**back1) JDBC In Simple Way ....................................................................................................**

**2) Storage Areas ..............................................................................................................**

**3) JDBC ........................................................................................................................ ...**

**4) JDBC Architecture .....................................................................................................**

**5) JDBC API ...................................................................................................................**

**6) Types of Drivers ........................................................................................................**

**7) Standard Steps For Developing JDBC Application ......................................................**

**8) Select Operations And Non-Select Operations ..........................................................**

**9) Programs On Database Operations ...........................................................................**

**10) Aggregate Functions ..................................................................................................**

**11) Real Time Coding Standards For JDBC Application .....................................................**

**12) Working With MySQL Database ................................................................................**

**13) Life Cycle of SQL Query Execution .............................................................................**

**14) PreparedStatement (I) ..............................................................................................**

**15) SQL Injection Attack ..................................................................................................**

**16) Stored Procedures and CallableStatement ................................................................**

**17) Cursors ......................................................................................................................**

**18) Functions ..................................................................................................................**

**19) Batch Updates .........................................................................................................**

**20) Handling Date Values For Database Operations .......................................................**

**21) Working with Large Objects (BLOB and CLOB) .........................................................**

**22) Connection Pooling .................................................................................................**

**23) Properties ...............................................................................................................**

**24) Transaction Management in JDBC ...........................................................................**

**25) MetaData ................................................................................................................**

**26) JDBC with Excel Sheets ............................................................................................**

**27) ResultSet Types .......................................................................................................**

**28) RowSets ..................................................................................................................**

**29) Top Most Important JDBC FAQ’s ……………………………………………………………………………**

**30) JDBC Interview FAQ’s …………………………………………………………………………………………….**

**ADVANCED JAVA**

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

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

**Eg: Calculator, MS Word**

**Any Core Java Application**

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

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

**Eg: elanzasoftvideos.com, gmail.com, facebook.com, elanzasoft.com**

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

◈ **JDBC**

◈ **Servlets**

◈ **JSP's**

**Where ever Presentation Logic is required i.e. to display something to the End User then we**

**should go for JSP i.e. JSP meant for View Component.**

**Eg: display login page**

**display inbox page**

**display error page**

**display result page**

**etc..**

**Where ever some Processing Logic is required then we should go for Servlet i.e. Servlet meant for**

**Processing Logic/ Business Logic. Servlet will always work internally.**

**Eg: Verify User**

**Communicate with Database**

**Process End User's Data**

**etc..**

**From Java Application (Normal Java Class OR Servlet) if we want to communicate with Database**

**then we should go for JDBC.**

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

**To get Mails Information from Database**

**In Java there are 3 Editions are available**

**1. Java Standard Edition (JSE | J2SE)**

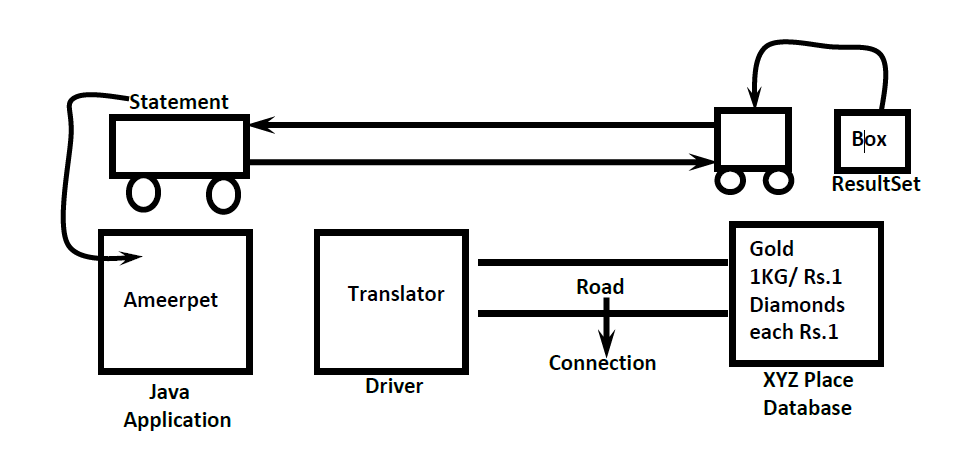
**2. Java Enterprise Edition (JEE | J2EE)**

**3. Java Micro Edition (JME | J2ME)**

**JDBC is the Part of JSE**

**Servlets and JSP's are the Part of JEE**

**JDBC in Simple Way**

****

**Driver (Translator):**

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

**Connection (Road):**

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

**Statement (Vehicle):**

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

**from Database.**

**ResultSet:**

**ResultSet holds Results of SQL Query.**

**Steps for JDBC Application:**

**1. Load and Register Driver**

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

**3. Create Statement Object**

**4. Send and Execute SQL Query**

**5. Process Results from ResultSet**

**6. Close Connection**

**Demo Program:**

**1) import java.sql.\*;**

**2) public class JdbcDemo**

**3) {**

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

**5) {**

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

**7) Connection con=DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");**

**8) Statement st = con.createStatement();**

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

**10) while(rs.next())**

**11) {**

**12) System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+"..."+rs.get**

**String(4));**

**13) }**

**14) con.close();**

**15) }**

**16) }**

**Storage Areas**

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

**Information, Calls Information etc..**

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

**1) Temporary Storage Areas**

**2) Permanent Storage Areas**

**Temporary Storage Areas:**

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

**Eg: All JVM Memory Areas (like Heap Area, Method Area, Stack Area etc).**

**Once JVM shutdown all these Memory Areas will be cleared automatically.**

**Permanent Storage Areas:**

**Also known as Persistent Storage Areas.**

**Here we can store Data permanently.**

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

**File Systems:**

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

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

**Limitations:**

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

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

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

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

**Inconsistency Problems.**

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

**Databases:**

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

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

**Operations very easily.**

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

**Hence Data is secured.**

**4) Inside Database Data will be stored in the form of Tables. While developing Database Table**

**Schemas, Database Admin follow various Normalization Techniques and can implement**

**various Constraints like Unique Key Constrains, Primary Key Constraints etc which prevent**

**Data Duplication. Hence there is no chance of Data Inconsistency Problems.**

**Limitations of Databases:**

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

**2) Database can provide support only for Structured Data (Tabular Data OR Relational Data) and**

**cannot provide support for Semi Structured Data (like XML Files) and Unstructured Data (like**

**Video Files, Audio Files, Images etc)**

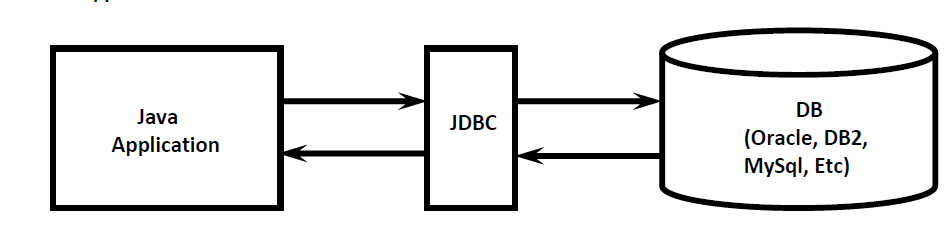
**To overcome this Problems we should go for more Advanced Storage Areas like Big Data**

**Technologies, Data warehouses etc..**

**JDBC**

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

**Application.**

****

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

 **JDBC is a Specification defined by Java Vendor (Sun Micro Systems) and implemented by**

**Database Vendors.**

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

**JDBC Features:**

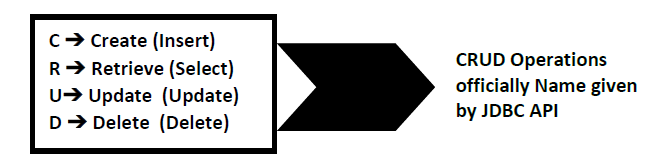
**1) JDBC API is Standard API. We can communicate with any Database without rewriting our**

**Application i.e. it is Database Independent API.**

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

**JDBC Is Platform Independent Technology.**

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

****

**We can also perform Complex Operations (like Inner Joins, Outer Joins, calling Stored Procedures**

**etc) very easily by using JDBC API.**

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

**on JDBC API.**

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

***http://www.oracle.com/technetwork/java/index-136695.html***

**JDBC Versions:**

 **JDBC 3.0 is Part J2SE 1.4**

 **No Update in Java SE 5.0**

 **JDBC 4.0 is Part Java SE 6.0**

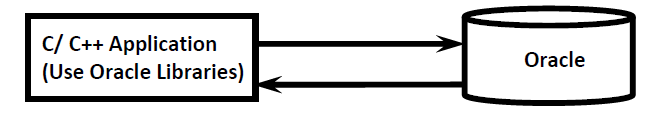
 **JDBC 4.1 is Part Java SE 7.0**

 **JDBC 4.2 is Part Java SE 8.0**

**Evolution of JDBC:**

 **If we want to communicate with Database by using C OR C++, compulsory we have to use**

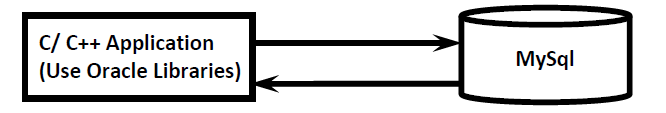
**database specific Libraries in our Application directly.**

****

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

 **The Problem in this Approach is, if we want to migrate Database to another Database then we**

**have to rewrite Total Application once again by using new Database specific Libraries.**

****

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

 **To overcome this Problem, Microsoft People introduced "ODBC" Concept in 1992. It is**

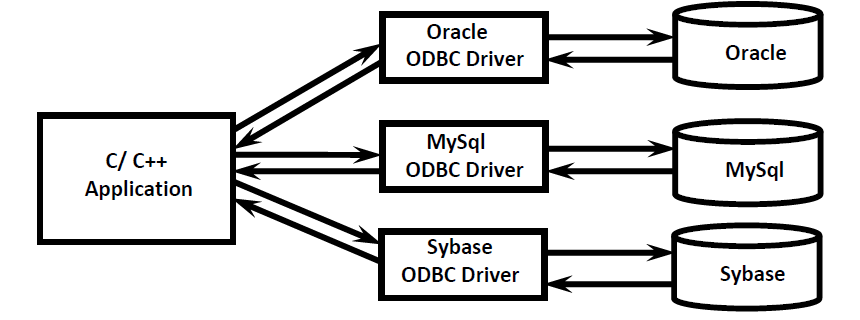
**Database Independent API.**

 **With ODBC API, Application can communicate with any Database just by selecting**

**corresponding ODBC Driver.**

 **We are not required to use any Database specific Libraries in our Application. Hence our**

**Application will become Database Independent.**

****



**Limitations of ODBC:**

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

**2) ODBC Drivers are implemented in C Language. If we use ODBC for Java Applications, then**

**Performance will be down because of internal conversions from Java to C and C to Java.**

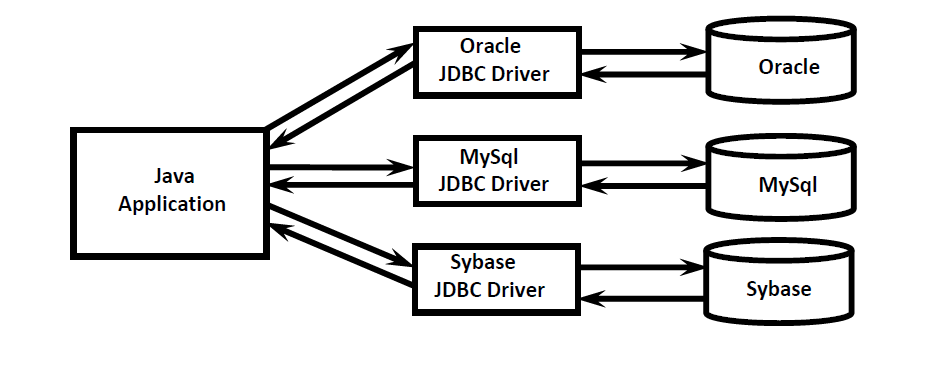
 **Because of above Reasons, ODBC Concept is not suitable for Java Applications.**

