**J2EE**

**(Shishira Bhatt)**

**Done By Azam Amir Reza azam20104u@gmail.com**

**28-Sep-15:**

**Content:**

* Basic on JDBC
* Understand the API
* JDBC architecture
* Advantages
* Understanding the server and its internal
* Installation of DD server and DB Client
* JDBC steps
* Different types of statements
* Prepared statements
* Callable statements
* Batch updates
* JDBC transaction
* Stored procedure
* Execution of stored procedure to callable statements
* Metadata
* Types of metadata
* Types of drivers
* Connection pool
* Design patterns
* Best practices
* Usage of constant file

**JEE (Java Enterprise Edition):**

* Web application architecture
* Understanding web server internals and installation and configuration
* Understanding protocol and its types
* Static and dynamic web application
* Servlet
* Servlet life cycle
* Servlet config
* Servlet context
* Servlet chaining
* Request dispatcher
* Understanding HttpServlet
* Session Management

1. Cookies
2. http session

**JSP (Java Server Page):**

* Basics of JSP
* JSP elements
* Implicit objects
* JSP life cycle
* JSP servlet communication
* Standard action
* Exception handling in JSP

**29-Sep-15:**

**Config Files (Configuration Files):**

* Config file is a file which can store configuration details of data ,it can be system configuration or server configuration or project configuration
* Generally the config files are “.**xml**” and “.**properties**”
* For “.**xml**” and “.**properties**” are platform independent as well as technology independent

**JAR Files (Java Arch File):**

* JAR file is a compress format of java and its related resources
* JAR file generally contents java executable like “.class” and configuration files
* We can also keep “.java” files but it’s not a good practice

JAVA

JAR

.class files

.java

Config

.xml

.properties

**Std (Standard) for package:**

1. Must be lower case
2. No special keywords
3. No keywords must be used

src

com/org

Company

Application

Module1

.class

Module2

.class

Module3

.class

……………………….etc

***Inbuilt java packages:***

* Java.lang
* Java.util
* Java.sql
* …..etc

**30-Sep-15:**

|  |
| --- |
| <<**abstract**>>  AbstractClass |
| Data member |
| Method |

|  |
| --- |
| <<**interface**>>  Inf |
| Data member |
| Method |

***UML (Unified Modeling Lang):***

Notations are

**+ 🡪**public

- 🡪private

# 🡪protect

* **When the method return type is interface then the method must return implementation object.**

Ex, class Test {

Inf testMeth(){//Interface as return type

return new Impl();//returning implementation object

}

}

* **When a method parameter is super class, then argument of the method must be subclass object.**

**19-Oct-15**

**Generating of jar file using eclipse:**

* Right click on the project export
* Java
* Jar
* Select appropriate option
* Brows the destination where save the jar files

**20-Oct-15:**

* *It’s always a good practice to refer internal library then external library.*

**Use the member of JAR files:**

1. To use the member of jar files, first we need to import them.
2. Step2 to refer the jar files:
3. Copy the required jar file into project lib folder

2nd write click on the project select properties option and then click java build path then libraries then add jar or add external jar

Brows the jar file click on ok

**Class Load:**

+class🡪public class

* Loading a class (.class) into the JVM memory is called class loading
* A class is generally loaded when any of the member are called(methods, constructor, or variable🡪callable)

|  |  |  |  |
| --- | --- | --- | --- |
| Class L  {  Int ml=20;  } | Class X extends Y  {  Static void mx(){  Sop(new L().ml)  }  } | Class C{  Void mc(){  X.mx  }  } | Class Test{  Psvm{  New C.mc();  }  } |

* It is possible to load a class without calling any of its members.
* forName is static method present in java.lang.Class. the method takes fully qualified class name as String arguments

Class.forName(“com.studentcomp.practiceapp.Fish”);

The forName method possible throws a checked exception or ClassNotFoundException.

It is possible to load a class which is the part of jar file.

