java

import com.durgasoft.beans.Student;

import com.durgasoft.factory.ConnectionFactory;

import com.durgasoft.factory.StudentDaoFactory;

import com.durgasoft.factory.StudentServiceFactory;

import com.durgasoft.service.StudentService;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main {

   static {

       ConnectionFactory.*getConnection*();

       StudentServiceFactory.*getStudentService*();

       StudentDaoFactory.*getStudentDao*();

   }

   public static void main(String[] args) {

       try(

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               ){

           String sid = "", sname = "", saddr = "";

           Student student = null;

           String status = "";

           StudentService studentService = StudentServiceFactory.*getStudentService*();

           System.*out*.println("Welcome To Student Management System");

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

           while (true){

               System.*out*.println();

               System.*out*.println("1. ADD Student");

               System.*out*.println("2. SEARCH Student");

               System.*out*.println("3. UPDATE Student");

               System.*out*.println("4. DELETE Student");

               System.*out*.println("5. EXIT");

               System.*out*.print("Your Option[1,2,3,4,5] : ");

               int userOption = Integer.*parseInt*(bufferedReader.readLine());

               switch (userOption){

                   case 1:

                       System.*out*.println("ADD Student Module");

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

                       System.*out*.print("Student Id       : ");

                       sid = bufferedReader.readLine();

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

                       sname = bufferedReader.readLine();

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

                       saddr = bufferedReader.readLine();

                       student = new Student();

                       student.setSid(sid);

                       student.setSname(sname);

                       student.setSaddr(saddr);

                       status = studentService.addStudent(student);

                       if(status.equalsIgnoreCase("success")){

                           System.*out*.println("Status    : SUCCESS");

                       }

                       if(status.equalsIgnoreCase("failure")){

                           System.*out*.println("Status    : FAILURE");

                       }

                       if(status.equalsIgnoreCase("existed")){

                           System.*out*.println("Status    : EXISTED");

                       }

                       break;

                   case 2:

                       System.*out*.println("SEARCH Student Module");

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

                       System.*out*.print("Student Id       : ");

                       sid = bufferedReader.readLine();

                       student = studentService.searchStudent(sid);

                       if(student == null){

                           System.*out*.println("Status: STudent Not Existed");

                       }else{

                           System.*out*.println("Student Details");

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

                           System.*out*.println("Student Id     : "+student.getSid());

                           System.*out*.println("Student Name   : "+student.getSname());

                           System.*out*.println("Student Address: "+student.getSaddr());

                       }

                       break;

                   case 3:

                       System.*out*.println("Update Module");

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

                       System.*out*.print("Student Id     : ");

                       sid = bufferedReader.readLine();

                       student = studentService.searchStudent(sid);

                       if(student == null){

                           System.*out*.println("Status   : Student Not Existed");

                       }else{

                           System.*out*.print("Student Name [Old : "+student.getSname()+"] new : ");

                           String newSname = bufferedReader.readLine();

                           if(newSname == null || newSname.equals("")){

                               newSname = student.getSname();

                           }

                           System.*out*.print("Student Address [Old : "+student.getSaddr()+"] new : ");

                           String newSaddr = bufferedReader.readLine();

                           if(newSaddr == null || newSaddr.equals("")){

                               newSaddr = student.getSaddr();

                           }

                           Student newStudentData = new Student();

                           newStudentData.setSid(sid);

                           newStudentData.setSname(newSname);

                           newStudentData.setSaddr(newSaddr);

                           status = studentService.updateStudent(newStudentData);

                           System.*out*.println("Status   : "+status);

                       }

                       break;

                   case 4:

                       System.*out*.println("Delete Module");

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

                       System.*out*.print("Student Id     : ");

                       sid = bufferedReader.readLine();

                       student = studentService.searchStudent(sid);

                       if(student == null){

                           System.*out*.println("Status : Student Not Existed");

                       }else{

                           status = studentService.deleteStudent(sid);

                           System.*out*.println("Status    : "+status);

                       }

                       break;

                   case 5:

                       System.*out*.println("\*\*\*\*\*\*\*\*Thank You For Using Student Management System Application\*\*\*\*\*\*\*\*");

                       System.*exit*(0);

                       break;

                   default :

                       System.*out*.println("Wrong Option, pleace provide the options from 1,2,3,4, and 5");

                       break;

               }

           }

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Student.java

package com.durgasoft.beans;

public class Student {

   private String sid;

   private String sname;

   private String saddr;

   public String getSid() {

       return sid;

   }

   public void setSid(String sid) {

       this.sid = sid;

   }

   public String getSname() {

       return sname;

   }

   public void setSname(String sname) {

       this.sname = sname;

   }

   public String getSaddr() {

       return saddr;

   }

   public void setSaddr(String saddr) {

       this.saddr = saddr;

   }

}

StudentService.java

package com.durgasoft.service;

import com.durgasoft.beans.Student;

public interface StudentService {

   public String addStudent(Student student);

   public Student searchStudent(String sid);

   public String updateStudent(Student student);

   public String deleteStudent(String sid);

}

StudentServiceImplo.java

package com.durgasoft.service;

import com.durgasoft.beans.Student;

import com.durgasoft.dao.StudentDao;

import com.durgasoft.factory.StudentDaoFactory;

import com.durgasoft.factory.StudentServiceFactory;

public class StudentServiceImpl implements StudentService{

   @Override

   public String addStudent(Student student) {

       StudentDao studentDao = StudentDaoFactory.*getStudentDao*();

       String status = studentDao.add(student);

       return status;

   }

   @Override

   public Student searchStudent(String sid) {

       StudentDao studentDao = StudentDaoFactory.*getStudentDao*();

       Student student = studentDao.search(sid);

       return student;

   }

   @Override

   public String updateStudent(Student student) {

       StudentDao studentDao = StudentDaoFactory.*getStudentDao*();

       String status = studentDao.update(student);

       return status;

   }

   @Override

   public String deleteStudent(String sid) {

       StudentDao studentDao = StudentDaoFactory.*getStudentDao*();

       String status = studentDao.delete(sid);

       return status;

   }

}

StudentDao.java

package com.durgasoft.dao;

import com.durgasoft.beans.Student;

public interface StudentDao {

   public String add(Student student);

   public Student search(String sid);

   public String update(Student student);

   public String delete(String sid);

}

StudentDaoImpl.java

package com.durgasoft.dao;

import com.durgasoft.beans.Student;

import com.durgasoft.factory.ConnectionFactory;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

public class StudentDaoImpl implements StudentDao{

   @Override

   public String add(Student student) {

       String status = "";

       try{

           Connection connection = ConnectionFactory.*getConnection*();

           Statement statement = connection.createStatement();

*/\**

*ResultSet resultSet = statement.executeQuery("select \* from student where SID = '"+student.getSid()+"'");*

*boolean b = resultSet.next();*

*\*/*

Student student1 = search(student.getSid());

           if(student1 != null){

               status = "EXISTED";

           }else{

               String query = "insert into student values('"+student.getSid()+"','"+student.getSname()+"','"+student.getSaddr()+"')";

               int rowCount = statement.executeUpdate(query);

               if(rowCount == 1){

                   status = "SUCCESS";

               }else{

                   status = "FAILURE";

               }

           }

       }catch(Exception e){

           status = "FAILURE";

           e.printStackTrace();

       }

       return status;

   }

   @Override

   public Student search(String sid) {

       Student student = null;

       try{

           Connection connection = ConnectionFactory.*getConnection*();

           Statement statement = connection.createStatement();

           ResultSet resultSet = statement.executeQuery("select \* from student where SID = '"+sid+"'");

           boolean b = resultSet.next();

           if(b == true){

               student = new Student();

               student.setSid(resultSet.getString("SID"));

               student.setSname(resultSet.getString("SNAME"));

               student.setSaddr(resultSet.getString("SADDR"));

           }else{

               statement = null;

           }

       }catch(Exception e){

           e.printStackTrace();

       }

       return student;

   }

   @Override

   public String update(Student student) {

       String status = "";

       try{

           Connection connection = ConnectionFactory.*getConnection*();

           Statement statement = connection.createStatement();

           int rowCount = statement.executeUpdate("update student set SNAME = '"+student.getSname()+"', SADDR = '"+student.getSaddr()+"' where SID = '"+student.getSid()+"'");

           if(rowCount == 1){

               status = "SUCCESS";

           }else{

               status = "FAILURE";

           }

       }catch(Exception e){

           status = "FAILURE";

           e.printStackTrace();

       }

       return status;