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

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

 **JDBC Drivers are implemented in Java. If we use JDBC for Java Applications, then internal**

**Conversions are not required and hence there is no Effect on Performance.**

****

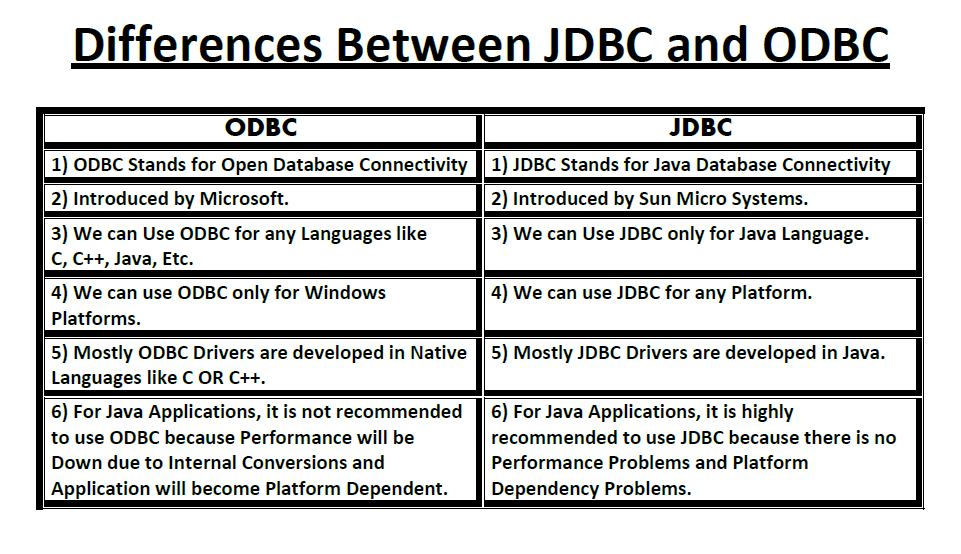
**\*\*\*Note:**

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

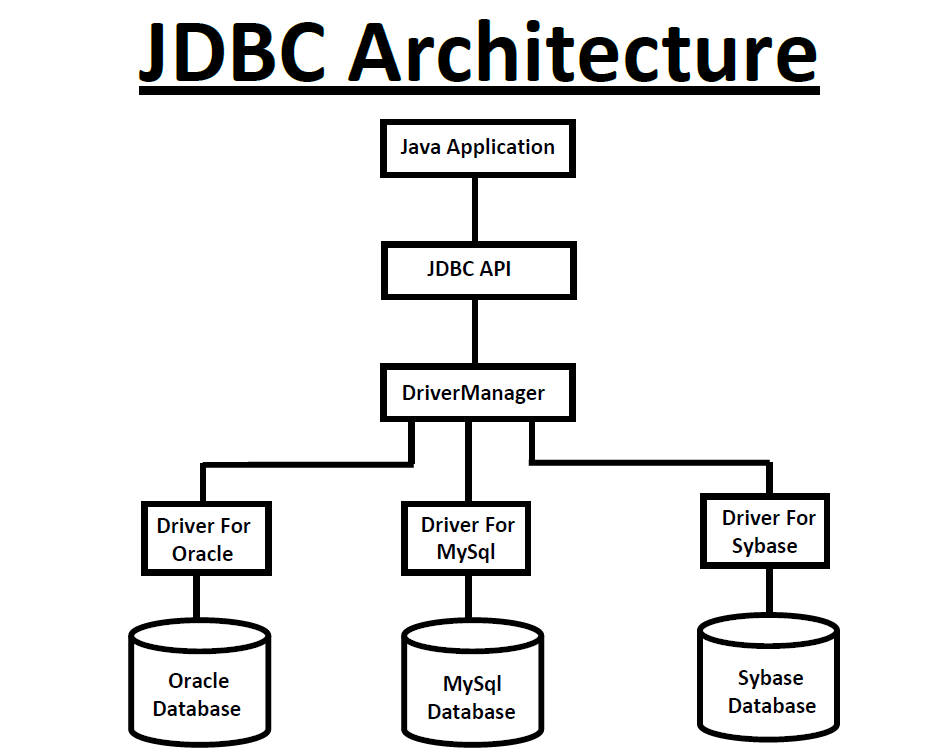
**Platform.**

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

**Java**

****



****

 **JDBC API provides DriverManager to our Java Application.**

 **Java Application can communicate with any Database with the help of DriverManager and**

**Database Specific Driver.**

**DriverManager:**

* **It is the Key Component in JDBC Architecture.**
* **DriverManager is a Java Class present in *java.sql* Package.**
* **It is responsible to manage all Database Drivers available in our System.**
* **DriverManager is responsible to register and unregister Database Drivers.**

**DriverManager.registerDriver(Driver);**

**DriverManager.unregisterDriver(Driver);**

* **DriverManager is responsible to establish Connection to the Database with the help of Driver Software.**

**Connection con = DriverManager.getConnection (jdbcurl, username, pwd);**

**Database Driver:**

 **It is the very Important Component of JDBC Architecture.**

 **Without Driver Software we cannot touch Database.**

 **It acts as Bridge between Java Application and Database.**

 **It is responsible to convert Java Calls into Database specific Calls and Database specific Calls**

**into Java Calls.**

**Note:**

**1) Java Application is Database Independent but Driver Software is Database Dependent.**

**Because of Driver Software only Java Application will become Database Independent.**

**2) Java Application is Platform Independent but JVM is Platform Dependent. Because of JVM only**

**Java Application will become Platform Independent.**

**JDBC API**

 **JDBC API provides several Classes and Interfaces.**

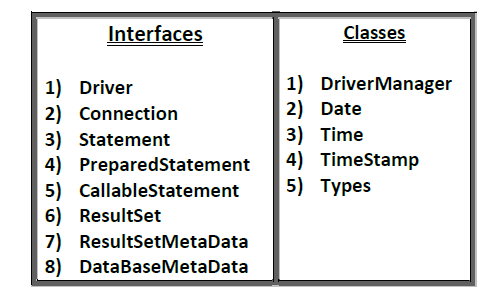
 **Programmer can use these Classes and Interfaces to communicate with the Database.**

 **Driver Software Vendor can use JDBC API while developing Driver Software.**

 **JDBC API defines 2 Packages**

**1) java.sql Package:**

**It contains basic Classes and Interfaces which can be used for Database Communication.**

****

**2) javax.sql Package:**

**It defines more advanced Classes and Interfaces which can be used for Database**

**Communication.**

**There are multiple Sub Packages are also available**

* **javax.sql.rowset;**
* **javax.sql.rowset.serial;**
* **javax.sql.rowset.spi;**



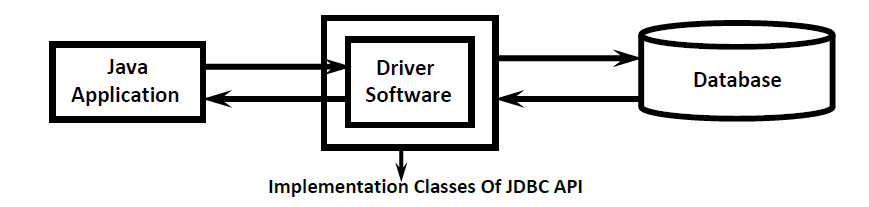
 **Programmers are not responsible to provide Implementation for JDBC API Interfaces.**

 **Most of the times Database Vendor is responsible to provide Implementation as the Part of**

**Driver Software.**

 **Every Driver Software is a Collection of Classes implementing various Interfaces of JDBC API,**

**which can be used to communicate with a particular Database.**

****

 **For Example, Driver Software of Oracle means Collection of Implementation Classes of JDBC**

**API, which can be used to communicate with Oracle Database.**

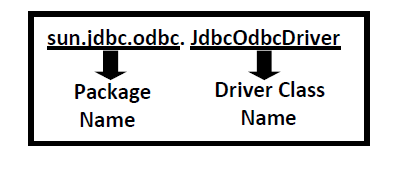
 **Every Driver Software is identified with some Special Class which is nothing but Driver Class. It**

**is the Implementation Class of Driver Interface present in. *java.sql* Package.**

 **As the Part of JDK, SUN People provided one Built-In Driver Software which implements JDBC**

**API, which is nothing but Type-1 Driver (JDBC-ODBC Bridge Driver).**

 **The corresponding Driver Class Name is:**

****

**Difference between Driver Interface, Driver Class and Driver Software:**

**1) Driver Interface:**

**This Interface present in *java.sql* Package.**

**This Interface acts as Requirement Specification to implement Driver Class.**

**2) Driver Class:**

**It is the Implementation Class of Driver Interface**

**Eg: sun.jdbc.odbc.jdbcodbcdriver**



**3) Driver Software:**

* **It is the Collection of Implementation Classes of various Interfaces present in JDBC API.**
* **It acts as Bridge between Java Application and Database.**
* **It is responsible to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls.**
* **Usually Driver Softwares are available in the Form of jar File.**

**Eg:**

**ojdbc14.jar**

**ojdbc6.jar**

**ojdbc7.jar**

**mysql-connector.jar**

**Driver Softwares can be provided by the following Vendors**

*  **Java Vendor (Until 1.7 Version Only)**
*  **Database Vendor**
*  **Third Party Vendor**

**Type-1 Driver (JDBC-ODBC Bridge Driver) provided by Java Vendor.**

**Thin Driver provided by Oracle Database Vendor.**

**Inet is a Third Party Vendor and providing several Driver Softwares for different Databases.**

**Eg:**

*  **Inet Oraxo For Oracle Database**
*  **Inet Merlia For Microsoft SQL Server**
*  **Inet Sybelux For Sybase Database**

**Note: It is highly recommended to use Database Vendor provided Driver Softwares.**

* **While developing Driver Software, Vendors may use only Java OR Java with other Languages like C OR C++.**
* **If Driver Software is developed only in Java Language then such Type of Drivers are called Pure Java Drivers.**
* **If Driver Software developed with Java and other Languages, then such Type of Driver**

**Softwares are called Partial Java Drivers.**

**Types of Drivers**

**While communicating with Database, we have to convert Java Calls into Database specific Calls**

**and Database specific Calls into Java Calls. For this Driver Software is required. In the Market**

**Thousands of Driver Softwares are available. But based on Functionality all Driver Software Drivers**

**are divided into 4 Types.**

**1) Type-1 Driver (JDBC-ODBC Bridge Driver OR Bridge Driver)**

**2) Type-2 Driver (Native API-Partly Java Driver OR Native Driver)**

**3) Type-3 Driver (All Java Net Protocol Driver OR Network Protocol Driver OR Middleware Driver)**

**4) Type-4 Driver (All Java Native Protocol Driver OR Pure Java Driver OR Thin Driver)**

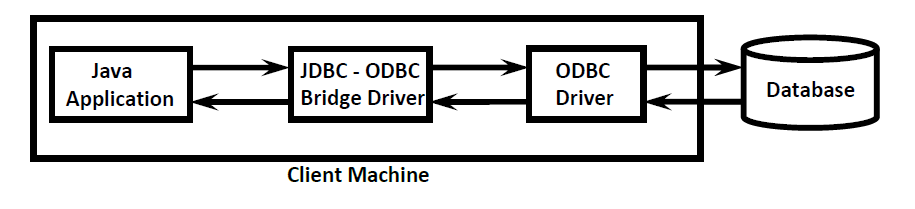
**Note:**

**Progress Data direct Software Company introduced Type-5 Driver, but it is not Industry Recognized**

**Driver.**

**Type-1 Driver:**

**Also known as *JDBC-ODBC Bridge Driver* OR *Bridge Driver.***

****

**This Driver provided by Sun Micro Systems as the Part of JDK. But this Support is available until 1.7**

**Version only.**

**Internally this Driver will take Support of ODBC Driver to communicate with Database.**

**Type-1 Driver converts JDBC Calls (Java Calls) into ODBC Calls and ODBC Driver converts ODBC Calls**

**into Database specific Calls.**

**Hence Type-1 Driver acts as Bridge between JDBC and ODBC.**

**Advantages :**

**1. It is very easy to use and maintain.**

**2. We are not required to install any separate Software because it is available as the Part of JDK.**

**3. Type-1 Driver won't communicates directly with the Database. Hence it is Database**

**Independent Driver. Because of this migrating from one Database to another Database will**

**become Easy.**

**Limitations:**

**1. It is the slowest Driver among all JDBC Drivers (Snail Driver), because first it will convert JDBC**

**Calls into ODBC Calls and ODBC Driver converts ODBC Calls into Database specific Calls.**

**2. This Driver internally depends on ODBC Driver, which will work only on Windows Machines.**

**Hence Type-1 Driver is Platform Dependent Driver.**

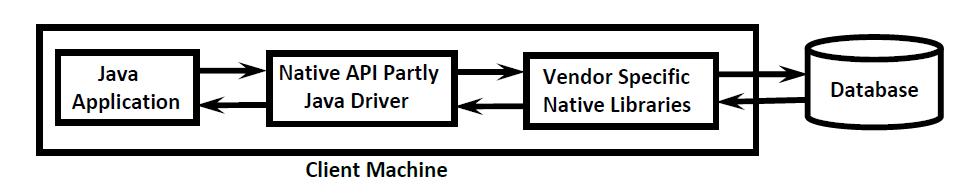
**3. No Support from JDK 1.8 Version onwards.**

**Note:**

**Because of above Limitations it is never recommended to use Type-1 Driver.**

**Type-2 Driver:**

**It is also known as *Native API -Partly Java Driver* OR *Native Driver.***

****

**Type-2 Driver is exactly same as Type-1 Driver except that ODBC Driver is replaced with Vendor**

**specific Native Libraries.**

**Type-2 Driver internally uses Vendor specific Native Libraries to Communicate with Database.**

**Native Libraries means the Set of Functions written in Non-Java (Mostly C OR C++).**

**We have to install Vendor provided Native Libraries on the Client Machine.**

**Type-2 Driver converts JDBC Calls into Vendor specific Native Library Calls, which can be**

**understandable directly by Database Engine.**

**Advantages:**

**1. When compared with Type-1 Driver Performance is High, because it required only one Level**

**Conversion from JDBC to Native Library Calls.**

**2. No need of arranging ODBC Drivers.**

**3. When compared with Type-1 Driver, Portability is more because Type-1 Driver is applicable only**

**for Windows Machines.**

**Limitations:**

**1. Internally this Driver using Database specific Native Libraries and hence it is Database**

**Dependent Driver. Because of this migrating from one Database to another Database will become**

**Difficult.**

**2. This Driver is Platform Dependent Driver.**

**3. On the Client Machine compulsory we should install Database specific Native Libraries.**

**4. There is no Guarantee for every Database Vendor will provide This Driver.**

**(Oracle People provided Type-2 Driver but MySql People won't provide this Driver)**

**Eg: OCI (Oracle Call Interface) Driver is Type-2 Driver provided by Oracle.**

**OCI Driver internally uses OCI Libraries to communicate with Database.**

**OCI Libraries contain "C Language Functions"**

**OCI Driver and corresponding OCI Libraries are available in the following Jar File. Hence we have to**

**place this Jar File in the Class Path.**

**ojdbc14.jar** ➔ **Oracle 10g (Internally Uses Java 1.4V)**

**ojdbc6.jar** ➔ **Oracle 11g (Internally Uses Java 1.6V)**

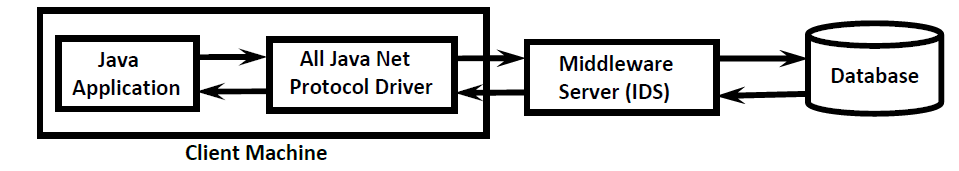
**ojdbc7.jar** ➔ **Oracle 12c (Internally Uses Java 1.7V)**