**package** test\_proj;

**public** **class** Tree {

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

**try**{

Class.*forName*("com.jspiders.denioapp.Asd");

System.out.println("class is loaded successfully");

}**catch**(ClassNotFoundException e)

e.printStackTrace();

}

**21-Oct-15:**

While writing JDBC code it is a possibility of checked Exception **1)ClassNotFoundException 2)SQLException**

**Java Data Base Connection:**

Is used to communicate and to perform DB operation from java program to the data base server

**DB**

**JDBC**

**Java**

**My-SQL**

JAR

**>Driver class**

**Data Base Software:**

* **Oracle**
* **My-SQL**
* **Post-Gre SQL**
* **MS-SQL server**
* **IBM DBL**
* **derby**

Class.*forName*("com.mysql.jdbc.Driver");//mysql- driver class

* To communicate java pro to database **server** we need to load the driver class
* Driver class is the JDBC driver
* JDBC driver is given by database provider
* Oracle gives oracle JDBC driver
* My-SQL gives My-SQL JDBC driver

**Build:**

* copy driver jar file into eclipse lib folder that is project its folder
* add the jar into build path

**Evening class:**

**Design Pattern:**

Design pattern is a optimize solution for the commonly re-occurring Design pattern is need.

* There are 240 + design are available

Example of design pattern singleton, factory, façade, Adapter, Proxy.

Front controller, application controller

Design pattern is the solution which can only way implement.

* Generally class is loaded in JVM memory when any of its member are called that is constructor methods or data member for the first time
* A class is loaded only once
* It is possible to load a class without calling any of the member of a class by using forName();
* ForName() is a static method present in java.long.class
* The forName() method takes fully qualified class name as an argument

Syntaxes;

+ static Class forName(“String funcn”);

Throws ClassNotFoundException

* If the class is not present then the method throws a checked exception called ClassNotFound exception
* It is possible to load a class which is the part of jar files

**package** test\_proj;

**public** **class** Tree {

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

**try**{

Class.*forName*("com.jspiders.denioapp.Asd");

System.out.println("class is loaded successfully");

}**catch**(ClassNotFoundException e)

e.printStackTrace();

}

**Factory Design Pattern:**

* A factory produces multiple objects of same category and return them. That is factory deals with object creation.
* Factory takes an input which defines the type and returns appropriate object
* If the wrong input is given in the factory then its returns null
* A factory generally takes string input and hushed appropriate objects

string

Pen

input

extends

new skt

new marker

Ink

Marker

Sketch

pen

output

+ class PenFactory{

+ static Pen getPen(String type){

Pen p=null;

If(type.equals(“Sketch”)){

p=new SketchPen();

}

Return p;

}

}

Factory return type can be an interface can be abstract class or can be any super class

22-Oct-15:

**Properties files:**

1. Properties is a type of data which source the data in the form of key and value
2. The keys must always be unique but the values can be duplicate
3. The important feature of property files are it is platform independent that is technology independent.
4. To read the data somewhere property files we have to use

java.IO.FileReader

java.util.properites

1. While reading the data from a property files, we may possible get a checked exception called java.io.FileNotFoundException

Java.io.IOException.

1. Since file reader is a costly resource we have to ideally close it within the finally block
2. To fetch the data from a property files we have to overloaded methods
3. + String getProperty(String key);

In this method the key is taken as an argument

If the key is present then the method return associated value

If the key is not present then the method returns null, but not any error or any exception

1. +String getProperty(String key, String msg);

If the key present then the method returns associated value

If the key is not pre then the method returns msg

Note:

**Argument:** Data pass to method as known as argument

**Parameter:** Parameter is the one which hold the argument

**package** com.shishira.properityApp;

**import** java.io.\*;

**import** java.util.Properties;

**public** **class** PropertyReader {

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

Properties props=**new** Properties();

FileReader fr=**null**;

**try** {

fr=**new** FileReader("config/user.prperties");

props.load(fr);

String UserName=props.getProperty("user");

String Password=props.getProperty("pass");

System.*out*.println(UserName+" "+Password);

String User=props.getProperty("user", "key not found");

System.*out*.println(User);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**finally**{

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

**try** {

fr.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

}

24-Oct-15:

**Costly Resource:**

* Costly resources is something resources which consume system resources or the network resources costly resources must be closed after the used, other ways it’s reduce the performance of the application.
* Close is the methods which close the costly resources.