   }

   @Override

   public String delete(String sid) {

       String status = "";

       try{

           Connection connection = ConnectionFactory.*getConnection*();

           Statement statement = connection.createStatement();

           int rowCount = statement.executeUpdate("delete from student where SID = '"+sid+"'");

           if(rowCount == 1){

               status = "SUCCESS";

           }else{

               status = "FAILURE";

           }

       }catch(Exception e){

           status = "FAILURE";

           e.printStackTrace();

       }

       return status;

   }

}

StudentServiceFactory.java

package com.durgasoft.factory;

import com.durgasoft.service.StudentService;

import com.durgasoft.service.StudentServiceImpl;

public class StudentServiceFactory {

   private static StudentService *studentService*;

   static {

*studentService* = new StudentServiceImpl();

   }

   public static StudentService getStudentService(){

       return *studentService*;

   }

}

StudentDaoFactory.java

package com.durgasoft.factory;

import com.durgasoft.dao.StudentDao;

import com.durgasoft.dao.StudentDaoImpl;

public class StudentDaoFactory {

   private static StudentDao *studentDao*;

   static {

*studentDao* = new StudentDaoImpl();

   }

   public static StudentDao getStudentDao(){

       return *studentDao*;

   }

}

ConnectionFactory.java

package com.durgasoft.factory;

import java.sql.Connection;

import java.sql.DriverManager;

public class ConnectionFactory {

   private static Connection *connection*;

   static{

       try{

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

*connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   public static Connection getConnection() {

       return *connection*;

   }

}

AWT-JDBC INtegration Examples:

EX1:

A close-up of a computer screen

Description automatically generated

EmployeeFrame.java

package com.durgasoft.ui;

import com.durgasoft.action.EmployeeAction;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

public class EmployeeFrame extends Frame implements ActionListener {

   Label l1, l2, l3, l4;

   TextField tf1, tf2, tf3, tf4;

   Button b;

   String status = "";

   public EmployeeFrame(){

       this.setVisible(true);

       this.setSize(500, 500);

       this.setTitle("Wmployee Add Frame");

       this.setBackground(Color.*GREEN*);

       this.setLayout(new FlowLayout());

       this.addWindowListener(new WindowAdapter() {

           @Override

           public void windowClosing(WindowEvent e) {

               System.*exit*(0);

           }

       });

       l1 = new Label("Employee Number  ");

       l2 = new Label("Employee Name    ");

       l3 = new Label("Employee Salary  ");

       l4 = new Label("Employee Address ");

       tf1 = new TextField(20);

       tf2 = new TextField(20);

       tf3 = new TextField(20);

       tf4 = new TextField(20);

       b = new Button("ADD");

       b.addActionListener(this);

       Font font = new Font("arial", Font.*BOLD*, 20);

       l1.setFont(font);

       l2.setFont(font);

       l3.setFont(font);

       l4.setFont(font);

       tf1.setFont(font);

       tf2.setFont(font);

       tf3.setFont(font);

       tf4.setFont(font);

       b.setFont(font);

       this.add(l1); this.add(tf1);

       this.add(l2); this.add(tf2);

       this.add(l3); this.add(tf3);

       this.add(l4); this.add(tf4);

       this.add(b);

   }

   @Override

   public void actionPerformed(ActionEvent ae) {

       try{

           int eno = Integer.*parseInt*(tf1.getText());

           String ename = tf2.getText();

           float esal = Float.*parseFloat*(tf3.getText());

           String eaddr = tf4.getText();

           EmployeeAction employeeAction = new EmployeeAction();

           status = employeeAction.add(eno, ename, esal, eaddr);

           repaint();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   @Override

   public void paint(Graphics g) {

       Font font = new Font("arial", Font.*BOLD*, 35);

       g.setFont(font);

       g.drawString("Status   : "+status, 50, 400);

   }

}

EmployeeAction.java

package com.durgasoft.action;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class EmployeeAction {

   Connection connection = null;

   Statement statement = null;

   ResultSet resultSet = null;

   public EmployeeAction(){

       try{

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   public String add(int eno, String ename, float esal, String eaddr){

       String status = "";

       try{

           resultSet = statement.executeQuery("select \* from emp1 where ENO = "+eno);

           boolean b = resultSet.next();

           if(b == true){

               status = "Employee Existed Already";

           }else{

               int rowCount = statement.executeUpdate("insert into emp1 values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");

               if(rowCount == 1){

                   status = "Employee Inserted Successfully";

               }else{

                   status = "Employee Insertion Failure";

               }

           }

       }catch(Exception e){

           status = "Failure";

           e.printStackTrace();

       }

       return status;

   }

}

Main.java

import com.durgasoft.ui.EmployeeFrame;

public class Main {

   public static void main(String[] args) {

       EmployeeFrame employeeFrame = new EmployeeFrame();

   }

}

EX2:

—---

A diagram of a diagram

Description automatically generated with medium confidence

Employee.java

package com.durgasoft.beans;

public class Employee {

   private int eno;

   private String ename;

   private float esal;

   private String eaddr;

   public int getEno() {

       return eno;

   }

   public void setEno(int eno) {

       this.eno = eno;

   }

   public String getEname() {

       return ename;

   }

   public void setEname(String ename) {

       this.ename = ename;

   }

   public float getEsal() {

       return esal;

   }

   public void setEsal(float esal) {

       this.esal = esal;

   }

   public String getEaddr() {

       return eaddr;

   }

   public void setEaddr(String eaddr) {

       this.eaddr = eaddr;

   }

}

EmployeeSearchFrame.java

package com.durgasoft.ui;

import com.durgasoft.action.EmployeeAction;

import com.durgasoft.beans.Employee;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

public class EmployeeSearchFrame extends Frame implements ActionListener {

   Label label;

   TextField textField;

   Button button;

   Employee employee;

   public EmployeeSearchFrame(){

       this.setVisible(true);

       this.setSize(500, 500);

       this.setTitle("Employee Search Frame");

       this.setBackground(Color.*cyan*);

       this.setLayout(new FlowLayout());

       this.addWindowListener(new WindowAdapter() {

           @Override

           public void windowClosing(WindowEvent e) {

               System.*exit*(0);

           }

       });

       label = new Label("Employee Number     ");

       textField = new TextField(20);

       button = new Button("SEARCH");

       button.addActionListener(this);

       Font font = new Font("arial", Font.*BOLD*, 20);

       label.setFont(font);

       textField.setFont(font);

       button.setFont(font);

       this.add(label);

       this.add(textField);

       this.add(button);

   }

   @Override

   public void actionPerformed(ActionEvent e) {

       try{

           int eno = Integer.*parseInt*(textField.getText());

           EmployeeAction employeeAction = new EmployeeAction();

            employee = employeeAction.search(eno);

           repaint();

       }catch(Exception ex){

           ex.printStackTrace();

       }

   }

   @Override

   public void paint(Graphics g) {

       Font f = new Font("arial", Font.*BOLD*, 30);

       g.setFont(f);

       if(employee == null){

           g.drawString("Status : Employee Not Existed", 50, 250);

       }else{

           g.drawString("Employee Number   : "+employee.getEno(), 50, 250);

           g.drawString("Employee Name     : "+employee.getEname(), 50, 300);

           g.drawString("Employee Salary   : "+employee.getEsal(), 50, 350);

           g.drawString("Employee Address  : "+employee.getEaddr(), 50, 400);

       }

   }

}

EmployeeAction.java

package com.durgasoft.action;

import com.durgasoft.beans.Employee;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class EmployeeAction {

   Connection connection = null;

   Statement statement = null;

   ResultSet resultSet = null;

   public EmployeeAction() {

       try{

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   public Employee search(int eno){

       Employee employee = null;

       try{

           resultSet = statement.executeQuery("select \* from emp1 where ENO = "+eno);

           boolean b = resultSet.next();

           if(b == true){

               employee = new Employee();

               employee.setEno(resultSet.getInt("ENO"));

               employee.setEname(resultSet.getString("ENAME"));

               employee.setEsal(resultSet.getFloat("ESAL"));

               employee.setEaddr(resultSet.getString("EADDR"));

           }else{

               employee = null;

           }

       }catch(Exception e){

           e.printStackTrace();

       }

       return employee;

   }

}

Main.java

import com.durgasoft.ui.EmployeeSearchFrame;

public class Main {

   public static void main(String[] args) {

       EmployeeSearchFrame employeeSearchFrame = new EmployeeSearchFrame();

   }

}

EX:

A screenshot of a computer

Description automatically generated

UserFrame.java

package com.durgasoft.ui;

import com.durgasoft.action.UserAction;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

public class UserFrame extends Frame implements ActionListener {

   Label l1, l2;

   TextField tf1, tf2;

   Button b;

   String status = "";

   public UserFrame(){

       this.setVisible(true);

       this.setSize(500,500);

       this.setTitle("User Login Frame");

       this.setBackground(Color.*pink*);

       this.setLayout(new FlowLayout());

       this.addWindowListener(new WindowAdapter() {

           @Override

           public void windowClosing(WindowEvent e) {

               System.*exit*(0);

           }

       });

       l1 = new Label("User Name   ");

       l2 = new Label("Password    ");

       tf1 = new TextField(20);

       tf2 = new TextField(20);

       tf2.setEchoChar('\*');

       b = new Button("Login");

       b.addActionListener(this);

       Font font = new Font("arial", Font.*BOLD*, 20);

       l1.setFont(font);

       l2.setFont(font);

       tf1.setFont(font);

       tf2.setFont(font);

       b.setFont(font);

       this.add(l1); this.add(tf1);

       this.add(l2); this.add(tf2);

       this.add(b);

   }

   @Override

   public void actionPerformed(ActionEvent e) {

       String uname = tf1.getText();

       String upwd = tf2.getText();

       UserAction userAction = new UserAction();

       status = userAction.checkLogin(uname, upwd);

       repaint();

   }

   @Override

   public void paint(Graphics g) {

       Font font = new Font("arial", Font.*BOLD*, 30);

       g.setFont(font);

       g.drawString("Status  : "+status, 50, 300);

   }

}

UserAction.java

package com.durgasoft.action;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class UserAction {

   Connection connection = null;

   Statement statement = null;

   ResultSet resultSet = null;

   public UserAction(){

       try{

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   public String checkLogin(String uname, String upwd) {

       String status = "";

       try{

           resultSet = statement.executeQuery("select \* from reg\_Users where UNAME = '"+uname+"' and UPWD = '"+upwd+"'");

           boolean b = resultSet.next();

           if(b == true){

               status = "SUCCESS";

           }else{

               status = "FAILURE";

           }

       }catch(Exception e){

           e.printStackTrace();

       }

       return status;

   }

}

Main.java

import com.durgasoft.ui.UserFrame;

public class Main {

   public static void main(String[] args) {

       UserFrame userFrame = new UserFrame();

   }

}

EX

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

DataFrame.java

package com.durgasoft.ui;

import com.durgasoft.action.DBAction;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.util.List;

import java.util.Map;

public class DataFrame extends Frame implements ActionListener {

   Label label;

   TextField textField;

   Button button;

   Map<String, List> tableData;

   public DataFrame(){

       this.setVisible(true);

       this.setSize(500, 500);

       this.setTitle("Data Frame");

       this.setLayout(new FlowLayout());

       this.setBackground(Color.*green*);

       this.addWindowListener(new WindowAdapter() {

           @Override

           public void windowClosing(WindowEvent e) {

               System.*exit*(0);

           }

       });

       label = new Label("Table Name");

       textField = new TextField(20);

       button = new Button("GetData");

       button.addActionListener(this);

       Font font = new Font("arial", Font.*BOLD*, 20);

       label.setFont(font);

       textField.setFont(font);

       button.setFont(font);

       this.add(label);

       this.add(textField);

       this.add(button);

   }

   @Override

   public void actionPerformed(ActionEvent e) {

       String tableName = textField.getText();

       DBAction dbAction = new DBAction();

       tableData = dbAction.getData(tableName);

       repaint();

   }

   @Override

   public void paint(Graphics g) {

       Font font = new Font("arial", Font.*BOLD*, 20);

       g.setFont(font);

       java.util.List headers = (java.util.List) tableData.get("headers");

       java.util.List body = (java.util.List) tableData.get("body");

       int x = 50;

       int y = 200;

       for (Object obj: headers) {

           g.drawString(obj.toString(),x, y);

           x = x + 100;

       }

       g.drawString("-----------------------------------------------", 50, 230);

       y = 250;

       for(Object obj: body){

           x = 50;

           for(Object ob: (java.util.List)obj){

               g.drawString(ob.toString(),x,y);

               x = x + 100;

           }

           y = y + 50;

       }

   }

}

DBAction.java

package com.durgasoft.action;

import java.sql.\*;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class DBAction {

   Connection connection = null;

   Statement statement = null;

   ResultSet resultSet = null;

   public DBAction(){

       try {

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

   public Map<String, List> getData(String tableName) {

       Map<String, List> tableData = null;

       try {

           tableData = new HashMap<>();

           resultSet = statement.executeQuery("select \* from "+tableName);

           ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

           List<String> headers = new ArrayList<>();

           int columnCount = resultSetMetaData.getColumnCount();

           for(int index = 1; index <= columnCount; index++){

               headers.add(resultSetMetaData.getColumnName(index));

           }

           List<List> data = new ArrayList<>();

           while(resultSet.next()){

               List<String> rowData = new ArrayList<>();

               for(int index = 1; index <= columnCount; index++){

                   rowData.add(resultSet.getString(index));

               }

               data.add(rowData);

           }

           tableData.put("headers", headers);

           tableData.put("body", data);

       }catch(Exception e){

           e.printStackTrace();

       }

       return tableData;

   }

}

Main.java

import com.durgasoft.ui.DataFrame;

public class Main {

   public static void main(String[] args) {

       DataFrame dataFrame = new DataFrame();

   }

}

EX:

—----

A screenshot of a computer

Description automatically generated

QueryBuilderFrame.java

package com.durgasoft.ui;

import com.durgasoft.action.QueryBuilderAction;

import javax.swing.\*;

import javax.swing.table.JTableHeader;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.util.List;

import java.util.Map;

public class QueryBuilderFrame extends Frame implements ActionListener {

   Label label = null;

   TextArea textArea = null;

   Button button = null;

   boolean b = false;

   QueryBuilderAction queryBuilderAction = null;

   public QueryBuilderFrame(){

       this.setVisible(true);

       this.setSize(600, 650);

       this.setTitle("Query Bilder Frame");

       this.setLayout(new FlowLayout());

       this.setBackground(Color.*cyan*);

       this.addWindowListener(new WindowAdapter() {

           @Override

           public void windowClosing(WindowEvent e) {

               System.*exit*(0);

           }

       });

       label = new Label("SQL Query");

       textArea = new TextArea(5, 30);

       button = new Button("Execute");

       button.addActionListener(this);

       Font font = new Font("arial", Font.*BOLD*, 20);

       label.setFont(font);

       textArea.setFont(font);

       button.setFont(font);

       this.add(label);

       this.add(textArea);

       this.add(button);

       queryBuilderAction = new QueryBuilderAction();

   }

   @Override

   public void actionPerformed(ActionEvent e) {

       String query = textArea.getText();

       b = queryBuilderAction.execute(query);

       repaint();

   }

   @Override

   public void paint(Graphics g) {

       Font font = new Font("arial", Font.*BOLD*, 30);

       g.setFont(font);

       if(b == true){

           Map data = queryBuilderAction.getData();

           Object[] headers = (Object[]) data.get("headers");

           Object[][] body = (Object[][]) data.get("body");

           JFrame jframe = new JFrame();

           jframe.setSize(400, 400);

           jframe.setVisible(true);

           jframe.setDefaultCloseOperation(JFrame.*HIDE\_ON\_CLOSE*);

           Container container = jframe.getContentPane();

           container.setLayout(new BorderLayout());

           JTable table = new JTable(body, headers);

           JTableHeader tableHeader = table.getTableHeader();

           container.add(tableHeader, BorderLayout.*NORTH*);

           container.add(table, BorderLayout.*CENTER*);

       }else{

           int rowCount = queryBuilderAction.getRowCount();

           g.drawString("RowCount  : "+rowCount, 50, 400);

       }

   }

}

QueryBuilderAction.java

package com.durgasoft.action;

import java.sql.\*;

import java.util.HashMap;

import java.util.Map;

public class QueryBuilderAction {

   Connection connection;

   Statement statement;

   ResultSet resultSet;

   public QueryBuilderAction(){

       try{

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

           statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

       }catch (Exception e){

           e.printStackTrace();

       }

   }

   public boolean execute(String query){

       boolean state = false;

       try {

           state = statement.execute(query);

       }catch(Exception e){

           e.printStackTrace();

       }

       return state;