**Note: The only Driver which is both Platform Dependent and Database Dependent is Type-2**

**Driver. Hence it is not recommended to use Type-2 Driver.**

**Type-3 Driver:**

**Also known as *All Java Net Protocol Driver* OR *Network Protocol Driver* OR *Middleware Driver***



**Type-3 Driver converts JDBC Calls into Middleware Server specific Calls. Middleware Server can**

**convert Middleware Server specific Calls into Database specific Calls.**

**Internally Middleware Server may use Type-1, 2 OR 4 Drivers to communicates with Database.**

**Advantages:**

**1. This Driver won't communicate with Database directly and hence it is Database Independent**

**Driver.**

**2. This Driver is Platform Independent Driver.**

**3. No need of ODBC Driver OR Vendor specific Native Libraries**

**Limitations:**

**1. Because of having Middleware Server in the Middle, there may be a chance of Performance**

**Problems.**

**2. We need to purchase Middleware Server and hence the cost of this Driver is more when**

**compared with remaining Drivers.**

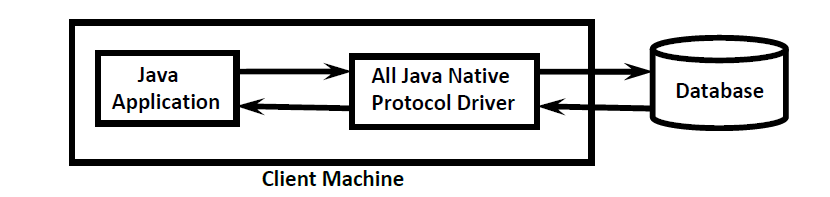
**Eg: IDS Driver (Internet Database Access Server)**

**Note: The only Driver which is both Platform Independent and Database Independent is Type-3**

**Driver. Hence it is recommended to use.**

**Type-4 Driver:**

**Also known as *Pure Java Driver* OR *Thin Driver.***

******

**This Driver is developed to talk with the Database directly without taking Support of *ODBC Driver***

**OR *Vendor Specific Native Libraries* OR *Middleware Server*.**

**This Driver uses Database specific Native Protocols to communicate with the Database.**

**This Driver converts JDBC Calls directly into Database specific Calls.**

**This Driver developed only in Java and hence it is also known as Pure Java Driver. Because of this,**

**Type-4 Driver is Platform Independent Driver.**

**This Driver won't require any Native Libraries at Client side and hence it is light weighted. Because**

**of this it is treated as Thin Driver.**

**Advantages:**

**1. It won't require any Native Libraries, *ODBC Driver* OR *Middleware Server***

**2. It is Platform Independent Driver**

**3. It uses Database Vendor specific Native Protocol and hence Security is more.**

**Limitation:**

**The only Limitation of this Driver is, it is Database Dependent Driver because it is communicating**

**with the Database directly.**

**Eg: Thin Driver for Oracle**

**Connector/J Driver for MySQL**

**Note: It is highly recommended to use Type-4 Driver.**

**Java Application** ➔ **Type-1 Driver** ➔ **ODBC Driver** ➔ **DB**

**Java Application** ➔ **Type-2 Driver** ➔ **Vendor Specific Native Libraries** ➔ **DB**

**Java Application** ➔ **Type-3 Driver** ➔ **Middleware Server** ➔ **DB**

**Java Application** ➔ **Type-4 Driver** ➔ **DB**

**Which Driver should be used?**

**1. If we are using only one Type of Database in our Application then it is recommended to use**

**Type-4 Driver.**

**Eg: Stand Alone Applications, Small Scale Web Applications**

**2. If we are using multiple Databases in our Application then Type-3 Driver is recommended to**

**use.**

**Eg: Large Scale Web Applications and Enterprise Applications**

**3. If Type-3 and Type-4 Drivers are not available then only we should go for Type-2 Driver.**

**4. If no other Driver is available then only we should go for Type-1 Driver.**

**Differences between *Thin* and *Thick* Driver:**

**If Driver won't require any extra Component to communicate with Database, such type of Driver is**

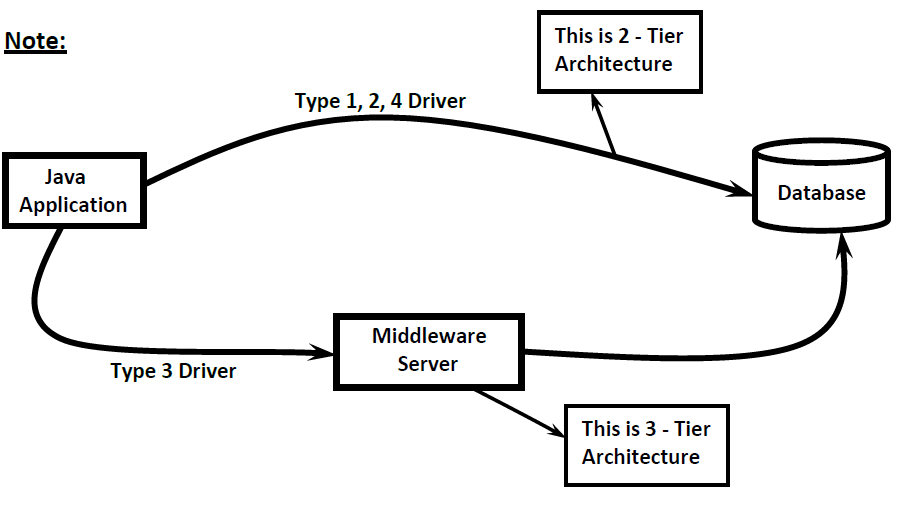
**called Thin Driver.**

**Eg: Type-4 Driver**

**If Driver require some extra Component (like *ODBC Driver* OR *Vendor specific Native Libraries***

**OR *Middleware Server*), such Type of Driver is called Thick Driver.**

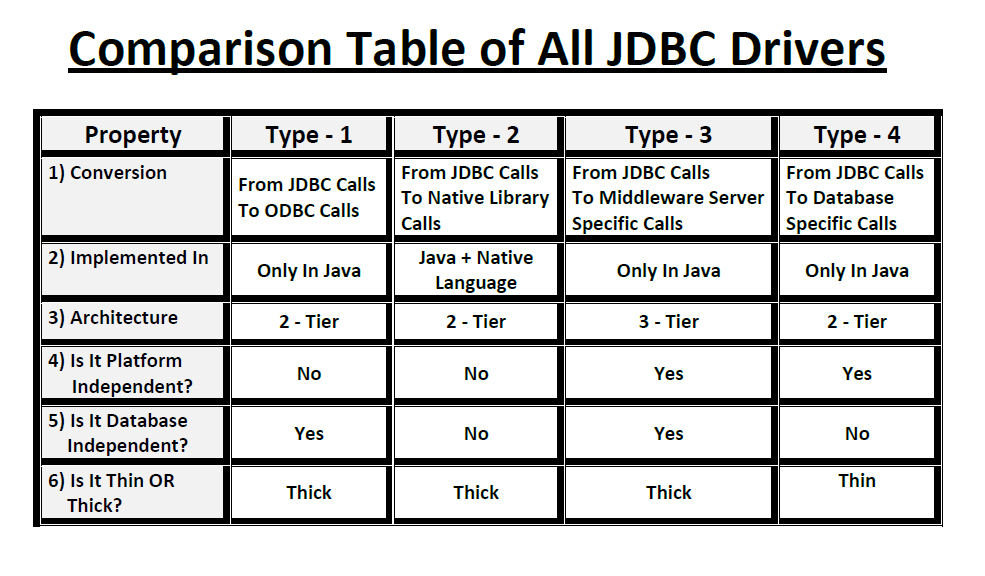
**Eg: Type-1, Type-2 and Type-3 Drivers**

****

**Note:**

**Type-1, Type-2 and Type-4 Drivers follow 2-Tier Architecture.**

**Type-3 Driver follows 3-Tier Architecture.**

****

**Standard Steps for developing JDBC Application**

**1. Load and register Driver Class**

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

**3. Create Statement Object**

**4. Send and execute SQL Query**

**5. Process Result from ResultSet**

**6. Close Connection**

**Step 1: Load and Register Driver Class**

**JDBC API is a Set of Interfaces defined by Java Vendor.**

**Database Vendor is responsible to provide Implementation. This Group of Implementation Classes**

**is nothing but "Driver Software".**

**We have to make this Driver Software available to our Java Program. For this we have to place**

**corresponding Jar File in the Class Path.**

**Note:**

**Type-1 Driver is available as the Part of JDK and hence we are not required to set any Class Path**

**explicitly.**

**Every Driver Software is identified by some special Class, which is nothing but Driver Class.**

**For Type-1 Driver, the corresponding Driver Class Name is**

****

**We can load any Java Class by using *Class.forName()* Method. Hence by using the same Method**

**we can load Driver Class.**

****

**Whenever we are loading Driver Class automatically Static Block present in that Driver Class will**

**be executed.**

**1) class JdbOdbcDriver**

**2) {**

**3) static**

**4) {**

**5) JdbOdbcDriver driver= new JdbOdbcDriver();**

**6) DriverManager.registerDriver(driver);**

**7) }**

**8) }**

**Because of this Static Block, whenever we are loading automatically registering with**

***DriverManager* will be happened. Hence we are not required to perform this activity explicitly.**

**If we want to register explicitly without using *Class.forName()* then we can do as follows by using**

***registerDriver()* Method of *DriverManager* Class.**

****

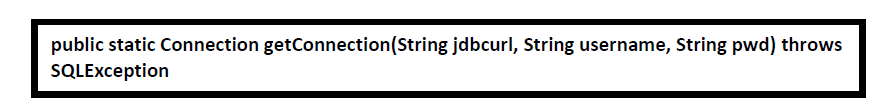
**Note: From JDBC 4.0 V (Java 1.6 V) onwards Driver Class will be loaded automatically from Class**

**Path and we are not required to perform this step explicitly.**

**Step-2: Establish Connection between Java Application and Database**

**Once we loaded and registered Driver, by using that we can establish Connection to the Database.**

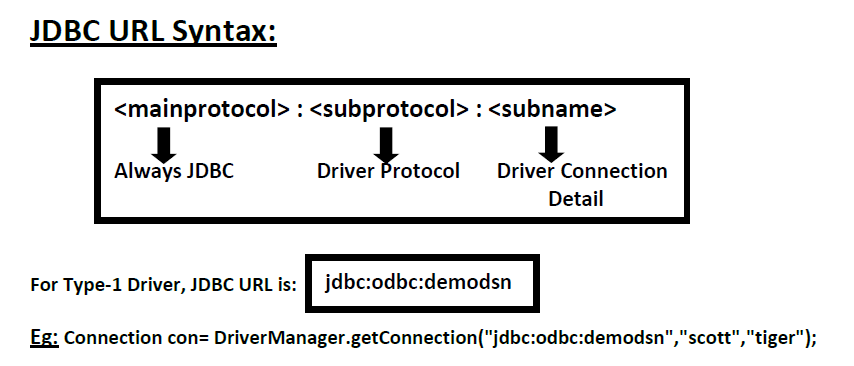
**For this *DriverManager* Class contains *getConnection()* Method.**

****

**Eg: Connection con= DriverManager.getConnection(jdbcurl,username,pwd);**

**"Jdbcurl" represents URL of the Database.**

***username* and *pwd* are Credentials to connect to the Database.**

****

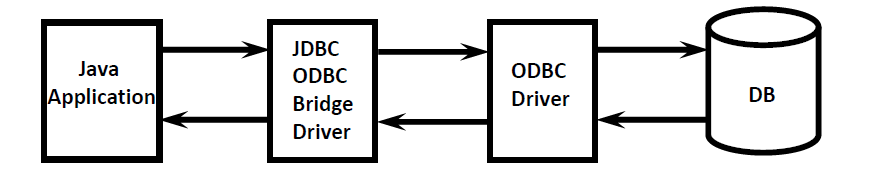
**Note:**

**DriverManager will use Driver Class internally to connect with Database.**

**DriverManager Class *getConnection()* Method internally calls Driver Class *connect()* Method.**

**DSN (Data Source Name) for Type-1 Driver:**

**Internally Type-1 Driver uses ODBC Driver to connect with Database.**

****

**ODBC Driver needs Database Name & its Location to connect with Database.**

**ODBC Driver collect this Information from DSN i.e. internally ODBC Driver will use DSN to get**

**Database Information (DSN Concept applicable only for Type-1 Driver)**

**There are 3 Types of DSN**

**1. User DSN**

**2. System DSN**

**3. File DSN**

**1) User DSN:**

**It is the non-sharable DSN and available only for Current User.**

**2) System DSN:**

**It is the sharable DSN and it is available for all Users who can access that System.**

**It is also known as Global DSN.**

**3) File DSN:**

**It is exactly same as User DSN but will be stored in a File with .dsn Extension.**

**Steps to configure DSN:**

**Start** ➔ **Settings** ➔ **Control Panel** ➔ **Performance and Maintenance** ➔ **Administrative Tools** ➔

**Data Sources (ODBC)** ➔ **Add** ➔ **Microsoft ODBC for Oracle** ➔ **Finish**

**For Windows 7 OR 8 OR 10:**

**C:\Windows\Syswow64 OR System32\Odbcad32.Exe** ➔ **Add** ➔ **Microsoft ODBC For Oracle** ➔

**Finish**

**Write A Java Program To Establish Connection To The Oracle Database By Using Type-1 Driver?**

**1) import java.sql.\*;**

**2) public class DbConnectDemo1**

**3) {**

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

**5) {**

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

**7) Connection con=DriverManager.getConnection("jdbc:odbc:demodsn7","scott","tiger"**

**);**

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

**9) {**

**10) System.out.println("Connection established Successfully");**

**11) }**

**12) else**

**13) {**

**14) System.out.println("Connection not established");**

**15) }**

**16) }**

**17) }**

**In the above Program Line 1 is Optional, because from JDBC 4.0V/ Java 1.6V onwards Driver Class**

**will be loaded automatically from the Class Path based on "jdbcurl".**

**Note:**

**To Compile and Run above Program we are not required to Place/Set any Jar File in the Class Path,**

**because Type-1 Driver is available by default as the Part of JDK.**

**\*\*\*Q. Connection is an interface, then how we can get Connection Object?**

**We are not getting Connection Object and we are getting its Implementation Class Object.**

**This Implementation Class is available as the Part of Driver Software. Driver Software Vendor is**

**responsible to provide Implementation Class.**

**We can print corresponding Class Name as follows SOP(con.getClass().getName());**

**o/p: sun.jdbc.odbc.JdbcOdbcConnection**

**Q.What Is The Advantage Of Using Interface Names In Our Application Instead Of Using**

**Implementation Class Names?**

**Interface Reference can be used to hold implemented Class Object. This Property is called**

**Polymorphism.**

**Connection** ➔ **sun.jdbc.odbc.JdbcOdbcConnection** ➔ **Type-1**

**Connection** ➔ **orcale.jdbc.OracleT4Connection** ➔ **Type-2**

**In JDBC Programs, Interface Names are fixed and these are provided by JDBC API. But**

**Implementation Classes are provided by Driver Software Vendor and these Names are varied from**

**Vendor to Vendor.**

**If we Hard Code Vendor provided Class Names in our Program then the Program will become**

**Driver Software Dependent and won't work for other Drivers.**

**If we want to change Driver Software then Total Program has to rewrite once again, which is**

**difficult. Hence it is always recommended to use JDBC API provided Interface Names in our**

**Application.**

**Step-3: Creation of Statement Object**

**Once we established Connection between *Java Application* and *Database*, we have to prepare SQL**

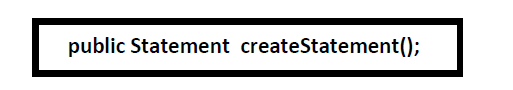
**Query and we have to send that Query to the Database. Database Engine will execute that Query**