**package** com.shishira.properityApp;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.Properties;

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

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

FileReader fr=**null**;

**try** {

//hot vessel

fr=**new** FileReader("config/user.properties");

Properties props=**new** Properties();

props.load(fr);//cup

String UserName=props.getProperty("user");

String Password=props.getProperty("pass");

System.*out*.println(UserName);

System.*out*.println(Password);

} **catch** (IOException e) {

e.printStackTrace();

}

**Finally**{

**try** {

fr.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

o/p:

Azam

Theone

**26-oct-15:**

Note:

* While closing the costly resource checked for null condition to avoid null pointer exception.

**27-Oct-15:**

**Evening batch class Note:**

**Factory Design Pattern example**

**package** com.shishira.properityApp;

**public** **class** Watch {

**public** **void** displayTime(){

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

}

}

**package** com.shishira.properityApp;

**public** **class** BondWatch **extends** Watch{

**public** **void** displayTime(){

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

}

}

**public** **class** AnalogWatch **extends** Watch{

**public** **void** displayTime(){

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

}

}

**package** com.shishira.properityApp;

**public** **class** SmartWatch **extends** Watch{

**public** **void** displayTime(){

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

}

}

**package** com.shishira.properityApp;

**public** **class** WatchFactory {

**public** **static** Watch createWatch(String type) {

//Object creation Logic

Watch w=**null**;

**if**(type.equalsIgnoreCase("analog")){

w=**new** AnalogWatch();

}**else** **if**(type.equalsIgnoreCase("bond")){

w=**new** BondWatch();

}**else** **if**(type.equalsIgnoreCase("smart")){

w=**new** SmartWatch();

}

**return** w;

}

}

**package** com.shishira.properityApp;

**import** java.util.Scanner;

**public** **class** FactoryTest {

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

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

System.*out*.println("Enter watch type ");

String type=sc.next();

sc.close();

Watch w=WatchFactory.*createWatch*(type);

w.displayTime();

}

}

**27-Oct-15:**

**XML File:**

* Xml stands for extensible markup language which is used to store and transport the data
* Xml is a case sensitive language and strictly define language.
* In xml we can define only user define text
* The important feature are platform independent and technology independent
* Since xml is user define language, we have to follow the rules to define the custom tags
* The rules are as follows:

1. What is the root tags
2. What are the sub tags and what are the values that can be assign for the sub tag must be known
3. All this rules are define either in an XSD or DTD

* XSD stands for Xml Schema Definition and DTD stands for Document Type Definition.
* An xml file which follows all this rules if an valid xml
* Xml in case of programing languages is use to configure the resources.
* In general xml is used to configure the user data.
* **Xml Parscr**: Xml Parscr is used to read or write the data from an xml file
* While read an xml file, if there are any error then its throws xmlPerscrException

Example DOM=document object model

Xml is also known as deployment

**29-Oct-15:**

**API (**Application Programing Interface**):**

* **JDBC** is an API which gives many data services using which we can communicate to any of the database server in a loosely coupled manner.
* Whenwe changed one database server to another we do not need to change our java program hence we can achieved loos coupling between java program and DB server to throw JDBC API

DB

A

P

I

J

My-SQL

* **JDBC** is an Java base API it’s all in the form of jar file
* An API may have many services or implementations which are exposing to the consumers throw interfaces.
* Consumer has to consume the services of an API throw interfaces

**30-Oct-15:**

JDBC API JDBC driver db vender

Driver

Connection

Statements

ResultSet

DriverManager

O

R

C

Oracle

Jdbc

Imple

Dev/prog I

.jar

Java

My

My-sql

Imple

HC .jar

Darbi

Darbi

Imple

(Java.sql and javaxsql)🡪rt.jar .jar

* Java program can access the data services expose by JDBC API through **Interface** and **Helper** classes
* JDBC has two components

1. JDBC API 2) JDBC framework

* JDBC API contents many interfaces and helper classes which has distributed into two packages

1. Java.sql and 2) javax.sql

Most of the features of JDBC are present in the java.sql and few are in javax.sql.