   }

   public Map getData(){

       Map data = new HashMap();

       try {

           resultSet = statement.getResultSet();

           resultSet.last();

           int rows = resultSet.getRow();

           ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

           int columnCount = resultSetMetaData.getColumnCount();

           Object[] headers = new Object[columnCount];

           for(int columnIndex = 1; columnIndex <= columnCount; columnIndex++){

               headers[columnIndex-1] = resultSetMetaData.getColumnName(columnIndex);

           }

           data.put("headers", headers);

           Object[][] body = new Object[rows][columnCount];

           resultSet.beforeFirst();

           int rowNum = 0;

           while (resultSet.next()){

               for(int colIndex = 1; colIndex <= columnCount; colIndex++){

                   body[rowNum][colIndex-1] = resultSet.getString(colIndex);

               }

               rowNum++;

           }

           data.put("body", body);

       }catch (Exception e){

           e.printStackTrace();

       }

       return data;

   }

   public int getRowCount(){

       int rowCount = 0;

       try{

           rowCount = statement.getUpdateCount();

       }catch(Exception e){

           e.printStackTrace();

       }

       return rowCount;

   }

}

Main.java

import com.durgasoft.ui.QueryBuilderFrame;

public class Main {

   public static void main(String[] args) {

       QueryBuilderFrame frame = new QueryBuilderFrame();

   }

}

Types of ResultSets:

—-------------------------

IN Jdbc , ResultSets are classified in the following two ways.

1. On the basis of ResultSet Cursor Movement there are two types of ResultSets.
2. Forward Only ResultSet
3. Scrollable ResultSet
4. Scroll Sensitive ResultSet
5. Scroll Insensitive ResultSet.

2. On the basis of ResultSet concurrency there are two types of ResultSets.

1. Read only ResultSet
2. Updatable ResultSet

Forward Only resultSet is able to read data in forward direction only.

To represent Forward Only ResultSet ResultSet interface has provided a predefined constant in the form of

public static final int TYPE\_FORWARD\_ONLY;

Scrollable ResultSets are able to read data in both the forward direction and Backward direction.

There are two types of Scrollable ResultSets.

1. Scroll Sensitive ResultSet: If any ResultSet object allows the later database  modifications automatically then that ResultSet is a “Scroll Sensitive ResultSet” object.

To represent Scroll Sensitive ResultSet object ResultSet interface has provided a predefined constant.

public static int TYPE\_SCROLL\_SENSITIVE;

1. Scroll Insensitive ResultSet: If any ResultSet object does not allow the later database  modifications automatically then that ResultSet is a “Scroll Insensitive ResultSet” object.

To represent this ResultSet object ResultSet interface has

provided a predefined constant in the from of

public static final int TYPE\_SCROLL\_INSENSITIVE;

Read Only ResultSet is able to read data from the ResultSet object, not for modifications in the ResultSet object.

To represent Read only resultSet, ResultSet interface has provided a predefined constant in the from of

public static final int CONCUR\_READ\_ONLY.

Updatable ResultSet is able to perform updations in ResultSet objects.

To represent Updatable ResultSet object, ResultSet interface has provided a predefined constant in the from of

public static final int CONCUR\_UPDATABLE;

The default ResultSet type in JDBC is “Read Only and Forward Only”.

To create a particular type of resultSet in jdbc applications we have to provide the above specified ResultSet constants as parameters to the createStatement() method.

public Statement createStatement(

      int ForwardOnly/ScrollSensitive/ScrollInsensitive,

int ReadOnly/Updatable)throws SQLException

EX:

Statement st = con.createStatement(

 ResultSet.TYPE\_SCROLL\_SENSITIVE,

ResultSet.CONCUR\_UPDATABLE);

ResultSet rs = st.executeQuery(“select \* from emp1”);

Scrollable ResultSet:

—----------------------

Scrollable ResultSet is able to read data in both forward direction and backward direction.

To read data in Forward direction we have to use the following methods.

1. public boolean next()

It will check whether the next record is available or not, and it will return true value when the next record is identified. It will return false value when the next record is not identified.

1. public xxx getXxx(int columnINdex/String colName)

Where xxx may be a byte, short,.....

To read data in Backward direction we have to use the following methods.

1. public boolean previous()throws SQLException

It is able to check whether the previous record exists or not, it will return true value when the previous record exists, it will return false value when the previous record does not exist.

1. public xxx getXxx(int colindex/String colName)

Where xxx may be byte, short, int,.....

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               ) {

           System.*out*.println("Employee Data In Forward Direction");

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

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

           while(resultSet.next()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

           System.*out*.println("Employee Data In Backward Direction");

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

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

           while(resultSet.previous()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

Scroll Sensitive ResultSet Object:

—--------------------------------------

If any ResultSet object is able to allow the later database modifications automatically then that ResultSet is called “Scroll Sensitive ResultSet”.

To implement Scroll Sensitive ResultSet we have to use the following steps.

1. Create Scroll Sensitive ResultSet:

Statement st = con.createStatement(

ResultSet.TYPE\_SCROLL\_SENSITIVE,

ResultSet.CONCUR\_UPDATABLE);

ResultSet rs = st.executeQuery(“select \* from emp1”);

1. Display data from Scroll Sensitive ResultSet object:

while(rs.next()){

Sopln(rs.getXxx());

—----

}

1. Pause Application Flow of execution , perform Database operations at database and commit the database operations.

System.in.read(); // To pause application execution.

In database:

sql>update emp1 set ESAL = ESAL + 500 where ESAL < 10000;

sql>commit;

1. Go to the Java application, bring ResultSetCursor to the starting point of the ResultSet.

rs.beforeFirst();

1. Refresh each and every record and get data from the same ResultSet object:

while(rs.next()){

rs.refreshRow();

Sopln(rs.getXxx());

—----

}

Note: If the ResultSet data before updations and After Updations is same then the ResultSet object is not Scroll sensitive, if the ResultSet data is different before updations and after updations then that ResultSet is Scroll Sensitive ResultSet.

EX:

—---

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

      try {

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

         connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

         Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

         ResultSet resultSet = statement.executeQuery("select \* from emp1");

          System.*out*.println("Employee Details Before Updations");

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

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

          while (resultSet.next()){

              System.*out*.print(resultSet.getInt("ENO")+"\t");

              System.*out*.print(resultSet.getString("ENAME")+"\t");

              System.*out*.print(resultSet.getFloat("ESAL")+"\t");

              System.*out*.print(resultSet.getString("EADDR")+"\n");

          }

          System.*out*.println("Application is in Pausing state, please perform updations in Database");

          System.*in*.read();

          resultSet.beforeFirst();

          System.*out*.println("Employee Details After Updations");

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

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

          while (resultSet.next()){

              resultSet.refreshRow();

              System.*out*.print(resultSet.getInt("ENO")+"\t");

              System.*out*.print(resultSet.getString("ENAME")+"\t");

              System.*out*.print(resultSet.getFloat("ESAL")+"\t");

              System.*out*.print(resultSet.getString("EADDR")+"\n");

          }

      }catch (Exception e){

          e.printStackTrace();

      }finally{

          try{

             connection.close();

          }catch(Exception e){

              e.printStackTrace();

          }

      }

   }

}

OP:

Employee Details Before Updations

ENO ENAME ESAL EADDR

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

111 AAA 50000.0 Hyd

222 BBB 6000.0 Hyd

333 CCC 7000.0 Hyd

444 DDD 8000.0 Hyd

555 EEE 9000.0 Hyd

Application is in Pausing state, please perform updations in Database

Employee Details After Updations

ENO ENAME ESAL EADDR

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

111 AAA 5500.0 Hyd

222 BBB 6500.0 Hyd

333 CCC 7500.0 Hyd

444 DDD 8500.0 Hyd

555 EEE 9500.0 Hyd

Scroll Insensitive ResultSets:

—---------------------------------

These are the Scrollable ResultSet objects, which are not allowing later database updating automatically.

Inb Jdbc applications, Type-4 Driver provided by MySQL Database is supporting Scroll Sensitive ResultSet object, it is not supporting Scroll insensitive ResultSet object.

In Jdbc applications, Type-4 Driver provided by Oracle is not supporting both Scroll Sensitive ResultSet and Scroll insensitive ResultSet objects.

In Jdbc applications, Type-1 Driver with Oracle DSN is supporting Scroll Sensitive ResultSet object and it is not supporting Scroll Insensitive resultSet object.

In Jdbc applications, Type-1 Driver with MySQL DSN is supporting Scroll insensitive ResultSet, but it is not supporting Scroll Sensitive ResultSet object.