**and send Result to Java Application.**

**To send SQL Query to the Database and to bring Results from Database to Java Application some**

**Vehicle must be required, which is nothing but Statement Object.**

**We can create Statement Object by using *createStatement()* Method of Connection Interface.**

****

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

**Step-4: Prepare, Send and Execute SQL Query**

**According to Database Specification, all SQL Commands are divided into following Types...**

**1. DDL (Data Definition Language) Commands:**

**Eg: Create Table, Alter Table, Drop Table Etc**

**2. DML (Data Manipulation Language) Commands:**

**Eg: Insert, Delete, Update**

**public Statement createStatement();**

**3. DQL (Data Query Language) Commands:**

**Eg: Select**

**4. DCL (Data Control Language) Commands:**

**Eg: Alter Password, Grant Access Etc..**

**5. Data Administration Commands**

**Eg: Start Audit**

**Stop Audit**

**6. Transactional Control Commands**

**Commit, Rollback, Savepoint Etc**

**According to Java Developer Point of View, all SQL Operations are divided into 2 Types...**

**1. Select Operations (DQL)**

**2. Non-Select Operations (DML, DDL Etc)**

**Basic SQL Commands**

**1) To Create a Table:**

**Create table movies (no number, name varchar2(20),hero varchar2(20),heroine varchar2(20));**

**2) To Drop/Delete Table:**

**drop table movies;**

**3) To Insert Data:**

**insert into movies values(1,'bahubali2','prabhas','anushka');**

**4) To Delete Data:**

**delete from movies where no=3;**

**5) To Update Data:**

**update movies set heroine='Tamannah' where no=1;**

**Select Operations and Non-Select Operations**

**Select Operations:**

**Whenever we are performing Select Operation then we will get a Group of Records as Result.**

**Eg: select \* from movies;**

**Non-Select Operations:**

**Whenever we are performing Non-Select Operation then we will get Numeric Value that**

**represents the Number of Rows affected.**

**Eg: update movies set heroine='Tamannah' where no=1;**

**Once we create Statement Object, we can call the following Methods on that Object to execute**

**our Queries.**

**1. executeQuery()**

**2. executeUpdate()**

**3. execute()**

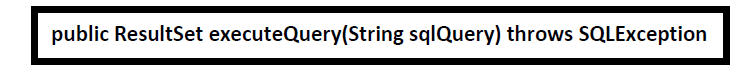
**1) executeQuery() Method:**

**We can use this Method for Select Operations.**

**Because of this Method Execution, we will get a Group of Records, which are represented by**

**ResultSet Object.**

**Hence the Return Type of this Method is ResultSet.**

****

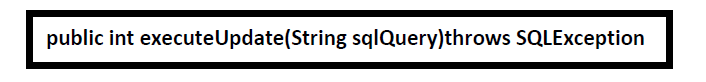
**Eg: ResultSet rs = st.executeQuery("select \* from movies");**

**2) executeUpdate() Method:**

**We can use this Method for Non-Select Operations (Insert|Delete|Update)**

**Because of this Method Execution, we won't get a Group of Records and we will get a Numeric**

**Value represents the Number of Rows effected. Hence Return Type of this Method is int**

****

**Eg: int rowCount = st.executeUpdate("delete from employees where esal>100000");**

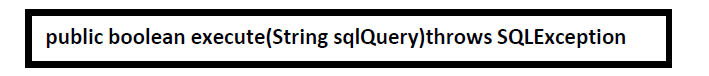
**SOP("The number of employees deleted:"+rowCount);**

**3) execute() method:**

**We can use this Method for both Select and Non-Select Operations.**

**If we don't know the Type of Query at the beginning and it is available dynamically at runtime**

**then we should use this execute() Method.**

****

**Eg:**

**1) boolean b = st.execute("dynamically provided query");**

**2)**

**3) if(b==true)//select query**

**4) {**

**5) ResultSet rs=st.getResultSet();**

**6) //use rs to get data**

**7) }**

**8) else// non-select query**

**9) {**

**10) int rowCount=st.getUpdateCount();**

**11) SOP("The number of rows effected:"+rowCount);**

**12) }**

**executeQuery() Vs executeUpdate() Vs execute():**

**1. If we know the Type of Query at the beginning and it is always Select Query then we should use**

**"*executeQuery()* Method".**

**2. If we know the Type of Query at the beginning and it is always Non-Select Query then we should**

**use *executeUpdate()* Method.**

**3. If we don't know the Type of SQL Query at the beginning and it is available dynamically at**

**Runtime (May be from *Properties File* OR From *Command Prompt* Etc) then we should go for**

**execute() Method.**

**Note:**

**Based on our Requirement we have to use corresponding appropriate Method.**

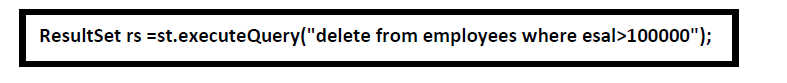
* **st.executeQuery();**
* **st.executeUpdate();**
* **st.execute();**
* **st.getResultSet();**
* **st.getUpdateCount();**

**public boolean execute(String sqlQuery)throws SQLException**

**Case-1: executeQuery() Vs Non-Select Query**

**Usually we can use executeQuery() Method for Select Queries. If we use for Non-Select Queries**

**then we cannot expect exact Result. It is varied from Driver to Driver.**

****

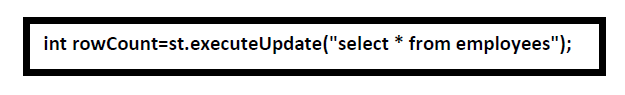
**For Type-1 Driver we will get SQLException. But for Type-4 Driver we won't get any Exception and**

**Empty ResultSet will be returned.**

**Case-2: executeUpdate() Vs Select Query**

**Usually we can use *executeUpdate()* Method for Non-Select Queries. But if we use for Select**

**Queries then we cannot expect the Result and it is varied from Driver to Driver.**

****

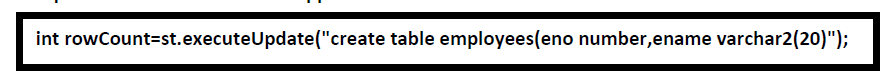
**For Type-1 Driver we will get *SQLException* where as for Type-4 Driver we won't get any Exception**

**and simply returns the Number of Rows selected.**

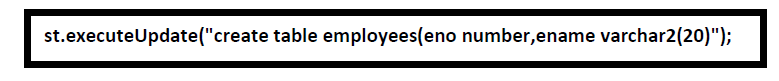
**Case-3: executeUpdate() Vs DDL Queries**

**If we use *executeUpdate()* Method for DDL Queries like Create Table, Alter Table, Drop Table Etc,**

**then Updated Record Count is not applicable. The Result is varied from Driver to Driver.**

****

**For Type-1 Driver, we will get -1 and For Type-4 Driver, we will get 0**

****

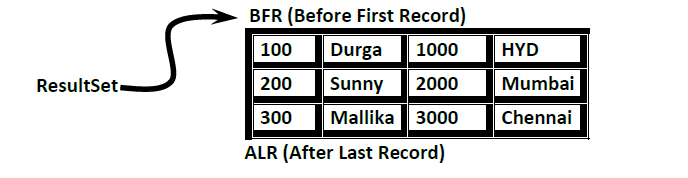
**Step-5: Process Result from ResultSet**

**After executing Select Query, Database Engine will send Result back to Java Application. This**

**Result is available in the form of ResultSet.**

**i.e. ResultSet holds Result of executeQuery() Method, which contains a Group of Records.**

**By using ResultSet we can get Results.**

****

**ResultSet is a Cursor always locating Before First Record (BFR).**

**To check whether the next Record is available OR not, we have to use *rs.next()* Method.**

****

**This Method Returns True if the next Record is available, otherwise returns False.**

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

**2) {**

**3) read data from that record**

**4) }**

**If next Record is available then we can get Data from that Record by using the following Getter**

**Methods.**

**1. getXxx(String columnName)**

**2. getXxx(int columnIndex)**

**Like getInt(), getDouble(), getString() etc..**

**Note:**

**In JDBC, Index is always one based but not Zero based i.e. Index of First Column is 1 but not 0.**

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

**2) {**

**3) SOP(rs.getInt("ENO")+".."+rs.getString("ENAME")+".."+rs.getDouble("ESAL")+".."+rs.getS**

**tring("EADDR"));**

**4) OR**

**5) SOP(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+".."+rs.getString(4));**

6) **}**

**Note:**

**Readability wise it is recommended to use Column Names, but Performance wise it is**

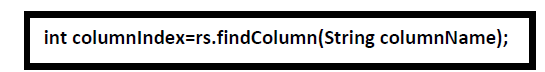
**recommended to use Column Index. (Because comparing Numbers is very easy than comparing**

**String Values)**

**Hence if we are handling very large Number of Records then it is highly recommended to use**

**Index.**

**If we know Column Name then we can find corresponding Index as follows...**

****

**Conclusions:**

**1. ResultSet follows "Iterator" Design Pattern.**

**2. ResultSet Object is always associated with Statement Object.**

**3. Per Statement only one ResultSet is possible at a time. if we are trying to open another**

**ResultSet then automatically first ResultSet will be closed.**

**Eg:**

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

**RS rs1 = st.executeQuery("select \* from movies");**

**RS rs2 = st.executeQuery("select \* from employees");**

**In the above Example Rs1 will be closed automatically whenever we are trying to open Rs2.**

**Step 6: Close the Connection**

**After completing Database Operations it is highly recommended to close the Resources whatever**

**we opened in reverse order of opening.**

**1.rs.close();**

**It closes the ResultSet and won't allow further processing of ResultSet**

**2. st.close();**

**It closes the Statement and won't allow sending further Queries to the Database.**

**3. con.close();**

**It closes the Connection and won't allow for further Communication with the Database.**

**Conclusions:**

* **Per Statement only one ResultSet is possible at a time.**
* **Per Connection multiple Statement Objects are possible.**
* **Whenever we are closing Statement Object then automatically the corresponding ResultSet will be closed.**
* **Similarly, whenever we are closing Connection Object automatically corresponding Statement Objects will be closed.**
* **Hence we required to use only *con.close();***

**1.7 Version: try With Resources**

**Usually we will close the Resources inside finally Block.**

**1) try**

**2) {**

**3) Open Database Connection**

**4) }**

**5) catch(X e)**

**6) {**

**7) }**

**8) finally**

**9) {**

**10) Close That Database Connection**

**11) }**

**But in Java 1.7 Version try with Resources Concept introduced.**

**The Advantage of this Concept is, whatever Resources we opened as the Part of *try* Block, will be**

**closed automatically once Control reaches End of *try* Block either Normally OR Abnormally. We**

**are not required to close explicitly.**

**Hence until 1.6 Version *finally* Block is just like Hero but from 1.7 Version onwards *finally* Block**

**became Zero.**

**try (Resource)**

**{**

**}**

**Eg:**

**try (Connection con = DM.getConnection(-,-,-))**

**{**

**Use con based on our Requirement**

**Once Control reaches End of try Block, automatically con will be closed, we are not**

**required to close explicitly**

**}**

**Real Time Coding Standards for JDBC Application**

**1. Every Java Class should be Part of some Package. Hence it is recommended to take Package Statement.**

**2. It is recommended to use explicit Class Imports than implicit Class Imports because these**

**imports improve Readability of the Code.**

**Eg:**

**import java.sql.\*;** ➔ **Implicit Class Import**

**import java.sql.Connection;** ➔ **Explicit Class Import**

**3. It is recommended to use *try-catch* over *throws* Statement, because there is a Guarantee for the normal termination of the Program.**