* We write java program by consuming JDBC API.

**JDBC Driver:**

* JDBC driver is an implementation of JDBC API.
* JDBC driver is given by different DB (Data Base) Vendors and it is specific to the database server that is every database has its own JDBC driver. Like oracle have oracle JDBC driver.
* JDBC driver are given in the form of jar file.

**Important Note:**

To write JDBC program we need both JDBC API as well as driver.

JAVA PROGRAM

**JDBC**

**ARCHITECH**:

Java –db calls Implements java-db calls

**JDBC-API**

My-

SQL driver

Oracle

JDBC driver

Native-db calls Native-db calls

MY-SQL DB

ORACLE DB

**04-Nov-15:**

**JDBC steps:**

1. Load and register driver
2. Establish the connection with DB server
3. Create the platform /statement
4. Execute SQL statement/query
5. Process the resulted data (optional)
6. Close all JDBC resources.
7. **Load and Register Driver:**

* All the driver classes are given by different DB venders and they are the part of JDBC driver
* The driver classes given by different vender which implements java.sql.Driver interface, which is the part of API.

Ex, public class OracleDriver implements java.sql.Driver{

----

}

public class Driver implements java.sql.Driver{

----

}

**JDBC API**

**I**

java.sql.Driver

implements implements

Driver

My-SQL JDBC Driver

Oracle Driver

Oracle JDBC Driver

**Driver classes by Venders:**

**Oracle:🡪**oracle jdbc.driver.OracleDriver

**My-SQL:🡪**com.mysql.jdbc.Driver

**Derby:🡪**org.apache.derby.jdbc.EmbadedDriver

**MS-SQL Server:🡪**com.mycrosoft.sqlserver.jdbc.SqlServerDriver

* Before performing any DB operation with the DB server, it is important to load and register the driver
* The driver object must be register with DriverManager using a static method.

Public static void registerDriver(pass the obj of Driver Class)

There are two ways of loading and register the driver:

1. Creating an object of driver class and registering the object with DriverManager.

Ex, Oracle:

OracleDriver od=new Oracle();

DriverManager.registerDriver(od);

Ex, My-SQL:

Driver d=new Driver();

DriverManager.registerDriver(d);

This type of code creates tight coupling, between java program and database server, hence we need to use another way i.e. by using Class.forName().

Oracle JDBC driver

JDBC API

oracle.jdbc.driver

Oracle Driver

DB

Driver

Java.sql

Oracle

I

Java

.jar

My-SQL JDBC Driver

HC

DriverManager

+ s v registerDriver(Driver d)

com.mysql.jdbc

Driver

DB

My-SQL

.jar

1. Class.forName()

Ex, Oracle:

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

My-SQL:

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

**JDBC Specification:**

Steps for specification (to be followed by venders)

1. Driver class (implementation class) should implement java.sql.Driver interface for My-SQL and oracleDriver for Oracle.
2. Inside this implementation class there should be a static block. (Loaded only once as classes is loaded only time).
3. This class object should be registered inside DriverManager class using register Driver Method and an object of same class should be passed.

* Fully qualified class name of driver is com.mysql.jdbc.Driver example of My-SQL.

Example of 1st way Loading and Registering the Driver of My-SQL:

+ class Driver implements java.sql.Driver{

static{

Driver d=new Driver();

DriverManager.registerDriver(d);

}

}

Example of 2nd way Loading and Registering the Driver:

1. Driver class and driver loaded Class.forName() we have to pass the fully qualified driver class as an argument.
2. This line possibly throw checked exception ClassNotFoundExceptoin.

Example

public class Demo{

public static void main(String [] args){

//load and register the driver

try{

//for Oracle

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

System.out.println(“oracle driver loaded”);

}

catch(ClassNotFoundExceptoin){

e.printStackTrace();

System.out.println(“oracle driver not loaded”);

}

}

}

**Connecting the database to the project after installing My-SQL Server:**

package.org.shishira.basicApp;

class Test{

public static void main(String [] args){

try{

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

System.out.println(“Driver Loaded”);

}

catch(ClassNotFoundException e){

e.printStackTrace();

}

}

}

09-Nov-15:

Database server

Request

Client

Response

DB Server name:

* Oracle
* My-sql
* Derby

Note: There should be client software which should be installed in client machine to access the server. Example of DB server My-SQL and client software is **SQL-Yog**.