Scrollable ResultSet Methods:

—---------------------------------

The ResultSet interface has provided the following method for supporting Scrollable ResultSets.

1. public void beforeFirst():

It will bring the ResultSet cursor to the before first record position.

1. public boolean first():

It will bring the ResultSet cursor to the first record position.

1. public boolean next():

It will check whether the next record exists or not, if the next record exists then it will bring the cursor to the next record position.

1. public boolean previous():

It will check whether the previous record exists or not, if the previous record exists then it will bring the cursor to the previous record position.

1. public boolean last():

It will bring the ResultSet cursor to the last record position.

1. public void afterLast():

It will bring the ResultSet cursor to the after last record position.

1. public boolean absolute(int recordPosition):

It will bring the cursor to the specified record position, if we provide +ve value then the records count will be in forward direction from the first record, if we provide -ve value then records count will be in backward direction from the last record.

1. public boolean relative(int numberOfRecords):

It will bring the result set cursor by skipping the specified number of records , if we provide +ve value then the cursor moment will be in forward direction and the records count will be started from first record, if we provide -ve value then cursor moment will be will be in backward direction, where the records count will start from last record.

EX:

—--

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               ){

           resultSet.afterLast();

           resultSet.previous();

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.beforeFirst();

           resultSet.next();

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.last();

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.first();

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.absolute(4);

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.absolute(-4);

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.first();

           resultSet.relative(4);

           System.*out*.println(resultSet.getInt("ENO"));

           resultSet.last();

           resultSet.relative(-4);

           System.*out*.println(resultSet.getInt("ENO"));

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

Note: IN general, in Jdbc applications, it is not suggestible to use Scrollable ResultSet objects, because Scrollable ResultSet objects are heavy in nature, they may take more memory and more execution time.

Updatable ResultSets:

—-----------------------

The main purpose of Updatable ResultSet is to perform the database operations like insert, update, and delete without using the respective sql queries like insert, update and delete.

In general, when we are working with unknown databases there to perform the operations like insert, update and delete without knowing the respective sql queries we will use Updatable ResultSet.

Inserting Records Through the Updatable ResultSet object:

—-------------------------------------------------------------------

1. Create Updatable ResultSet object:

Statement st = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSet rs = st.executeQuery(“select \* from emp1”);

1. Move ResultSet cursor to the End of ResultSet for insertion of a new record.

rs.afterLast();

      Or

rs.moveToinsertRow();

1. INsert new record data in Updatable ResultSet object:

rs.updateInt(1, 555);

rs.updateString(2, “EEE”);

rs.updateFloat(3, 80000);

rs.updateString(4, “Hyd”);

1. Send new Record data from Updatable ResultSet to Database table;

rs.insertRow();

A screen shot of a computer

Description automatically generated

EX:

—---

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

               ){

           resultSet.moveToInsertRow();

           while(true){

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

               int eno = Integer.*parseInt*(bufferedReader.readLine());

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

               String ename = bufferedReader.readLine();

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

               float esal = Float.*parseFloat*(bufferedReader.readLine());

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

               String eaddr = bufferedReader.readLine();

               resultSet.updateInt(1, eno);

               resultSet.updateString(2, ename);

               resultSet.updateFloat(3, esal);

               resultSet.updateString(4, eaddr);

               resultSet.insertRow();

               System.*out*.println("Employee "+eno+" Inserted Successfully");

               System.*out*.print("One more Employee [Yes/No]?    : ");

               String option = bufferedReader.readLine();

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

                   continue;

               }else{

                   break;

               }

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

Updating Table through Updatable ResultSet:

—--------------------------------------------------

1. Create Updatable ResultSet object:

Statement st = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSet rs = st.executeQuery(“select \* from emp1”);

1. Move to the record and Update the values in the record:

rs.absolute(4);

rs.updateFloat(3, 8500);

1. Send the updations from Updatable ResultSet object to the database table:

rs.updateRow();

A screen shot of a computer

Description automatically generated

EX:

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

       ){

           while(resultSet.next()){

               float esal = resultSet.getFloat(3);

               if(esal < 10000){

                   esal = esal + 500;

                   resultSet.updateFloat(3, esal);

                   resultSet.updateRow();

                   System.*out*.println("Employee "+resultSet.getInt("ENO")+" Updated Successfully");

               }

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

Deleting Records through Updatable ResultSet:

—---------------------------------------------------

To delete records through Updatable ResultSet , go to the record position and use the following method.

rs.deleteRow();

EX:

—-

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*, ResultSet.*CONCUR\_UPDATABLE*);

               ResultSet resultSet = statement.executeQuery("select \* from emp1");

       ){

           while(resultSet.next()){

               float esal = resultSet.getFloat(3);

               int eno = resultSet.getInt(1);

               if(esal < 10000){

                   resultSet.deleteRow();

                   System.*out*.println("Employee "+eno+" Deleted Successfully");

               }

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

Note: IN Jdbc applications, it is not suggestible to use Updatable ResultSet and Scrollable ResultSets.

PreparedStatement:

—----------------------

In Jdbc applications, we will use Statement when we want to execute all the sql queries as independent sql queries.

In Jdbc applications, if we want to execute the same sql query in the next sequence there to improve application performance then we have to use preparedStatement.

For the above requirement if we use Statement then Database Engine will perform Query Tokenization, Query parsing, Query Optimization and Query Execution for each and every sql query execution without having variation from one time to another time , it will increase overall JDBC application execution time, it will reduce application performance.

A diagram of a computer program

Description automatically generated with medium confidence

To overcome the above problem we have to use PreparedStatement,   in PreparedStatement Database Engine will perform Query Tokenization, Query Parsing, Query Optimization only one time but Database Engine will perform the same operation number of times.

If we want to use PreparedStatement then we have to use the following steps.

1. Create PreparedStatement object with the query format:

PreparedStatement pst = con.prepareStatement("insert into emp1 values(?,?,?,?)");

Where ‘?’ is called Place Holder or Positional parameter.

When we execute the above instruction, the following actions will be performed .

1. JVM will send the provided query format to the Database Engine.
2. Database Engine will perform Query tokenization, Query Parsing and Query optimization over the provided sql query.
3. Database Engine will create a query plan or an execution plan with the provided positional parameters.
4. W.r.t the Query Plan JVM will create a PreparedStatement object in Java with the same positional parameters.

2. Set Data to the PreparedStatement object:

Public void setXxx(int paramIndex, xxx value)

EX: pst.setInt(1,111);

pst.setString(2, “AAA”);

pst.setFloat(3, 5000);

pst.setString(4, “Hyd”);

3. Execute the query:

1. If the query is select sql query then use the following method.

public ResultSet executeQuery()

1. If the query is a non select sql query then use the following method.

Public int executeUpdate()

A screenshot of a computer

Description automatically generated

EX:

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("insert into emp1 values(?,?,?,?)");

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               ){

           while (true){

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

               int eno = Integer.*parseInt*(bufferedReader.readLine());

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

               String ename = bufferedReader.readLine();

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

               float esal = Float.*parseFloat*(bufferedReader.readLine());

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

               String eaddr = bufferedReader.readLine();

               preparedStatement.setInt(1, eno);

               preparedStatement.setString(2, ename);

               preparedStatement.setFloat(3, esal);

               preparedStatement.setString(4, eaddr);

               int rowCount = preparedStatement.executeUpdate();

               if(rowCount == 1){

                   System.*out*.println("Employee "+eno+" Inserted Successfully");

               }else{

                   System.*out*.println("Employee insertion Failure");

               }

               System.*out*.print("One more Employee [Yes/No]?   : ");

               String option = bufferedReader.readLine();

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

                   continue;

               }else{

                   break;

               }

           }

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

EX:

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("update emp1 set ESAL = ESAL + ? where ESAL < ?");

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               ){

           System.*out*.print("bonus Amount   : ");

           int bonusAmount = Integer.*parseInt*(bufferedReader.readLine());

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

           float salRange = Float.*parseFloat*(bufferedReader.readLine());

           preparedStatement.setInt(1, bonusAmount);

           preparedStatement.setFloat(2, salRange);

           int rowCount = preparedStatement.executeUpdate();

           System.*out*.println("Employee Updated   : "+rowCount);

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

EX:

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("delete from emp1 where ESAL < ?");

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               ) {

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

           float salaryRange = Float.*parseFloat*(bufferedReader.readLine());

           preparedStatement.setFloat(1, salaryRange);

           int rowCount = preparedStatement.executeUpdate();

           System.*out*.println("Employee Deleted   : "+rowCount);

       }catch(Exception exception){

           exception.printStackTrace();

       }

   }

}

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("select \* from emp1");

               ResultSet resultSet = preparedStatement.executeQuery();

               ){

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

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

           while (resultSet.next()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Q)What are the differences between Statement and PreparedStatement?