**Even we are using *throws* Statement, somewhere compulsory we should handle that Exception by**

**using *try-catch.***

**Eg:**

**1) m1()**

**2) {**

**3) try**

**4) {**

**5) m2();**

**6) }**

**7) catch(Exception e)**

**8) {**

**9) }**

**10) }**

**11)**

**12) m2() throws Exception**

**13) {**

**14) ....**

**15) }**

**4. Avoid Duplicate Code as much as possible, otherwise Maintenance Problems may rise.**

**5. We have to use meaningful Names for Classes, Methods, Variables etc. It improves readability of the Code.**

**If any Code repeatedly required, we have to separate that Code inside some other Class and we**

**can call its Functionality where ever it is required.**

**In JDBC Applications, getting Connection and closing the Resources are common Requirement.**

**Hence we can separate this Code into some *util* Class, and we can reuse that Code where ever it is**

**required.**

**Working with MySQL Database**

**Current Version :5.7.14**

**Vendor: Sun Microsystems/Oracle Corporation**

**Open Source And Freeware**

**Default Port: 3306**

**Default User: root**

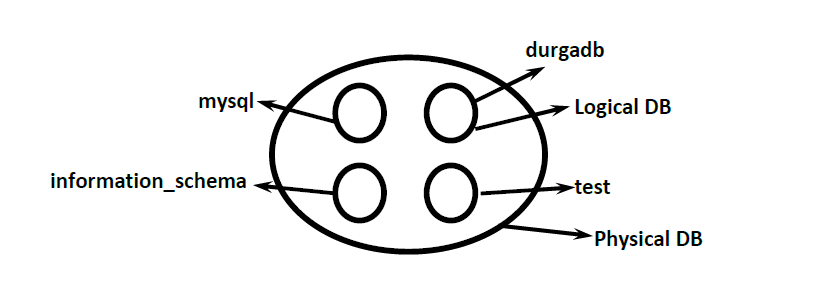
**Note:**

**In MySql, everything we have to work with our own Databases, which are also known as *Logical***

***Databases.***

**The Default Databases are:**

* **information\_schema**
* **mysql**
* **test**

****

**Here only one Physical Database but 4 Logical Databases are available.**

**Commonly used Commands:**

**1. To know available Databases**

**mysql> show databases;**

**2. To create our own Logical Database**

**mysql> create database elanzadb;**

**3. To Drop our own Database**

**mysql> drop database elanzadb;**

**4. To use a particular Logical Database**

**mysql> use elanzadb; OR mysql> connect elanzadb;**

**5. To Create a Table:**

**create table employees(eno int(5) primary key,ename varchar(20),esal double(10,2),eaddr**

**varchar(20));**

**6. To Insert Data**

**insert into employees values(100,'elanza',1000,'Hyd');**

**Instead of Single Quotes, we can use Double Quotes also.**

**JDBC Information:**

**In general, we can use Type-4 Driver to communicates with MySQL Database which is provided by**

**MySQL Vendor, and its Name is connector/J**

**Jar File: Driver Software is available in the following Jar File.**

**mysql-connector-java-5.1.41-bin.jar**

**We have to download separately from MySql Web Site.**

**jdbc url: jdbc:mysql://localhost:3306/elanzadb**

**jdbc:mysql:///elanzadb**

**If MySQL is available in Local System then we can specify JDBC URL as above.**

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

**User Name: root**

**pwd: root**

**We required to Set Class Path of MySql Driver Jar File**

**Variable Name: CLASSPATH**

**Variable Value: D:\mysql-connector-java-bin.jar;.;**

**Program to Demonstrate JDBC with MySql Database**

**1) import java.sql.\*;**

**2) public class JdbcMySQLDemo**

**3) {**

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

**5) {**

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

**7) Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/elanzad**

**b","root","root");**

**8) Statement st = con.createStatement();**

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

**10) while(rs.next())**

**11) {**

**12) System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+".."+rs.get**

**String(4));**

**13) }**

**14) con.close();**

**15) }**

**16) }**

**Life Cycle of SQL Query Execution**

**From Java application if we submit SQL Query by using Statement object execute method,**

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

**ResultSet rs = st.executeQuery(sqlQuery);**

**Then database engine will perform the following sequence of activities**

**1. Compilation**

**2. Execution**

**3. Fetch Result**

**1. Compilation:**

**As the part of compilation, 0database engine will perform the following activities**

**A.Query Tokenization:**

**In this step total SQL Query will be divided into number of tokens and generate a Stream of tokens as output.**

**B.Query Parsing:**

**In this step, database engine will create parse tree (query tree) with stream of tokens. If the Query tree is proper then there are no syntactical mistakes in that query.**

**If the query tree construction fails then it indicates that there are some syntactical errors present in SQL Query and SQLException will be raised.**

**C.Query Optimization:**

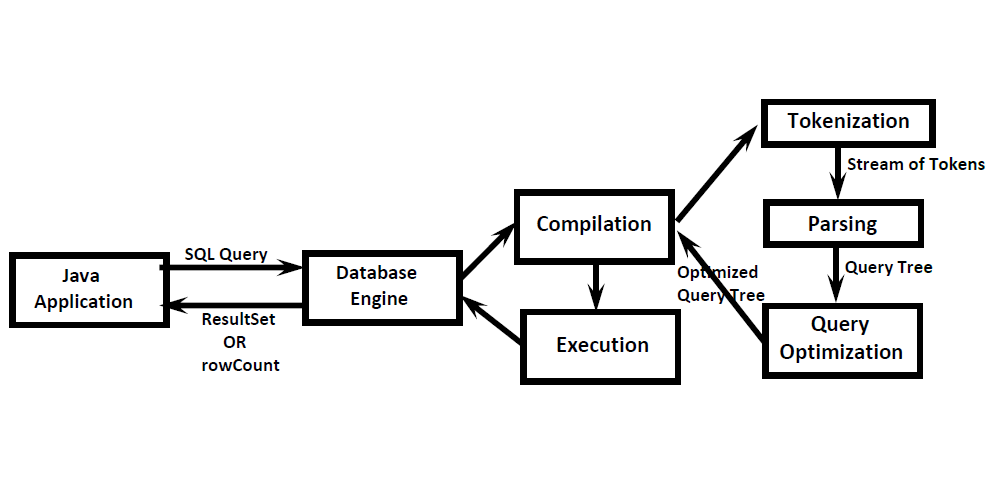
**The main purpose of query optimization is to improve performance. In this step optimized query tree will be constructed.**

**2. Execution of sql Query:**

**Once compilation success then database engine will take that query tree as input and execute that query by using interpreter.**

**3. Fetch the Result:**

**Database engine will provide result of SQL Query either in the form of ResultSet (for select query) OR in the form of rowCount (for non-select query) to the Java application.**

****

**PreparedStatement (I)**

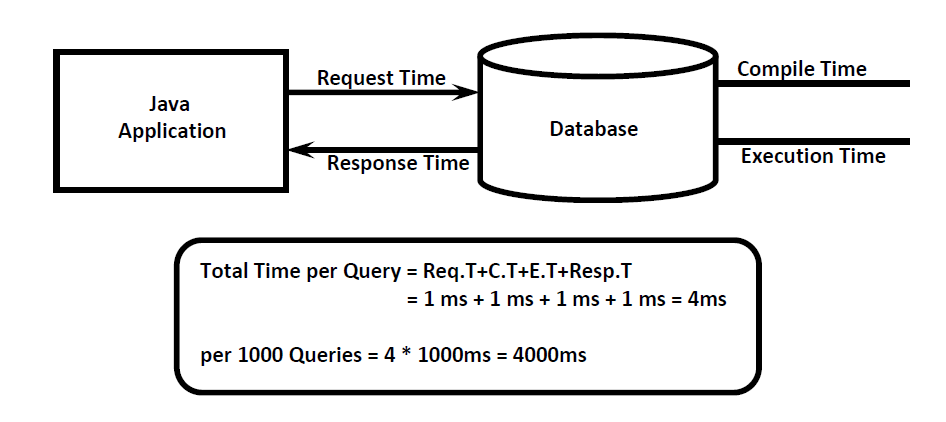
**Need of PreparedStatement:**

**In the case of normal Statement, whenever we are executing SQL Query, every time compilation**

**and execution will be happened at database side.**

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

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

****

**Sometimes in our application,we required to execute same query multiple times with same or**

**different input values.**

**Eg1:**

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

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

**Query is same but source and destination places may be different. This query is required to**

**execute lakhs of times per day.**

**Eg2:**

**In BookMyShow application, it is very common requirement to display theatre names where a**

**particular movie running/playing in a particular city**

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

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

**names and different locations.**

**For the above requirements if we use Statement object, then the query is required to compile and**

**execute every time, which creates performance problems.**

**To overcome this problem, we should go for PreparedStatement.**

**The main advantage of PreparedStatement is the query will be compiled only once even though**

**we are executing multiple times, so that overall performance of the application will be improved.**

**We can create PreparedStatement by using prepareStatement() method of Connection interface.**

**public PreparedStatement prepareStatement(String sqlQuery) throws SQLException**

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

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

**in the database.**

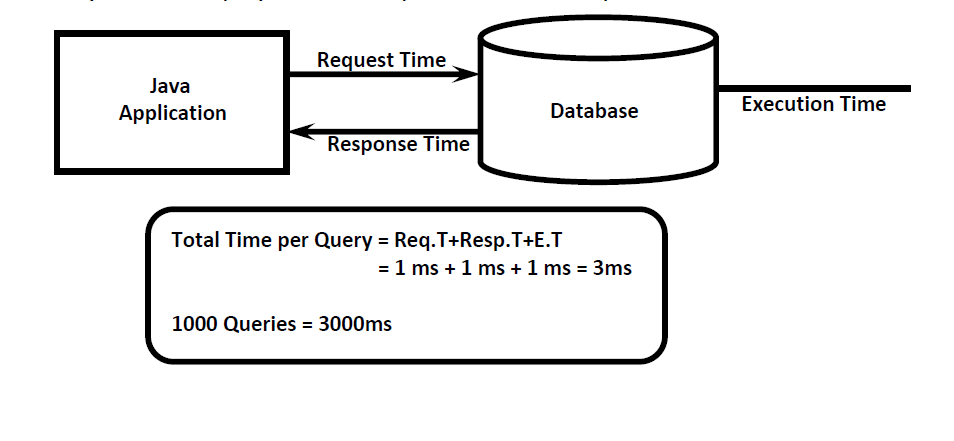
**That pre compiled query will be returned to the java application in the form of PreparedStatement**

**object.**

**Hence PreparedStatement represents "pre compiled sql query".**

**Whenever we call execute methods,database engine won't compile query once again and it will**

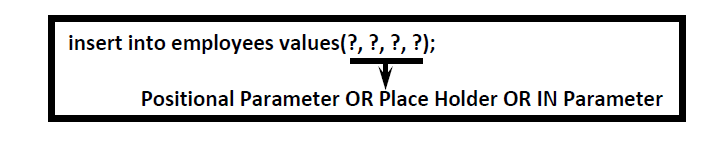
**directly execute that query,so that overall performance will be improved.**

****

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

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

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

****

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

**PreparedStatement pst = con.prepareStatement(sqlQuery);**

**At this line only query will be compiled.**

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

**using corresponding setter methods.**

**We have to consider these positional parameters from left to right and these are 1 index based. i.e**

**index of first positional parameter is 1 but not zero.**

**pst.setInt(1,100);**

**pst.setString(2,"elanza");**

**pst.setDouble(3,1000);**

**pst.setString(4,"Hyd");**

**Note:**

**Before executing the query, for every positional parameter we have to provide input values**

**otherwise we will get SQLException**

**4. Execute SQL Query:**

**PreparedStatement is the child interface of Statement and hence all methods of Statement**

**interface are bydefault available to the PreparedStatement.Hence we can use same methods to**

**execute sql query.**

**executeQuery()**

**executeUpdate()**

**execute()**

**Note:**

**We can execute same parameterized query multiple times with different sets of input values.**

**In this case query will be compiled only once and we can execute multiple times.**

**insert into employees values(?, ?, ?, ?);**

**Positional Parameter OR Place Holder OR IN Parameter**

**Q. Which of the following are valid sql statements?**

**1. delete from employees where ename=?**

**2. delete from employees ? ename=?**

**3. delete from ? where ename=?**

**4. delete ? employees where ename=?**

**Note:**

**We can use ? only in the place of input values and we cannot use in the place of sql**

**keywords,table names and column names.**

**Static Query vs Dynamic Query:**

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

**Eg: delete from employees where ename='elanza'**

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

**Eg: select \* from employees where esal>?**

**Program-1 to Demonstrate PreparedStatement:**

**1) import java.sql.\*;**

**2) public class PreparedStatementDemo1**

**3) {**

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

**5) {**

**6) String driver="oracle.jdbc.OracleDriver";**

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

**8) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**9) String user="scott";**

**10) String pwd="tiger";**

**11) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**12) String sqlQuery ="delete from employees where ename=?";**

**13)**

**14) PreparedStatement pst = con.prepareStatement(sqlQuery);**

**15) pst.setString(1,"Mallika");**

**16) int updateCount=pst.executeUpdate();**

**17) System.out.println("The number of rows deleted :"+updateCount);**

**18)**

**19) System.out.println("Reusing PreparedStatement to delete one more record...");**

**20) pst.setString(1,"Elanza");**

**21) int updateCount1=pst.executeUpdate();**

**22) System.out.println("The number of rows deleted :"+updateCount1);**

**23) con.close();**

**24) }**

25) **}**

**Program-2 to Demonstrate PreparedStatement:**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) public class PreparedStatementDemo2**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

**8) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**9) String user="scott";**

**10) String pwd="tiger";**

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

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) String sqlQuery="insert into employees values(?,?,?,?)";**