**Web Server:**

* Apache-Tomcat
* JBoss
* WebLogic
* WebSpere

**Client:**

1. **Web-client:(**browser**)**
2. **DB client:**
3. SQL-yog
4. Toad
5. Sqirrel
6. **Mobile client:(**WhatsApp**)**

**Services.msc in run command to see running services.**

**URL (uniform resource locator):**

* **URL** is an address or a path to identify any resource unique way over the web.
* **URL** generally contents all the host information.

Local host:

* jdbc:mysql://localhost:3606
* jdbc:mysql://localhost:3606/jsppiders
* jdbc:mysql://localhost:3606/jspiders?user=root&password=dinga

Remote Host:

* jdbc:mysql://192.168.25.1:3606
* jdbc:mysql:// 192.168.25.1:3606/jsppiders
* jdbc:mysql:// 192.168.25.1:3606/jspiders?user=root&password=dinga

**10-Nov-15:**

1. **Establish the connection with DB server:**

**Connection:**

Syntax,

Connection con=DriverManager.getConnection(“url”);// oracle url/ mysql url

**There are 3 overloaded getConnection() method:**

1. public static Connection getConnection(“url”);

throws SQLExceptoin

Ex, getConnection(“jdbc:mysql://localhost:3606?user=root&password=dinga”);

**package** jdbcDemo;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.util.Properties;

**public** **class** JdbcDemo {

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

**try** {

//load and register the driver

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

System.*out*.println("Driver loaded");

//established the connection with DB server

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306?user=root&password=tiger");

System.*out*.println("Connection Successfull");

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

e.printStackTrace();

}

}

}

1. public static Connection getConnection(String url, String username, String passwrd)

throws SQLExceptoin

Ex, getConnection(“jdbc:mysql://localhost:3606”,”root” ,”dinga”);

**package** jdbcDemo;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.util.Properties;

**public** **class** JdbcDemo3 {

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

Properties props=**new** Properties();

**try** {

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

System.*out*.println("Driver loaded");

FileReader fr=**new** FileReader("config\\jdbc1.properties");

props.load(fr);

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

System.*out*.println("Connection Successfull");

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

e.printStackTrace();

}

}

}

1. public static Connection getConnection(String url, Properties props)

throws SQLExceptoin

Ex, Properties props=new Properties();

FileReader fr=new FileReaer(“dbconfig.properties”);

props.load(fr);

getConnection(jdbc:mysql://localhost:3606”,props);

**package** jdbcDemo;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.util.Properties;

**public** **class** JdbcDemo2 {

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

Properties props=**new** Properties();

**try** {

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

System.*out*.println("Driver loaded");

FileReader fr=**new** FileReader("config\\jdbc1.properties");

props.load(fr);

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

System.*out*.println("Connection Successfull");

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

e.printStackTrace();

}

}

}

* After loading and registering the driver manager , we can established the connection
* Get connation is a static method which is present java.sql.DriverManager class which helps to establish the connection.
* The getConnection() method takes string url and based on the url its connection to an appropriate server.
* If the url is wrong then method throws SQLException
* getConnection() is pure implementation of factory design patter
* Connection is an interface which is the path of jdbc api for which implementation is given by different DB vender as the part of jdbc driver
* The getConnetion method returns null at throws SQlExc if the url is wrong.