—-----------------------------------------------------------

Ans:

—---

1. When we want to execute all the sql queries as individual sql queries there we will use Statement.

When we want to execute the same sql query in the next sequence there to improve performance of the application we will use PrepareStatement.

1. Statement is not supporting Dates manipulations.

PreparedStatement is supporting date manipulations.

1. Statement is not supporting BLOB and CLOB type data.

PreparedStatement is supporting BLOB and CLOB data.

1. Statement is able to provide less security for the data.

PreparedStatement is able to provide more security for the data.

1. Statement is able to provide SQL Injection problems.

PreparedStatement is able to solve SQL Injection problems.

Dates Manipulations in JDBC:

—----------------------------

In general, in database applications , it is very common to insert date values and to retrieve date values.

To insert date values into the database we have to use the following steps.

1. To insert current system date :
2. Create java.util.Date class object, it is representing the current system date.

java.util.Date date = new java.util.Date();

1. Get time value from java.util.Date object.

Long time = date.getTime();

1. Create java.sql.Date object with the above time value.

java.sql.Date date1 = new java.sql.Date(time);

1. Set java.sql.Date object to PreparedStatement.

pst.setDate(date1);

2. To insert a particular date value:

1. Create java.sql.Date object by using valueOf() method,

Date date = Date.valueOf(“1996-11-12”);

1. Set date value to PreparedStatement:

pst.setDate(date);

EX:

import java.sql.Connection;

import java.sql.Date;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("insert into emp2 values(?,?,?,?)");

               ){

           preparedStatement.setInt(1, 111);

           preparedStatement.setString(2, "Durga");

           Date date = Date.*valueOf*("1996-12-12");

           preparedStatement.setDate(3, date);

           java.util.Date date1 = new java.util.Date();

           long time = date1.getTime();

           Date doj = new Date(time);

           preparedStatement.setDate(4, doj);

           int rowCount = preparedStatement.executeUpdate();

           if(rowCount == 1){

               System.*out*.println("Employee inserted Successfully");

           }else{

               System.*out*.println("Employee Insertion Failure");

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

EX:

import java.sql.Connection;

import java.sql.Date;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.time.LocalDate;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("insert into emp2 values(?,?,?,?)");

               ){

           preparedStatement.setInt(1, 222);

           preparedStatement.setString(2, "Anil");

           Date date = Date.*valueOf*(LocalDate.*of*(1997,06,13));

           preparedStatement.setDate(3, date);

           Date doj = new Date(new java.util.Date().getTime());

           preparedStatement.setDate(4, doj);

           int rowCount = preparedStatement.executeUpdate();

           if(rowCount == 1){

               System.*out*.println("Employee inserted Successfully");

           }else{

               System.*out*.println("Employee Insertion Failure");

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

To read date values from Database table we have to use the following steps:

1. Create ResultSet object:

ResultSet rs = pst.executeQuery();

1. Read date value from ResultSet:

java.sql.Date date = rs.getDate(“DOB”);

1. Display Date value:

System.out.println(date);

EX:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("select \* from emp2 where ENO = ?");

               ){

           preparedStatement.setInt(1, 111);

           ResultSet resultSet = preparedStatement.executeQuery();

           boolean b = resultSet.next();

           if(b == true){

               System.*out*.println("Employee Details");

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

               System.*out*.println("Employee Number    : "+resultSet.getInt("ENO"));

               System.*out*.println("Employee Name      : "+resultSet.getString("ENAME"));

               System.*out*.println("Employee DOB       : "+resultSet.getDate("DOB"));

               System.*out*.println("Employee DOJ       : "+resultSet.getDate("DOJ"));

           }else{

               System.*out*.println("Employee Not Existed");

           }

       }catch (Exception e){

           e.printStackTrace();

       }

   }

}

SQL Injection Problem:

—----------------------

Consider the following program on the basis of Statement.

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Main {

   public static void main(String[] args) {

       try(

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               Statement statement = connection.createStatement();

               ) {

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

           String uname = bufferedReader.readLine();

           System.*out*.print("Password     : ");

           String upwd = bufferedReader.readLine();

           ResultSet resultSet = statement.executeQuery("select \* from reg\_Users where UNAME = '"+uname+"' and UPWD = '"+upwd+"'");

           boolean b = resultSet.next();

           if(b == true){

               System.*out*.println("Status : Login Success");

           }else{

               System.*out*.println("Status : Login Failure");

           }

       }catch(Exception exception){

           exception.printStackTrace();

       }

   }

}

On Console:

User Name    : durga

Password     : durga123

Status : Login Success

On Console:

User Name    : durga

Password     : abc

Status : Login Failure

IN the above execution process , if we provide the following dynamic input for the password data then we are able to get Login Success.

User Name    : durga

Password     : abc' OR '1'='1

Status : Login Success

In the above password data, we have provided some data which is not the right password but we have provided the remaining string to make the conditional expression in the SQL Query true, so we are able to get the Login Status like “Login Success”.

In the above application , if we provide the following dynamic to the username value then we are able to get Login Success.

User Name    : durga' #

Password     : aaa

Status : Login Success

IN the above dynamic input we have provided right username and immediately we have provided # in the dynamic input, where # is a comment sql query in MySQL, it will disable the remaining part of the query, it will make the conditional expression true and it will provide login success to the user.

In the above two cases we have injected some symbols or text in the sql query in order to get the unauthorized results , this problem is called “SQL Injection Problem”.

IN JDBC Applications . Statement is able to SQL Injections problem, but PreparedStatement will not allow SQL Injection problem, it solves SQL Injection Problem.

EX:

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       try(

               BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.*in*));

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("select \* from reg\_Users where UNAME = ? and UPWD = ?");

               ) {

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

           String uname = bufferedReader.readLine();

           System.*out*.print("Password     : ");

           String upwd = bufferedReader.readLine();

           preparedStatement.setString(1, uname);

           preparedStatement.setString(2, upwd);

           ResultSet resultSet = preparedStatement.executeQuery();

           boolean b = resultSet.next();

           if(b == true){

               System.*out*.println("Status : Login Success");

           }else{

               System.*out*.println("Status : Login Failure");

           }

       }catch(Exception exception){

           exception.printStackTrace();

       }

   }

}

On console

User Name    : durga

Password     : durga123

Status : Login Success

On console

User Name    : durga

Password     : abc

Status : Login Failure

On Console

User Name    : durga' #

Password     : aaa

Status : Login Failure

On Console

User Name    : durga

Password     : aaa' OR '1'='1

Status : Login Failure

**BLOB and CLOB:**

**—--------------**

**BLOB:**

**—-----**

BLOB: Binary Large Object

It is a Data type in Databases, it is able to manage large volumes of Binary Data.

EX:Image files.

In Jdbc applications if we want to insert an image file data then we have to use the following steps.

1. Create a table with a BLOB type column.

mysql> create table emp3(ENO int(5) primary key, ENAME char(10), EMP\_IMAGE blob);

Query OK, 0 rows affected, 1 warning (0.02 sec)

mysql> commit;

Query OK, 0 rows affected (0.00 sec)

mysql>

1. In Jdbc application, Create PreparedStatement object:

PreparedStatement pst = con.prepareStatement(“insert into emp3 values(?,?,?)”);

1. Set data to the positional parameters in PreparedStatement:
2. If the parameter is a normal parameter then use the setXxx() method.

pst.setInt(1,111);

pst.setString(2, “AAA”);

1. If the parameter is blob type parameter then use the following steps.
2. Create a java.io.File class object with the image file.