**14) PreparedStatement pst = con.prepareStatement(sqlQuery);**

**15)**

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

**17) while(true)**

**18) {**

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

**20) int eno=sc.nextInt();**

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

**22) String ename=sc.next();**

**23) System.out.println("Employee Sal:");**

**24) double esal=sc.nextDouble();**

**25) System.out.println("Employee Address:");**

**26) String eaddr=sc.next();**

**27) pst.setInt(1,eno);**

**28) pst.setString(2,ename);**

**29) pst.setDouble(3,esal);**

**30) pst.setString(4,eaddr);**

**31) pst.executeUpdate();**

**32) System.out.println("Record Inserted Successfully");**

**33) System.out.println("Do U want to Insert one more record[Yes/No]:");**

**34) String option = sc.next();**

**35) if(option.equalsIgnoreCase("No"))**

**36) {**

**37) break;**

**38) }**

**39) }**

**40) con.close();**

**41) }**

**42) }**

**Advantages of PreparedStatement:**

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

**compiled only once.**

**2. Network traffic will be reduced between java application and database b'z we are not required**

**to send query every time to the database.**

**3. We are not required to provide input values at the beginning and we can provide dynamically**

**so that we can execute same query multiple times with different sets of values.**

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

**specific format.**

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

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

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

**Limitation of PreparedStatement:**

**We can use PreparedStatement for only one sql query (Like CDMA Phone), but we can use simple**

**Statement to work with any number of queries (Like GSM Phone).**

**Eg:**

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

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

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

**st.executeUpdate("delete...");**

**Here We Are Using One Statement Object To Execute 3 Queries**

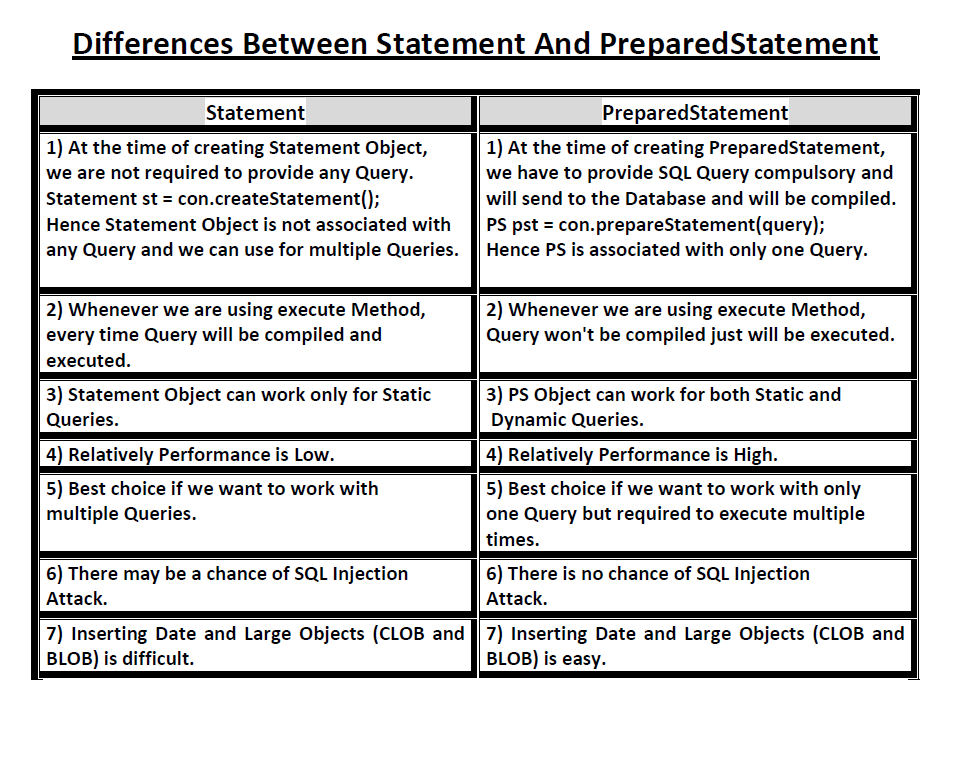
**PreparedStatement pst = con.prepareStatement("insert into employees..");**

**Here PreparedStatement object is associated with only insert query.**

**Note:**

**Simple Statement can be used only for static queries where as PreparedStatement can used for**

**both static and dynamic queries.**

****

**Stored Procedures and CallableStatement**

**In our programming if any code repeatedly required,then we can define that code inside a method**

**and we can call that method multiple times based on our requirement.**

**Hence method is the best reusuable component in our programming.**

**Similarly in the database programming,if any group of sql statements is repeatedly required then**

**we can define those sql statements in a single group and we can call that group repeatedly based**

**on our requirement.**

**This group of sql statements that perform a particular task is nothing but Stored Procedure.Hence**

**stored procedure is the best reusable component at database level.**

**Hence Stored Procedure is a group of sql statements that performs a particular task.**

**These procedures stored in database permanently for future purpose and hence the name stored**

**procedure.**

**Usually stored procedures are created by Database Admin (DBA).**

**Every database has its own language to create Stored Procedures.**

**Oracle has** ➔ **PL/SQL**

**MySQL has** ➔ **Stored Procedure Language**

**Microsoft SQL Server has** ➔ **Transact SQL(TSQL**

**Similar to methods stored procedure has its own parameters. Stored Procedure has 3 Types of**

**parameters.**

**1. IN parameters(to provide input values)**

**2. OUT parameters(to collect output values)**

**3. INOUT parameters(to provide input and to collect output)**

**Eg 1 :**

**Z:=X+Y;**

**X,Y are IN parameters and Z is OUT parameter**

**Eg 2:**

**X:=X+X;**

**X is INOUT parameter**

**Syntax for creating Stored Procedure (Oracle):**

**1) create or replace procedure procedure1(X IN number, Y IN number,Z OUT number) as**

**2) BEGIN**

**3) z:=x+y;**

4) **END;**

**Note:**

**SQL and PL/SQL are not case-sensitive languages. We can use lower case and upper case also.**

**After writing Stored Procedure, we have to compile for this we required to use "/" (forward slash)**

**/** ➔ **For compilation**

**while compiling if any errors occurs,then we can check these errors by using the following**

**command**

**SQL> show errors;**

**Once we created Stored Procedure and compiled successfully,we have to register OUT parameter**

**to hold result of stored procedure.**

**SQL> variable sum number; (declaring a variable)**

**We can execute with execute command as follows**

**SQL> execute procedure1(10,20,:sum);**

**SQL> print sum;**

**Eg 2:**

**1) create or replace procedure procedure1(X IN number,Y OUT number) as**

**2) BEGIN**

**3) Y:= x\*x;**

**4) END;**

**5) /**

**SQL> variable square number;**

**SQL> execute procedure1(10,:square);**

**SQL> print square;**

**SQUARE**

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

**100**

**87**



**Eg3: Procedure To Print Employee Salary Based On Given Employee Number.**

**1) create or replace procedure procedure2(eno1 IN number,esal1 OUT number) as**

**2) BEGIN**

**3) select esal into esal1 from employees where eno=eno1;**

**4) END;**

**5) /**

**SQL>variable salary number;**

**SQL>execute procedure2(100,:salary);**

**SQL>print salary;**

**Java Code for calling Stored Procedures:**

**If we want to call stored procedure from java application, then we should go for**

**CallableStatement.**

**CallableStatement is an interface present in java.sql package and it is the child interface of**

**PreparedStatement.**

**Driver software vendor is responsible to provide implementation for CallableStatement interface.**

**We can create CallableStatement object by using following method of Connection interface.**

**public CallableStatement prepareCall(String procedure\_call) throws SQLException**

**Eg: CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");**

**Statement (I)**

**PreparedStatement (I)**

**CallableStatement (I)**

**{call addProc (?,?,?)}**

**JDBC Keyword Procedure Name Parameters**

**88**



**Whenever JVM encounters this line,JVM will send call to database.Database engine will check**

**whether the specified procedure is already available or not. If it is available then it returns**

**CallableStatement object representing that procedure.**

**Mapping Java Types to database Types by using JDBC Types:**

**Java related data types and database related data types are not same.Some mechanism must be**

**required to convert java types to database types and database types to java types. This**

**mechanism is nothing but "JDBC Types",which are also known as "Bridge Types".**

**Note: JDBC data types are defined as constants in "java.sql.Types" class.**

**Process to call Stored Procedure from java application by using CallableStatement:**

**1. Make sure Stored procedure available in the database**

**1) create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT numbe**

**r) as**

**2) BEGIN**

**3) num3 :=num1+num2;**

**4) END;**

**5) /**

**2. Create a CallableStatement with the procedure call.**

**CallableStatement cst = con.prepareCall("{call addProc(?,?,?)}");**

**3. Provide values for every IN parameter by using corresponding setter methods.**

**4. Register every OUT parameter with JDBC Types.**

**Java Data Type JDBC Data Type Oracle Data Type**

**int Types.INTEGER number**

**float Types.FLOAT number**

**String Types.VARCHAR varchar.varchar2**

**java.sql.Date Types.DATE date**

**:**

**:**

**:**

**:**

**:**

**:**

**:**

**:**

**:**

**cst.setInt(1, 100);**

**cst.setInt(2, 200);**

**index value**

**89**



**If stored procedure has OUT parameter then to hold that output value we should register every**

**OUT parameter by using the following method.**

**public void registerOutParameter (int index, int jdbcType)**

**Eg: cst.registerOutParameter(3,Types.INTEGER);**

**Note:**

**Before executing procedure call, all input parameters should set with values and every OUT**

**parameter we have to register with jdbc type.**

**5. execute procedure call**

**cst.execute();**

**6. Get the result from OUT parameter by using the corresponding getXxx() method.**

**Eg: int result=cst.getInt(3);**

**Stored Procedures App1: JDBC Program to call StoredProcedure which can take two input**

**numbers and produces the result.**

**Stored Procedure:**

**1) create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT numbe**

**r) as**

**2) BEGIN**

**3) num3 :=num1+num2;**

**4) END;**

5) **/**

**StoredProceduresDemo1.java**

**1) import java.sql.\*;**

**2) class StoredProceduresDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");**

**8) cst.setInt(1,100);**

**9) cst.setInt(2,200);**

**10) cst.registerOutParameter(3,Types.INTEGER);**

**11) cst.execute();**

**12) System.out.println("Result.."+cst.getInt(3));**

**13) con.close();**

**14) }**

**15) }**

**90**



**Stored Procedures App2: JDBC Program to call StoredProcedure which can take employee**

**number as input and provides corresponding salary.**

**Stored Procedure:**

**1) create or replace procedure getSal(id IN number,sal OUT number) as**

**2) BEGIN**

**3) select esal into sal from employees where eno=id;**

**4) END;**

5) **/**

**StoredProceduresDemo2.java**

**1) import java.sql.\*;**

**2) class StoredProceduresDemo2**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) CallableStatement cst=con.prepareCall("{call getSal(?,?)}");**

**8) cst.setInt(1,100);**

**9) cst.registerOutParameter(2,Types.FLOAT);**

**10) cst.execute();**

**11) System.out.println("Salary ..."+cst.getFloat(2));**

**12) con.close();**

**13) }**

**14) }**

**Stored Procedures App3: JDBC Program to call StoredProcedure which can take employee**

**number as input and provides corresponding name and salary.**

**Stored Procedure:**

**1) create or replace procedure getEmpInfo(id IN number,name OUT varchar2,sal OUT numbe**

**r) as**

**2) BEGIN**

**3) select ename,esal into name,sal from employees where eno=id;**

**4) END;**

**5) /**

**StoredProceduresDemo3.java**

**1) import java.sql.\*;**

**2) class StoredProceduresDemo3**

**3) {**

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

**5) {**

**91**



**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) CallableStatement cst=con.prepareCall("{call getEmpInfo(?,?,?)}");**

**8) cst.setInt(1,100);**

**9) cst.registerOutParameter(2,Types.VARCHAR);**

**10) cst.registerOutParameter(3,Types.FLOAT);**

**11) cst.execute();**

**12) System.out.println("Employee Name is :"+cst.getString(2));**

**13) System.out.println("Employee Salary is :"+cst.getFloat(3));**

**14) con.close();**

**15) }**

16) **}**

**92**



**CURSORS**

**The results of SQL Queries will be stored in special memory area inside database software. This**

**memory area is called Context Area.**

**To access Results of this context area,Some pointers are required and these pointers are nothing**

**but cursors.**

**Hence the main objective of cursor is to access results of SQL Queries.**

**There are 2 types of cursors**

**1. Implicit cursors**

**2. Explicit cursors**

**1. Implicit cursors:**

**These cursors will be created automatically by database software to hold results whenever a**

**particular type of sql query got executed.**

**2. Explicit Cursors:**

**These cursors will be created explicitly by the developer to hold results of particular sql queries.**

**Eg 1: SYS\_REFCURSOR can be used to access result of select query i.e to access ResultSet.**

**Eg 2: %ROWCOUNT is an implicit cursor provided by Oracle to represent the number of rows**

**effected b'z of insert,delete and update queries.**

**Eg 3: %FOUND is an implicit cursor provided by Oracle to represent whether any rows effected or**

**not b'z of insert,delete and update operations(non-select query)**

**SYS\_REFCURSOR VS OracleTypes.CURSOR:**

**To register SYS\_REFCURSOR type OUT parameter JDBC does not contain any type.**

**To handle this situation,Oracle people provided**

**OracleTypes.CURSOR**

**class name variable**

**93**



**OracleTypes is a java class present in oracle.jdbc package and it is available as the part of**

**ojdbc6.jar**

**If OUT parameter is SYS\_REFCURSOR type,then we can get ResultSet by using getObject() method.**

**But return type of getObject() method is Object and hence we should perform typecasting.**

**ResultSet rs = (ResultSet)cst.getObject(1);**

**Eg:**

**1) create or replace prodecure getAllEmpInfo(emps OUT SYS\_REFCURSOR) as**

**2) BEGIN**

**3) OPEN emps for**

**4) select \* from employees;**

**5) end;**

6) **/**

**1) CallableStatement cst=con.prepareCall("{ call getAllEmpInfo(?)}");**

**2) cst.registerOutParameter(1,OracleTypes.CURSOR);**

**3) cst.execute();**

**4) RS rs = (RS)cst.getObject(1);**

**5) while(rs.next())**

**6) {**

**7) SOP(rs.getInt(1)+".."+rs...);**

**8) }**

**Stored Procedures App4: JDBC Program to call StoredProcedure which returns all**

**Employees info by using SYS\_REFCURSOR**

**Stored Procedure:**

**1) create or replace procedure getAllEmpInfo1(sal IN number,emps OUT SYS\_REFCURSOR) as**

**2) BEGIN**

**3) open emps for**

**4) select \* from employees where esal<sal;**

**5) END;**

**6) /**

**StoredProceduresDemo4.java**

**1) import java.sql.\*;**

**2) import oracle.jdbc.\*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar**

**3) class StoredProceduresDemo4**

**4) {**

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

**94**



**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**8) CallableStatement cst=con.prepareCall("{call getAllEmpInfo1(?,?)}");**