API driver DB vendr

I

Connection

java

orcl

H-C

DriverManager

+ s Conn getCon(url){

}

My-s

**11-Nov-15:**

**Stored Procedure:**

* Stored procedure is a SQL subroutine which is created, compiled and stored in the data base.
* We get the stored procedure in the ready to execute format.
* Stored procedure can have multiple SQL statements.
* A procedure can take any no of inputs(in parameter) and procedure can return any no of output(out parameter)

**Advantage of Stored Procedure:**

* Faster in performance
* Helps in separation or decoupling SQL logic from java logic
* Can write multiple query’s in single unite
* A procedure can return multiple results set

1. input(in para) 2) output(out para)

procName(para1, para2……)

begin

op1

op2

end

Syntax for writing the parameter:

[Type][Name][Data type][Length]

Ex, IN USN Varchar(10)

OUT Per decimal(4.2)

**Ex,Create procedure:**

Create procedure ‘getEmpSal’(IN id int(4) , OUT salary decimal(8,2))

Begin

Select sal into salary

From jspider.emp

Where eid=id;

end

**in sql yog:**

DELIMITER $$;

DROP PROCEDURE IF EXISTS `jspider`.`getEmpSal`$$

CREATE PROCEDURE `jspider`.`getEmpSal` (IN id int(4) , OUT salary decimal(8,2))

BEGIN

select sal into salary

from jspider.emp

where empid=id;

END$$

DELIMITER ;$$

**Example of inserting data:**

DELIMITER $$;

DROP PROCEDURE IF EXISTS `jspider`.`inserData`$$

CREATE PROCEDURE `jspider`.`inserData` (in emid int(10),in esal double(8,2), in emname varchar(10), in edept varchar(10))

BEGIN

insert into jspider.emp (eid,sal,ename,dept) values(emid,esal,ename,edept)

END$$

DELIMITER ;$$

**Procedure to result set:**

DELIMITER $$;

DROP PROCEDURE IF EXISTS `jspider`.`getEmpDetails`$$

CREATE PROCEDURE `jspider`,`getEmpDetails` (IN id int(4))

BEGIN

select\*

from jspider.emp

where eid=id;

END$$

DELIMITER ;$$

**3. Create the platform /statement:**

* To execute SQL statement and query we need to create platform so that on the created platform we can execute different types of SQL statements.
* Statements create a platform for execution of any types of SQL queries like DDL, DML, TCL, etc.
* Whenever a query is executed through statements then that query is compiled and then it is executed. That is the query is compiling and executed every time. Hence it must be used for execution of query’s which are not repeated.

Statement stmnt=con.createStatement();

There are three type of statement on JDBC:

1. Statement
2. PreparedStatement
3. CallableStatement

All the above statements are interfaces which are the part of JDBC API and implementation are given by different DB venders has the part of JDBC driver.

**4. Execute SQL Statement/Query:**

* We can execute the statement of SQL query using below methods:

1. Public Boolean **execute**(String any Query) throws SQLException
2. + int **executeUpdate**(String DML Query) throws SQLException
3. + ResultSet **executeQuery**(String DQL Query) throws SQLException

* The outcome of executUpdate() of DML query is an integer value, which indicates the no of record affected that is inserted or deleted or updated and we can get that count by using below method

+ int getUpadateCount();

**package** jdbcDemo;

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

**public** **class** DeletEmpDemo {

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

String delQry="delet from jspider.emp where dept='QA'";

**try** {

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

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

Statement stmt=con.createStatement();

stmt.execute(delQry);

**int** nord=stmt.getUpdateCount();

System.*out*.println(nord+" employ deleted");

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

e.printStackTrace();

}

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

}

}

18-Nov-15:

**Prepared Statement:**

* To execute similar repeated queries if we use statement then it compile and then execute, that is every query is compiled every time hence it takes more time. Hence the performance will be low.

Note: we used PreparedStatements for execution of similar multiple repeated queries.

Compile + Execute

Compile + Execute

Statement stmt=con.createStatement();

Q1=insert into jsp.emp values(9, ‘shi’,’mgmt’, 10000)

Q1=insert into jsp.emp values(9, ‘khi,’mgmt’, 12000)

Q1=insert into jsp.emp values(9, ‘mhi,qa, 400000)

Gdq=qn=insert into jsp.emp values(? , ? , ? , ?)