File f = new File(“D:\images\emp.jpg”);

1. Get image data into FileInputStream:

FileInputStream fis = new FileInputStream(f);

1. Create a Binary Stream with BLOB type positional parameters.

pst.setBinaryStream(3,fis,f.length());

4. Execute the Query:

int rowCount = pst.executeUpdate();

System.out.println(rowCount);

A screenshot of a computer program

Description automatically generated

EX:

import java.io.File;

import java.io.FileInputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

      try(

              Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

              PreparedStatement preparedStatement = connection.prepareStatement("insert into emp3 values(?,?,?)");

              ){

           preparedStatement.setInt(1, 111);

           preparedStatement.setString(2, "AAA");

           File file = new File("E:/images/baba.jpg");

           FileInputStream fileInputStream = new FileInputStream(file);

           preparedStatement.setBinaryStream(3, fileInputStream, file.length());

           int rowCount = preparedStatement.executeUpdate();

           if(rowCount == 1){

               System.*out*.println("Employee Inserted Successfully");

           }else{

               System.*out*.println("Employee Insertion Failure");

           }

          fileInputStream.close();

      }catch(Exception exception){

          exception.printStackTrace();

      }

   }

}

If we want to Read an Image data from database table then we have to use the following steps:

1. Create ResultSet object:

PreparedStatement pst = con.prepareStatement(“select \* from emp3 where ENO = ?”);

pst.setInt(1, 111);

ResultSet rs = pst.executeQuery();

1. Read data from ResultSet object:
2. Read normal data from the ResultSet object.

rs.next();

sopln(rs.getInt(“ENO”));

sopln(rs.getString(“ENAME”));

1. Read blob data as binary stream:

InputStream is = rs.getBinaryStream(“EMP\_IMAGE”);

FileOutputSTream fos = new FileOutputStream(“D:\images\empNew.jpg”);

int val = is.read();

while(val != -1){

fos.write(val);

Val = is.read();

}

is.close();

fos.close();

A screenshot of a computer program

Description automatically generated

import java.io.FileOutputStream;

import java.io.InputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/durgadb", "root", "root");

               PreparedStatement preparedStatement = connection.prepareStatement("select \* from emp3");

               ResultSet resultSet = preparedStatement.executeQuery();

               FileOutputStream fileOutputStream = new FileOutputStream("E:/images/baba1.jpg");

               ){

           resultSet.next();

           System.*out*.println("Employee Number    : "+resultSet.getInt("ENO"));

           System.*out*.println("Employee Name      : "+resultSet.getString("ENAME"));

           System.*out*.println("Employee Image     : E:/images/baba1.jpg");

           InputStream inputStream = resultSet.getBinaryStream("EMP\_IMAGE");

           int val = inputStream.read();

           while(val != -1){

               fileOutputStream.write(val);

               val = inputStream.read();

           }

           inputStream.close();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

CLOB:

—-----

CLOB: Character Large Object

It is a data type at Databases, it is able to manage large volumes of the character data.

CLOB operations are the same as the BLOB operations.

To perform CLOB operations we have to use the following replacements in BLOB operations.

BLOB —-------------------------> CLOB

FileInputStream —--------------> FileReader

FileOutputStream —-------------> FileWriter

setBinaryStream() -------------> setCharacterStream()

getBinaryStream()--------------> getBinaryStream()

InputStream  —-----------------> Reader

Jpg —--------------------------> xml, pdf,....

EX to INsert Clob in database:

import java.io.File;

import java.io.FileReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               PreparedStatement preparedStatement = connection.prepareStatement("insert into webapps values(?,?)");

               ){

           preparedStatement.setString(1, "APP01");

           File file = new File("D:\\AdvJava7am\\Servlets\\ECLIPSE-APPS\\app04-GenericServletDesign\\src\\main\\webapp\\WEB-INF\\web.xml");

           FileReader fileReader = new FileReader(file);

           preparedStatement.setClob(2, fileReader, file.length());

           int rowCount = preparedStatement.executeUpdate();

           if(rowCount == 1){

               System.*out*.println("Webapp Inserted Successfully");

           }else{

               System.*out*.println("Webapp Insertion Failure");

           }

           fileReader.close();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Ex To read CLOB from database Table:

import java.io.FileWriter;

import java.io.Reader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system","durga");

               PreparedStatement preparedStatement = connection.prepareStatement("select \* from webapps");

               ResultSet resultSet = preparedStatement.executeQuery();

               FileWriter fileWriter = new FileWriter("E:/documents/web.xml");

               ){

           resultSet.next();

           System.*out*.println("Application Name   : "+resultSet.getString("APPNAME"));

           Reader reader = resultSet.getCharacterStream("DEPL\_DESC");

           int val = reader.read();

           while(val != -1){

               fileWriter.write(val);

               val = reader.read();

           }

           reader.close();

       }catch(Exception e){

           e.printStackTrace();

       }

   }

}

Stored Procedures And Functions:

—---------------------------------

In database related applications, to improve code reusability and to implement complex database logics we will use Stored Procedures and Functions.

Q)What is the difference between Stored Procedures and Functions?

—----------------------------------------------------------

Ans:

—---

Stored Procedure is a set of sql queries maintained at the database to represent a particular action and it will not use a return statement to return a value.

Syntax:

create or replace procedure procedureName[(ParamList)]

AS

—Global Declarations—--

BEGIN

—---Instructions—-----

END ProcName;

/ —-> To save Procedure and to compile Procedure

Stored Function is a set of sql queries maintained at Database to represent a particular action and it will use a return statement to return a value.

create or replace function functionName[(ParamList)]return DataType

AS

—Global Declarations—--

BEGIN

—---instructions—---

return value;

END functionName;

/—--> To save function and to compile Function

In Stored Procedures and Functions there are three types of Parameters.

1. IN Type Parameters: They will take values from Procedure calls and they will provide values to the Procedure body.
2. OUT Type Parameters: They will take values from Procedure body and they will send values to the Procedure call
3. INOUT Type Parameters: These Parameters are acting as both IN Type Parameters and OUT Type Parameters.

Syntax:

VarName IN/OUT/INOUT DataType

In general, we will provide Stored procedures and functions at the database side , to access these stored procedures and functions from the Jdbc application we have to use java.sql.CallableStatement .

If we want to use CallableStatement in jdbc applications then we have to use the following steps.

1. Create CallableStatement object

To create a CallableStatement object we have to use the following method from java.sql.Connection.

public CallableStatement prepareCall(String Proc\_Call)

EX:

CallableStatement cst = con.prepareCall(“{call getSal(?,?)}”);

When we execute the above instruction , JVM will perform the following actions.

1. JVM will send the provided Procedure call to the Database Engine.
2. At the database, the Database engine will perform Query Tokenization, Query parsing, Query Optimization and Create a Query Plan with the place holders.
3. W.r.t the Query Plan, JVM will create CallableStatement object with the same Place holders.

2. Set Values to the IN type Parameters:

public void setXxx(int paramIndex, xxx value)

Where xxx may be byte, short, int,....

EX: cst.setInt(1,111);

3. Register OUT type Parameter with a particular Data Type:

public void registerOutParameter(int paramIndex, xxx

Type)

Where Type may be BYTE, SHORT, INTEGER,... constants

 from java.sql.Types class.

EX: cst.registerOutParameter(2, Types.FLOAT);

4. Execute Procedure:

public void execute()

cst.execute();

With the above method, Database Engine will execute the procedure , it will send values to the Out type Parameter in CallableStatement object.

5. Get Values from the Out Type Parameter:

public xxx getXxx(int paramindex)

Xxx may be byte, short, int,....

EX: float esal = cst.getFloat(2);

A screen shot of a computer

Description automatically generated

EX:

*/\**

*create or replace procedure getSal(no IN number, sal OUT float)*

*AS*

*BEGIN*

*select ESAL into sal from emp1 where ENO = no;*

*END getSal;*

*/*

*\*/*

*// Save the above procedure at Database.*

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Types;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");

               CallableStatement callableStatement = connection.prepareCall("{call getSal(?,?)}");

               ) {

           callableStatement.setInt(1, 111);

           callableStatement.registerOutParameter(2, Types.*FLOAT*);

           callableStatement.execute();

           System.*out*.println("111 Employee Salary    : "+callableStatement.getFloat(2));

       }catch (Exception exception){

           exception.printStackTrace();

       }

   }

}

EX:

*/\**

*create or replace function getAVGSal(no1 IN number, no2 IN number) return FLOAT*

*AS*

*sal1 float;*

*sal2 float;*

*BEGIN*

*select ESAL into sal1 from emp1 where ENO = no1;*

*select ESAL into sal2 from emp1 where ENO = no2;*

*return (sal1+sal2)/2;*

*END getAVGSal;*

*/*

*\*/*

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Types;

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               CallableStatement callableStatement = connection.prepareCall("{?=call getAVGSal(?,?)}");

               ) {

           callableStatement.setInt(2, 111);

           callableStatement.setInt(3, 222);

           callableStatement.registerOutParameter(1, Types.*FLOAT*);

           callableStatement.execute();

           callableStatement.execute();

           System.*out*.println("AVG Salary    : "+callableStatement.getFloat(1));

       }catch (Exception exception){

           exception.printStackTrace();

       }

   }

}

In Stored procedures and functions we will use cursor types[ SYS\_REFCURSOR] to represent records of data, to use cursor types in Stored procedures and functions we have to use the following steps.