**9) cst.setFloat(1,6000);**

**10) cst.registerOutParameter(2,OracleTypes.CURSOR);**

**11) cst.execute();**

**12) ResultSet rs = (ResultSet)cst.getObject(2);**

**13) boolean flag=false;**

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

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

**16) while(rs.next())**

**17) {**

**18) flag=true;**

**19) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**20) }**

**21) if(flag== false)**

**22) {**

**23) System.out.println("No Recors Available");**

**24) }**

**25) con.close();**

**26) }**

**27) }**

**Stored Procedures App5: JDBC Program to call StoredProcedure which returns all**

**Employees info by using SYS\_REFCURSOR based initial characters of the name**

**Stored Procedure:**

**1) create or replace procedure getAllEmpInfo2(initchars IN varchar,emps OUT SYS\_REFCURSO**

**R) as**

**2) BEGIN**

**3) open emps for**

**4) select \* from employees where ename like initchars;**

**5) END;**

6) **/**

**StoredProceduresDemo5.java**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) import oracle.jdbc.\*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar**

**4) class StoredProceduresDemo5**

**5) {**

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

**7) {**

**8) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**95**



**9) CallableStatement cst=con.prepareCall("{call getAllEmpInfo2(?,?)}");**

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

**11) System.out.println("Enter initial characters of the name");**

**12) String initialchars=sc.next()+"%";**

**13) cst.setString(1,initialchars);**

**14) cst.registerOutParameter(2,OracleTypes.CURSOR);**

**15) cst.execute();**

**16) ResultSet rs = (ResultSet)cst.getObject(2);**

**17) boolean flag= false;**

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

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

**20) while(rs.next())**

**21) {**

**22) flag=true;**

**23) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**24) }**

**25) if(flag== false)**

**26) {**

**27) System.out.println("No Recors Available");**

**28) }**

**29) con.close();**

**30) }**

**31) }**

**96**



**Functions**

**Functions are exactly same as procedures except that function has return statement directly.**

**Procedure can also returns values indirectly in the form of OUT parameters.**

**Usually we can use procedure to define business logic and we can use functions to perform some**

**calculations like getAverage() , getMax() etc..**

**Syntax for functions:**

**1) create or replace function getAvg(id1 IN number,id2 IN number)return number**

**2) as**

**3) sal1 number;**

**4) sal2 number;**

**5) BEGIN**

**6) select esal into sal1 from employees where eno=id1;**

**7) select esal into sal2 from employees where eno=id2;**

**8) return (sal1+sal2)/2;**

**9) END;**

**10) /**

**Function call can return some value.Hence the syntax of function call is**

**CS cst = con.prepareCall("{? = call getAvg(?,?)}");**

**return value of function call should be register as OUT parameter.**

**Stored Procedures App6: JDBC Program to call Function which returns average salary of**

**given two employees**

**Stored Procedure**

**1) create or replace function getAvg(id1 IN number,id2 IN number)return number**

**2) as**

**3) sal1 number;**

**4) sal2 number;**

**5) BEGIN**

**6) select esal into sal1 from employees where eno=id1;**

**7) select esal into sal2 from employees where eno=id2;**

**8) return (sal1+sal2)/2;**

**9) END;**

**10) /**

**97**



**StoredProceduresDemo6.java**

**1) import java.sql.\*;**

**2) class StoredProceduresDemo6**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) CallableStatement cst=con.prepareCall("{?=call getAvg(?,?)}");**

**8) cst.setInt(2,100);**

**9) cst.setInt(3,200);**

**10) cst.registerOutParameter(1,Types.FLOAT);**

**11) cst.execute();**

**12) System.out.println("Salary ..."+cst.getFloat(1));**

**13) con.close();**

**14) }**

15) **}**

**Stored Procedures App7: JDBC Program to call function returns all employees information**

**based on employee numbers**

**Stored Procedure**

**1) create or replace function getAllEmpInfo4(no1 IN number,no2 IN number) return SYS\_REF**

**CURSOR as**

**2) emps SYS\_REFCURSOR;**

**3) BEGIN**

**4) open emps for**

**5) select \* from employees where eno>=no1 and eno<=no2;**

**6) return emps;**

**7) END;**

8) **/**

**StoredProceduresDemo7.java**

**1) import java.sql.\*;**

**2) import oracle.jdbc.\*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar**

**3) class StoredProceduresDemo7**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**8) CallableStatement cst=con.prepareCall("{?=call getAllEmpInfo4(?,?)}");**

**9) cst.setInt(2,1000);**

**10) cst.setInt(3,2000);**

**11) cst.registerOutParameter(1,OracleTypes.CURSOR);**

**12) cst.execute();**

**98**



**13) ResultSet rs = (ResultSet)cst.getObject(1);**

**14) boolean flag=false;**

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

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

**17) while(rs.next())**

**18) {**

**19) flag=true;**

**20) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**21) }**

**22) if(flag== false)**

**23) {**

**24) System.out.println("No Recors Available");**

**25) }**

**26) con.close();**

**27) }**

**28) }**

**Stored Procedures App8: JDBC Program to call function to Demonstrate SQL%ROWCOUNT**

**implicit cursor**

**Stored Procedure**

**1) create or replace function getDeletedEMPInfo(no1 IN number,count OUT number) return S**

**YS\_REFCURSOR as**

**2) emps SYS\_REFCURSOR;**

**3) BEGIN**

**4) open emps for**

**5) select \* from employees where eno=no1;**

**6) delete from employees where eno=no1;**

**7) count :=SQL%ROWCOUNT;**

**8) return emps;**

**9) END;**

**10) /**

**StoredProceduresDemo8.java**

**1) import java.sql.\*;**

**2) import oracle.jdbc.\*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar**

**3) class StoredProceduresDemo8**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**8) CallableStatement cst=con.prepareCall("{?=call getDeletedEMPInfo(?,?)}");**

**9) cst.setInt(2,100);**

**10) cst.registerOutParameter(1,OracleTypes.CURSOR);**

**11) cst.registerOutParameter(3,Types.INTEGER);**

**99**



**12) cst.execute();**

**13) ResultSet rs = (ResultSet)cst.getObject(1);**

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

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

**16) while(rs.next())**

**17) {**

**18) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**19) }**

**20) int count=cst.getInt(3);**

**21) System.out.println("The number of rows deleted: "+count);**

**22) con.close();**

**23) }**

**24) }**

**Statement vs PreparedStatement vs CallableStatement:**

**1. We can use normal Statement to execute multiple queries.**

**st.executeQuery(query1)**

**st.executeQuery(query2)**

**st.executeUpdate(query2)**

**i.e if we want to work with multiple queries then we should go for Statement object.**

**2. If we want to work with only one query,but should be executed multiple times then we should**

**go for PreparedStatement.**

**3. If we want to work with stored procedures and functions then we should go for**

**CallableStatement.**

**100**



**Batch Updates**

**Need of Batch Updates:**

**When we submit multiple SQL Queries to the database one by one then lot of time will be wasted**

**in request and response.**

**In the case of simple Statement:**

**In the case of PreparedStatement:**

**In the above 2 cases , we are trying to submit 1000 queries to the database one by one. For**

**submitting 1000 queries we need to communicate with the database 1000 times. It increases**

**network traffic Between java application and database and even creates performance problems**

**also.**

**Compile Time**

**Java**

**Application Database**

**Request Time**

**Response Time Execution Time**

**Total Time per Query = Req.T+C.T+E.T+Resp.T**

**= 1 ms + 1 ms + 1 ms + 1 ms = 4ms**

**per 1000 Queries = 4 \* 1000ms = 4000ms**

**Total Time per Query = Req.T+Resp.T+E.T**

**= 1 ms + 1 ms + 1 ms = 3ms**

**1000 Queries = 3000ms**

**Java**

**Application Database**

**Request Time**

**Response Time**

**Execution Time**

**101**



**To overcome these problems, we should go for Batch updates. We can group all related SQL**

**Queries into a single batch and we can send that batch at a time to the database.**

**With Simple Statement Batch Updates:**

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

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

**= 2002ms**

**With PreparedStatement Batch Updates:**

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

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

**= 1002ms**

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

**1. We can reduce network traffic**

**2. We can improve performance.**

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

**1. public void addBatch(String sqlQuery)**

**To add query to batch**

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

**to execute a batch of sql queries**

**We can implement batch updates either by simple Statement or by PreparedStatement**

**Program to Demonstrate Batch Updates with Simple Statement**

**1) import java.sql.\*;**

**2) public class BatchUpdatesDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) Statement st = con.createStatement();**

**Query - 1**

**Query - 2**

**:**

**:**

**:**

**Query - n**

**Database**

**Java App**

**Batch of Queries**

**102**



**8) //st.addBatch("select \* from employees");**

**9) st.addBatch("insert into employees values(600,'Mallika',6000,'Chennai')");**

**10) st.addBatch("update employees set esal=esal+1000 where esal<4000");**

**11) st.addBatch("delete from employees where esal>5000");**

**12) int[] count=st.executeBatch();**

**13) int updateCount=0;**

**14) for(int x: count)**

**15) {**

**16) updateCount=updateCount+x;**

**17) }**

**18) System.out.println("The number of rows updated :"+updateCount);**

**19) con.close();**

**20) }**

**21) }**

**Program to Demonstrate Batch Updates with PreparedStatement**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) public class BatchUpdatesDemo2**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**8) PreparedStatement pst = con.prepareStatement("insert into employees values(?,?,?,?**