* PreparedStatement is an interface which is the part of JDBC API present in java.sql package and implementation is given by different DB vender as the part of JDBC driver.
* We pass generic dynamic query at the time of creation of Prepared Statement Object

PreparedStatement psmt=con.prepareStatement(Gdq)

* Generic Dynamic Query accepts place holders
* PreparedStatement is pre compiled statement
* PreparedStatement support the concept of compile once and execute multiple times.
* PreparedStatement gives the batter performance while executing similar multiple queries.

**Place Holder:**

1. place holder is like a variable with accept dynamic values
2. A query can have any no of place holder
3. We have to set the data for the place holder before the execution.
4. The no of place holder add the no of data must be exact match.
5. The data can be set to place holder using setXXX() method.

insert into jsp.emp values(? , ? , ? , ?)

select\* from student where stream=?

Update employee

Set salary =?

Where eid=?

Delet from product where pid=?

**1 2**

**Update employee set salary=? Where eid=?**

**1**

**delete from product where pid=?**

**1 2 3 4**

insert into jsp.emp values(? , ? , ? , ?)

|  |  |  |  |
| --- | --- | --- | --- |
| **Eid** | **Ename** | **Dept** | **sal** |

**1 2**

**Insert into jsp.emp(eid, sal) values(?, ?)**

**20-Nov-15:**

**Batch Updates:**

Every independent DB operation is considered as costly operation hence instead of executing independent DB operation it’s better to add a DML operation into a batch and then execute the batch.

B

A

T

C

H

DB Server

Java

* Batch update is applicable only for DML statements
* Using batch update we can greatly improve the performance
* A batch is considered as set of or collection of multiple DML statements
* Batch update can be executed by using statement as well as PreparedStatements.
* We can add all the SQL operation using below method:

+ void addBatch(String DML query)

* We can execute the batch below method:

+ int [] arr= executeBatch()

* Once we execute the batch, we get an integer array

1. The size of the array depends on the no of query added to the batch
2. The order of an array depends on the order in which the query are added to the batch
3. Every element is an integer value which indicates the outcome of execution of DML statements which work added to the batch.

**23-Nov-15:**

**JDBC Transaction:**

* A transaction consider is set of SQL operation which has to be executed together as a single unite whether in if any one operation fail than all the executed operation revert back, executed all or executed none.
* In JDBC transaction can be handle using below method

setAutoCommit()

commit()

rollBack()

rollBack(SavePoint)

* Afterestablishing the connection with DB server the auto commit mode is enabled, hence any changes made to the data is saved permanently in the data base
* Wecan explicitly disable the auto commit mode, but in this case after the execution we have to save the changes made

Connection con=DriverManager.getConnection(url);

con.setAutoCommit(false);

Statement stmt=con.createStatement();

Stmt.execute(qry);

con.commit();//save

**Try{**

Start con.setAutocommit(false);

Op1

Op2

Op2

End con.commit();

}catch(sqlExceptoin sqls){

con.rollBack();

}

**package** jdbcDemo;

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

**public** **class** TransactionDemo {

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

System.*out*.println("App Start");

Connection con=**null**;

Statement stmt=**null**;

String insertQryEdu="insert into jspider.student\_edu\_info values(2, 68,'cs'";

String insertQryPer="insert into jspider.student\_edu\_info values(2, 'raju','ljkhdfjkhjkdhfsjkdfhkjsdhfkj'";

**try** {

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

con=DriverManager.*getConnection*(url);

//transaction begin

con.setAutoCommit(**false**);

stmt=con.createStatement();

//first operation

stmt.execute(insertQryEdu);

System.*out*.println("op1 success");

//Second operation

stmt.execute(insertQryPer);

System.*out*.println("op2 success");

//transaction end

con.commit();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}**catch** (SQLException e){

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

**try**{

con.rollback();//importatn

}**catch**(SQLException el){

el.printStackTrace();

}**finally**{

**try**{

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

stmt.close();

}**if**(con!=**null**){

con.close();

}

}**catch**(SQLException eld){

eld.printStackTrace();

}

}

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

}

}

}

**24-Nov-15:**

**SavePoint:**

* SavePoint is an interface present in java.sql package and implementation is provided by different DB venders has the part of JDBC driver
* SavePoint is used to implement transaction feature of database