1. Declare a CURSOR type parameter as OUT type parameter.
2. Open the CURSOR variable before writing the select sql query.

In Jdbc applications, we have to register the OUT type parameter as OracleTypes.CURSOR.

In Jdbc applications, after executing the procedure or function get ResultSet data by using getObject() method.

EX:

import oracle.jdbc.OracleTypes;

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

*/\**

*create or replace procedure getEmps(emps OUT SYS\_REFCURSOR)*

*AS*

*BEGIN*

*open emps for select \* from emp1;*

*END getEmps;*

*/*

*\*/*

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               CallableStatement callableStatement = connection.prepareCall("{call getEmps(?)}")

               ){

           callableStatement.registerOutParameter(1, OracleTypes.*CURSOR*);

           callableStatement.execute();

           ResultSet resultSet = (ResultSet) callableStatement.getObject(1);

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

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

           while(resultSet.next()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

       }catch(Exception exception){

           exception.printStackTrace();

       }

   }

}

EX:

import oracle.jdbc.OracleTypes;

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

*/\**

*create or replace function getEmployees return SYS\_REFCURSOR*

*AS*

*employees SYS\_REFCURSOR;*

*BEGIN*

*open employees for select \* from emp1;*

*return employees;*

*END getEmployees;*

*/*

*\*/*

public class Main {

   public static void main(String[] args) {

       try(

               Connection connection = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

               CallableStatement callableStatement = connection.prepareCall("{? = call getEmployees()}")

       ){

           callableStatement.registerOutParameter(1, OracleTypes.*CURSOR*);

           callableStatement.execute();

           ResultSet resultSet = (ResultSet) callableStatement.getObject(1);

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

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

           while(resultSet.next()){

               System.*out*.print(resultSet.getInt("ENO")+"\t");

               System.*out*.print(resultSet.getString("ENAME")+"\t\t");

               System.*out*.print(resultSet.getFloat("ESAL")+"\t");

               System.*out*.print(resultSet.getString("EADDR")+"\n");

           }

       }catch(Exception exception){

           exception.printStackTrace();

       }

   }

}

Transaction Management:

—-------------------------

Transaction is a unit of work performed by the front-end application on a Back-end system.

EX: In Bank applications

1. Deposit some amount in an account.
2. Withdraw some amount from an account.
3. Transfer some amount from one account to another account

In general, in database applications every transaction must have the following properties.

A : Atomicity

C : Consistency

I : Isolation

D : Durability

Atomicity:

—-----------

In every transaction , we may perform one or more number of operations, here Atomicity property says that “Either to perform all the operations or to perform None of the operations”.

In any transaction, if we perform all the operations then the state of the transaction is “Success”, if we perform None of the operations then the state of the transaction is “Failure”.

As per the Transactions Atomicity property, Every Transaction must be either in SUCCESS or in FAILURE.

Consistency

—-----------

In database applications, Before the transaction and after the transaction database state must be stable.

EX1: If we deposit 5000 in an account that has the balance 10000 then the consistency check will be performed like below.

Before Transaction                After Transaction

—-------------------              —-------------------

User Amount : 5000                User Amount: 0

Acc Balance : 10000               Acc Balance: 15000

—-----------------------------------------------------

Total       : 15000               Total      : 15000

Before Transaction and after Transaction the SUM is same, so Database must be in Stable , here Consistency is achieved

EX: In the transaction Transfer Funds , we transfer 5000 from an account that has 10000 balance to an account that has 15000 balance.

Before Transaction             After Transaction

—------------------            —------------------

Account-1: 10000                Account-1: 5000

Account-2: 15000                Account-2: 20000

—-----------------              —------------------

Total    : 25000                Total    : 25000

Before Transaction and After Transaction the sub of both the account balances is the same , so database state is Stable.

Isolation:

—------------

If we perform more than one transaction in a single data item then those transactions are called “Concurrent Transactions” and this process is called “Transactions Concurrency”.

In the Transaction’s concurrency there is a chance to get data inconsistency , it provides wrong results in database applications.

In this context, to provide data consistency we have to use Isolation.

In database applications, “Isolation property says that all the transactions must be executed individually, that is one transaction execution should not give effect to another transaction execution”.

Durability:

—-----------

In the transaction execution, if catastrophic failures occur then the database must preserve all the updates that we performed during the transactions, this property is “Durability”.

In Jdbc, we are able to get the support for the properties like Atomicity and Isolation.

To achieve Atomicity property in Jdbc applications we will use the following steps.

1. Create Connection Object and remove Auto-Commit from Connection:

When we submit an sql query to the Connection, Connection will send that sql query to the Database Engine and Connection will make the Database Engine to execute the query and to store results into the database table permanently, this nature of the Connection is Auto-Commit, it is against to the Transactions Atomicity property.

In the above context, to remove Transactions auto-commit nature we have to use the following method.

public void setAutoCommit(boolean b)

If we provide true value then Connection will be in Auto-Commit.

If we provide false value then Connection will not be in Auto-Commit.

EX: con.setAutoCommit(false);

1. Perform the database operations as per the requirement.
2. At the end of the database logic ,  perform either commit operation or rollback operation depending on the situation.

public void commit()

public void rollback()

In Transactions Management we have to use try-catch-finally , at the end of the try block we have to perform commit() operation, if any exception is identified in the try block then we have to perform rollback operation in the catch block.

EX:

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       Connection connection = null;

       try{

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

           connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3300/durgadb", "root", "root");

           connection.setAutoCommit(false);

           PreparedStatement preparedStatement1 = connection.prepareStatement("update account set BALANCE = BALANCE - ? where ACCNO = ?");

           preparedStatement1.setInt(1, 5000);

           preparedStatement1.setString(2, "a111");

           int rowCount1 = preparedStatement1.executeUpdate();

           float f = 100/0;

           PreparedStatement preparedStatement2 = connection.prepareStatement("update account set BALANCE = BALANCE + ? where ACCNO = ?");

           preparedStatement2.setInt(1, 5000);

           preparedStatement2.setString(2, "a222");

           int rowCount2 = preparedStatement2.executeUpdate();

           if(rowCount1 == 1 && rowCount2 == 1){

               connection.commit();

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

           }else{

               connection.rollback();

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

           }

       }catch(Exception e){

           try{

               connection.rollback();

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

           }catch(Exception e1){

               e1.printStackTrace();

           }

           e.printStackTrace();

       }

   }

}

EX:

—---

import java.sql.\*;

public class Main {

   public static void main(String[] args) {

       Connection oracleCon = null;

       Connection mysqlCon = null;

       PreparedStatement oraclePst = null;

       PreparedStatement mysqlPst = null;

       try {

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

           Class.*forName*("oracle.jdbc.OracleDriver");

           oracleCon = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");

           mysqlCon = DriverManager.*getConnection*("jdbc:mysql://localhost:3300/durgadb", "root", "root");

           oracleCon.setAutoCommit(false);

           mysqlCon.setAutoCommit(false);

           mysqlPst = mysqlCon.prepareStatement("update account set BALANCE = BALANCE - ? where ACCNO = ?");

           mysqlPst.setInt(1, 5000);

           mysqlPst.setString(2, "a222");

           int rowCount1 = mysqlPst.executeUpdate();

           float f = 100/0;

           oraclePst = oracleCon.prepareStatement("update account set BALANCE = BALANCE + ? where ACCNO = ?");

           oraclePst.setInt(1, 5000);

           oraclePst.setString(2, "b222");

           int rowCount2 = oraclePst.executeUpdate();

           if(rowCount1 == 1 && rowCount2 == 1){

               mysqlCon.commit();

               oracleCon.commit();

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

           }else{

               mysqlCon.rollback();

               oracleCon.rollback();

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

           }

       }catch(Exception exception){

           try{

               oracleCon.rollback();

               mysqlCon.rollback();

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

           }catch(Exception exception1){

               exception1.printStackTrace();

           }

           exception.printStackTrace();

       }

   }

}

Isolation Levels:

—----------------

Nagoor Babu Class Room : 7386095600

Name

Adm Num

Email Id

Mobile Num

Courses

Timing

Pay slip

https://tinyurl.com/advjava10annotes

Night 9:00PM Java Versions : JAVA8 version Features: <https://attendee.gotowebinar.com/register/2348474801872643416>