**SavePoint sp=null;**

**try{ con.setAutocommit(false);**

**sql-op-1**

**sql-op-2**

**sql-op-3**

**sp=con.setSavePoint(“after 3”)**

**sql-op-4**

**sql-op-5**

**con.commit();**

**}catch(SQLException){**

**If(sp!=null){**

**con.rollBack(sp);**

**con.commit();**

**}else{**

**con.rollBack();**

**}**

**}**

**Interview Question:**

1. **What are different types of statements?**
2. **Different between Statement and prepared Statement?**
3. **Explain JDBC Transaction?**
4. **Name the method to handle JDBC Transaction.**
5. **What is Stored Procedure and what are the Advantages?**
6. **Steps to execute the Store Procedure through callable statement?**
7. **Different between execute executeQuery and execcuteUpdate?**
8. **What are the Possible Exception that we get write JDBC code?**
9. **What is ResultSet and what are the methods of resultSet?**
10. **Which type of driver is used in your project or code(type 4)**
11. **What are the advantage of batch update**

**Improved performance by adding multiple DML operation into Batch**

**Types of Driver:**

There are 4 types of driver

1. Type-1 [JDBC-ODBC Bridge / Bridge]
2. Type-2 [Partly java Native API]
3. Type-3 [Net Protocol- full java / Middleware]
4. Type-4 [Native Protocol – Full java / PURE]

**Type-1 driver:**

JDBC

A

P

I

DB-Server

Type-1

Driver

ODBC

Driver

Java Pr

**Java-db odbc call native db**

**calls calls**

**ODBC: (open database connection)**

* ODBC is an driver written in native language
* Using ODBC we can’t achieved loose coupling between java application and DB server

**Type-2 driver:**

**Vender-API**

DB-Server

Type-2

Driver

JDBC

A

P

I

Java Pr

A

P

I

**Java-DB** native

**calls** DB calls

* **Written partly in java and partly native language**
* **Impt of JDBC and vender API**
* **DB specific**
* **Not much support available**

**Type-3 Driver:**

DB-Server

Type-3

Driver

JDBC

A

P

I

Java Pr

Net protocol

Or middle weare server

Java db native

**calls db calss**

written in 100% java **data filtering**

**DB** independents **performation optimization**

Cost involved

Good for Distrubuted app

**Type-4:**

DB-Server

Type-4

Driver

JDBC

A

P

I

Java Pr

**Native db calls**

**Meta-Data:**

* Data about the data is called Meta-Data

Example, Database name, Driver name, the no of column in the table, name of particular column etc.

There are two types of Meta-Data in JDBC;

1. DataBaseMetaData
2. ResultSetMetaData
3. **DataBaseMetaData:** this is gives you better information about the data base, like the database name, version, driver name etc.
4. **ResultSetMetaData:** This is gives you Meta information about the table which is associated with one result set.

**Ex,** No of column in the table, Name of the particular column etc.

* getMetaData() is the method which is used to get Meta-Data.

Ex, DataBaseMetaData dbmeta=con.getMetaData();

ResultSet rs=stmt.executeQuery(“Select\* from emp”);

ResultSetMetaData remeta=rs.getMetaData();

Ex,

**package** jdbcDemo;

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

**public** **class** MetaData {

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

Connection con=**null**;

Statement stmt=**null**;

ResultSet rs=**null**;

**try** {

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

con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306?user=root&password=tiger");

DatabaseMetaData dbmeta=con.getMetaData();

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

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

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

stmt=con.createStatement();

rs=stmt.executeQuery("Select\* from jspider.emp");

ResultSetMetaData rsmeta=rs.getMetaData();

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

System.*out*.println(rsmeta.getColumnName(3));

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

e.printStackTrace();

}

**finally**{

**try** {

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

con.close();

}**else** **if** (stmt!=**null**) {

stmt.close();

}**else** **if** (rs!=**null**) {

rs.close();

}

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

o/p:

MySQL

5.0.19-nt

MySQL-AB JDBC Driver

4

ename