**)");**

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

**10) while(true)**

**11) {**

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

**13) int eno=sc.nextInt();**

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

**15) String ename=sc.next();**

**16) System.out.println("Employee Sal:");**

**17) double esal=sc.nextDouble();**

**18) System.out.println("Employee Address:");**

**19) String eaddr=sc.next();**

**20) pst.setInt(1,eno);**

**21) pst.setString(2,ename);**

**22) pst.setDouble(3,esal);**

**23) pst.setString(4,eaddr);**

**24) pst.addBatch();**

**25) System.out.println("Do U want to Insert one more record[Yes/No]:");**

**26) String option = sc.next();**

**27) if(option.equalsIgnoreCase("No"))**

**28) {**

**29) break;**

**30) }**

**103**



**31) }**

**32) pst.executeBatch();**

**33) System.out.println("Records inserted Successfully");**

**34) con.close();**

**35) }**

**36) }**

**Advantages of Batch Updates:**

**1. Network traffic will be reduced**

**2. Performance will be improved**

**Limitations of Batch updates:**

**1. We can use Batch Updates concept only for non-select queries. If we are trying to use for select**

**queries then we will get RE saying BatchUpdateException.**

**2. In batch if one sql query execution fails then remaining sql queries wont be executed.**

**\*\*\*\*\*Q: In JDBC How Many Execute Methods Are Avaialble?**

**In total there are 4 methods are available**

**1. executeQuery()** ➔ **For select queries**

**2. executeUpdate()** ➔ **For non-select queries(insert|delete|update)**

**3. execute()**

➔ **For both select and non-select queries**

➔ **For calling Stored Procedures**

**4. executeBatch()**➔ **For Batch Updates**

**104**



**Handling Date Values For Database Operations**

**Sometimes as the part of programing requirement,we have to insert and retrieve Date like**

**DOB,DOJ,DOM,DOP...wrt database.**

**It is not recommended to maintain date values in the form of String,b'z comparisons will become**

**difficult.**

**In Java we have two Date classes**

**1. java.util.Date**

**2. java.sql.Date**

**java.sql.Date is the child class of java.util.Date.**

**java.sql.Date is specially designed class for handling Date values wrt database.**

**Otherthan database operations,if we want to represent Date in our java program then we should**

**go for java.util.Date.**

**java.util.Date can represent both Date and Time where as java.sql.Date represents only Date but**

**not time.**

**1) class Test**

**2) {**

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

**4) {**

**5) java.util.Date udate=new java.util.Date();**

**6) System.out.println("util Date:"+udate);**

**7) long l =udate.getTime();**

**8) java.sql.Date sdate= new java.sql.Date(l);**

**9) System.out.println("sql Date:"+sdate);**

**10) }**

**11) }**

**util Date:Mon Mar 20 19:07:29 IST 2017**

**sql Date:2017-03-20**

**java.util.Date**

**java.sql.Date**

**105**



**Differences between *java.util.Date* and *java.sql.Date***

**java.util.Date java.sql.Date**

**1) It is general Utility Class to handle Dates in**

**our Java Program.**

**1) It is specially designed Class to handle Dates**

**w.r.t DB Operations.**

**2) It represents both Data and Tieme. 2) It represents only Date but not Time.**

**Note: In sql package Time class is availble to represent Time values and TimeStamp class is**

**available to represent both Date and Time.**

**Inserting Date Values into Database:**

**Various databases follow various styles to represent Date.**

**Eg:**

**Oracle: dd-MMM-yy 28-May-90**

**MySQL: yyyy-mm-dd 1990-05-28**

**If we use simple Statement object to insert Date values then we should provide Date value in the**

**database supported format,which is difficult to the programmer.**

**If we use PreparedStatement,then we are not required to worry about database supported form,**

**just we have to call**

**pst.setDate (2, java.sql.Date);**

**This method internally converts date value into the database supported format.**

**Hence it is highly recommendedto use PreparedStatement to insert Date values into database.**

**Steps to insert Date value into Database:**

**DB: create table users(name varchar2(10),dop date);**

**1. Read Date from the end user(in String form)**

**System.out.println("Enter DOP(dd-mm-yyyy):");**

**String dop=sc.next();**

**2. Convert date from String form to java.util.Date form by using SimpleDateFormat object.**

**SDF sdf= new SDF("dd-MM-yyyy");**

**java.util.Date udate=sdf.parse(dop);**

**3. convert date from java.util.Date to java.sql.Date**

**long l = udate.getTime();**

**106**



**java.sql.Date sdate=new java.sql.Date(l);**

**4. set sdate to query**

**pst.setDate(2,sdate);**

**Program To Demonstrate Inserting Date Values Into Database:**

**DB: create table users(name varchar2(10),dop date);**

**DateInsertDemo.java**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) import java.text.\*;**

**4) public class DateInsertDemo**

**5) {**

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

**7) {**

**8) String driver="oracle.jdbc.OracleDriver";**

**9) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**10) String user="scott";**

**11) String pwd="tiger";**

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

**End User provided**

**Date (In String Form)**

**java.util.Date**

**java.sql.Date**

**PS.setDate (1, sdate)**

**Database specific**

**form**

**By using SimpleDateFormat parse()**

**107**



**13) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

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

**15) System.out.println("Enter Person Name:");**

**16) String uname=sc.next();**

**17) System.out.println("Enter DOP(dd-mm-yyyy):");**

**18) String dop=sc.next();**

**19)**

**20) SimpleDateFormat sdf= new SimpleDateFormat("dd-MM-yyyy");**

**21) java.util.Date udate=sdf.parse(dop);**

**22) long l = udate.getTime();**

**23) java.sql.Date sdate= new java.sql.Date(l);**

**24) String sqlQuery="insert into users values(?,?)";**

**25) PreparedStatement ps = con.prepareStatement(sqlQuery);**

**26) ps.setString(1,uname);**

**27) ps.setDate(2,sdate);**

**28) int rc =ps.executeUpdate();**

**29) if(rc==0)**

**30) System.out.println("Record Not inserted");**

**31) else**

**32) System.out.println("Record inserted");**

**33)**

**34) con.close();**

**35) }**

**36) }**

**\*\*\*Note: If end user provides Date in the form of "yyyy-MM-dd" then we can convert directly**

**that String into java.sql.Date form as follows...**

**String s = "1980-05-27";**

**java.sql.Date sdate=java.sql.Date.valueOf(s);**

**Program To Demonstrate Inserting Date Values Into Database:**

**DB: create table users(name varchar2(10),dop date);**

**DateInsertDemo1.java**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) import java.text.\*;**

**4) public class DateInsertDemo1**

**5) {**

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

**7) {**

**8) String driver="oracle.jdbc.OracleDriver";**

**9) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**10) String user="scott";**

**11) String pwd="tiger";**

**108**



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

**13) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

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

**15) System.out.println("Enter Person Name:");**

**16) String uname=sc.next();**

**17) System.out.println("Enter DOP(yyyy-MM-dd):");**

**18) String dop=sc.next();**

**19)**

**20) java.sql.Date sdate=java.sql.Date.valueOf(dop);**

**21) String sqlQuery="insert into users values(?,?)";**

**22) PreparedStatement ps = con.prepareStatement(sqlQuery);**

**23) ps.setString(1,uname);**

**24) ps.setDate(2,sdate);**

**25) int rc =ps.executeUpdate();**

**26) if(rc==0)**

**27) System.out.println("Record Not inserted");**

**28) else**

**29) System.out.println("Record inserted");**

**30)**

**31) con.close();**

**32) }**

**33) }**

**Retrieving Date values from the database:**

**For this we can use either simple Statement or PreparedStatement.**

**The retrieved Date values are Stored in ResultSet in the form of "java.sql.Date" and we can get**

**this value by using getDate() method.**

**Once we got java.sql.Date object,we can format into our required form by using**

**SimpleDateFormat object.**

**Database**

**java.sql.Date**

**Our required String**

**Form**

**s.date = rs.getDate(2);**

**String s = sdf.format (sdate);**

**109**



**Program To Retrieve Date Values From The Database:**

**1) import java.sql.\*;**

**2) import java.text.\*;**

**3) public class DateRetriveDemo**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

**8) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**9) String user="scott";**

**10) String pwd="tiger";**

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

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) PreparedStatement ps = con.prepareStatement("select \* from users6");**

**14) ResultSet rs =ps.executeQuery();**

**15) SimpleDateFormat sdf=new SimpleDateFormat("dd-MMM-yyyy");**

**16) while(rs.next())**

**17) {**

**18) String name=rs.getString(1);**

**19) java.sql.Date sdate=rs.getDate(2);**

**20) String s = sdf.format(sdate);**

**21) System.out.println(name+"..."+s);**

**22) }**

**23) con.close();**

**24) }**

**25) }**

**FAQs:**

**1. In Java how many Date classes are available?**

**2. What is the difference Between java.util.Date and java.sql.Date?**

**3. What is the relation Between java.util.Date and java.sql.Date?**

**4.How to perform the following conversions?**

**1. java.util.Date to java.sql.Date**

**2. String to Date**

**3. Date to String**

**Note: SimpleDateFormat class present in java.text package.**

**sdf.format (date)**

**sdf.parse (string)**

**String Form Date Form**

**110**



**Working with Large Objects (BLOB And CLOB)**

**Sometimes as the part of programming requirement,we have to insert and retrieve large files like**

**images,video files,audio files,resume etc wrt database.**

**Eg:**

**upload image in matrinomial web sites**

**upload resume in job related web sites**

**To store and retrieve large information we should go for Large Objects(LOBs).**

**There are 2 types of Large Objects.**

**1. Binary Large Object (BLOB)**

**2. Character Large Object (CLOB)**

**1) Binary Large Object (BLOB)**

**A BLOB is a collection of binary data stored as a single entity in the database.**

**BLOB type objects can be images,video files,audio files etc..**

**BLOB datatype can store maximum of "4GB" binary data.**

**2) CLOB (Character Large Objects):**

**A CLOB is a collection of Character data stored as a single entity in the database.**

**CLOB can be used to store large text documents(may plain text or xml documents)**

**CLOB Type can store maximum of 4GB data.**

**Eg: hydhistory.txt**

**Steps to insert BLOB type into database:**

**1. create a table in the database which can accept BLOB type data.**

**create table persons(name varchar2(10),image BLOB);**

**2. Represent image file in the form of Java File object.**

**File f = new File("katrina.jpg");**

**3. Create FileInputStream to read binary data represented by image file**

**FileInputStream fis = new FileInputStream(f);**

**111**



**4. Create PreparedStatement with insert query.**

**PreparedStatement pst = con.prepareStatement("insert into persons values(?,?)");**

**5. Set values to positional parameters.**

**pst.setString(1,"katrina");**

**To set values to BLOB datatype, we can use the following method: setBinaryStream()**

**public void setBinaryStream(int index,InputStream is)**

**public void setBinaryStream(int index,InputStream is,int length)**

**public void setBinaryStream(int index,InputStream is,long length)**

**Eg:**

**pst.setBinaryStream(2,fis);** ➔ **Oracle 11g**

**pst.setBinaryStream(2,fis,(int)f.length());** ➔ **Oracle 10g**

**6. execute sql query**

**pst.executeUpdate();**

**Program to Demonstrate insert BLOB type into database:**

**DB: create table persons(name varchar2(10),image BLOB);**

**BLOBDemo1.java**

**1) import java.sql.\*;**

**2) import java.io.\*;**

**3) public class BLOBDemo1**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

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

**9) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**10) String user="scott";**

**11) String pwd="tiger";**

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) String sqlQuery="insert into persons values(?,?)";**

**14) PreparedStatement ps = con.prepareStatement(sqlQuery);**

**15) ps.setString(1,"Katrina");**

**16) File f = new File("katrina.jpg");**

**17) FileInputStream fis = new FileInputStream(f);**

**18) ps.setBinaryStream(2,fis);**

**19) System.out.println("inserting image from :"+f.getAbsolutePath());**

**20) int updateCount=ps.executeUpdate();**

**21) if(updateCount==1)**

**112**



**22) {**

**23) System.out.println("Record Inserted");**

**24) }**

**25) else**

**26) {**

**27) System.out.println("Record Not Inserted");**

**28) }**

**29)**

**30) }**

**31) }**

**Retrieving BLOB Type from Database:**

**We can use either simple Statement or PreparedStatement.**

**Source File**

**(10 KB)**

**Destination File**

**(10 KB)**

**Without buffering 10 \* 1024 read & write Operations are required**

**Because of Buffer we have to perform only 10 Read Operations & 10 Write Operations**

**Source File**

**(10 KB)**

**Destination File**

**(10 KB)**

**Buffer**

**byte[]**

**1 KB**

**113**



**Steps to Retrieve BLOB type from Database**

**1. Prepare ResultSet object with BLOB type**

**RS rs = st.executeQuery("select \* from persons");**

**2. Read Normal data from ResultSet**

**String name=rs.getString(1);**

**3. Get InputStream to read binary data from ResultSet**

**InputStream is = rs.getBinaryStream(2);**

**4. Prepare target resource to hold BLOB data by using FileOutputStream**

**FOS fos = new FOS("katrina\_new.jpg");**

**5. Read Binary Data from InputStream and write that Binary data to output Stream.**

**Program to to Retrieve BLOB type from Database:**

**1) import java.sql.\*;**

**2) import java.io.\*;**

**3) public class BLOBDemo2**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

**int i = is. read();**

**while (i != -1)**

**{**

**fos.write(i);**

**i = is.read();**

**}**

**byte[] buffer = new byte[1024];**

**while (is.read(buffer)>0)**

**{**

**fos.write(buffer);**

**}**

**Without Buffer With Buffer**

**Database**

**Image**

**byte[]**

**Buffer**

**InputStream**

**InputStream is =**

**rs.getBinaryStream();**

**is.read(buffer);**

**Target**

**File**

**FOS**

**fos.write(byte[])**

**114**



**8) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**9) String user="scott";**

**10) String pwd="tiger";**

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

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) PreparedStatement ps = con.prepareStatement("select \* from persons");**

**14) ResultSet rs =ps.executeQuery();**

**15) FileOutputStream os = new FileOutputStream("katrina\_sat.jpeg");**

**16) if(rs.next())**

**17) {**

**18) String name=rs.getString(1);**

**19) InputStream is = rs.getBinaryStream(2);**

**20) byte[] buffer = new byte[2048];**

**21) while(is.read(buffer)>0)**

**22) {**

**23) os.write(buffer);**

**24) }**

**25) os.flush();**

**26) System.out.println("image is available in :katrina\_sat.jpeg");**

**27) }**

**28) con.close();**

**29) }**

**30) }**

**CLOB (Character Large Objects):**

**A CLOB is a collection of Character data stored as a single entity in the database.**

**CLOB can be used to store large text documents(may plain text or xml documents)**

**CLOB Type can store maximum of 4GB data.**

**Eg: hydhistory.txt**

**Steps to insert CLOB type file in the database:**

**All steps are exactly same as BLOB, except the following differences**

**1. Instead of FileInputStream, we have to take FileReader.**

**2. Instead of setBinaryStream() method we have to use setCharacterStream() method.**

**public void setCharacterStream(int index,Reader r) throws SQLException**

**public void setCharacterStream(int index,Reader r,int length) throws SQLException**

**public void setCharacterStream(int index,Reader r,long length) throws SQLException**

**115**



**Program to insert CLOB type file in the database:**

**DB: create table cities(name varchar2(10),history CLOB);**

**CLOBDemo1.java**

**1) import java.sql.\*;**

**2) import java.io.\*;**

**3) public class CLOBDemo1**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

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

**9) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**10) String user="scott";**

**11) String pwd="tiger";**

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) String sqlQuery="insert into cities values(?,?)";**

**14) PreparedStatement ps = con.prepareStatement(sqlQuery);**

**15) ps.setString(1,"Hyderabad");**

**16) File f = new File("hyd\_history.txt");**

**17) FileReader fr = new FileReader(f);**

**18) ps.setCharacterStream(2,fr);**

**19) System.out.println("file is inserting from :"+f.getAbsolutePath());**

**20) int updateCount=ps.executeUpdate();**

**21) if(updateCount==1)**

**22) {**

**23) System.out.println("Record Inserted");**

**24) }**

**25) else**

**26) {**

**27) System.out.println("Record Not Inserted");**

**28) }**

**29)**

**30) }**

**31) }**

**Retrieving CLOB Type from Database:**

**All steps are exactly same as BLOB, except the following differences..**

**1. Instead of using FileOutputStream,we have to use FileWriter**

**2.Instead of using getBinaryStream() method we have to use getCharacterStream() method**

**116**



**Program For Retrieving CLOB Type from Database:**

**1) import java.sql.\*;**

**2) import java.io.\*;**

**3) public class CLOBDemo2**

**4) {**

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

**6) {**

**7) String driver="oracle.jdbc.OracleDriver";**

**8) String jdbc\_url="jdbc:oracle:thin:@localhost:1521:XE";**

**9) String user="scott";**

**10) String pwd="tiger";**

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

**12) Connection con = DriverManager.getConnection(jdbc\_url,user,pwd);**

**13) PreparedStatement ps = con.prepareStatement("select \* from cities");**

**14) ResultSet rs =ps.executeQuery();**

**15) FileWriter fw = new FileWriter("output\_sat.txt");**

**16) if(rs.next())**

**17) {**

**18) String name=rs.getString(1);**

**19) Reader r = rs.getCharacterStream(2);**

**20) /\*char[] buffer = new char[1024];**

**21) while(r.read(buffer)>0)**

**22) {**

**23) fw.write(buffer);**

**24) }\*/**

**25) int i=r.read();**

**26) while(i != -1)**

**27) {**

**28) fw.write(i);**

**29) i = r.read();**

**30) }**

**31) fw.flush();**

**32) System.out.println("Retrieved Successfully file :output\_sat.txt");**

**33) }**

**34) con.close();**

**35) }**

**36) }**

**Q. What is the difference between BLOB and CLOB?**

**We can use BLOB Type to represent binary information like images, video files, audio files etc**

**Where as we can use CLOB Type to represent Character data like text file, xml file etc...**

**117**



**Assignment for Inserting and Retrieving Date, BLOB and CLOB type data:**

**Create a table named with jobseeker and insert data and retrieve data**

**jobseeker**

**(name varchar2(20),dob Date,image BLOB,resume CLOB);**

**name="elanza";**

**dob="28-05-1968";**

**image="elanza.jpg";**

**resume="resume.txt";**

**String** ➔ **udate** ➔ **sdate**

**SDF sdf= new SDF("dd-MM-yyyy");**

**java.util.Date udate= sdf.parse(dob);**

**long l = udate.getTime();**

**java.sql.Date sdate=new java.sql.Date(l);**

**FIS fis = new FIS("elanza.jpg");**

**FR fr= new FR("resume.txt");**

**PS pst=con.pS("insert into jobseeker values(?,?,?,?)");**

**pst.setString(1,name);**

**pst.setDate(2,sdate);**

**pst.setBinaryStream(3,fis);**

**pst.setCharacterStream(4,fr);**

**pst.executeUpdate();**

**=================================**

**FOS fos= new FOS("updatedimage.jpg");**

**PW pw= new PW("updatedresume.txt");**

**SDF sdf= new SDF("dd-MM-yyyy");**

**PS pst=con.PS("select \* from jobseeker");**

**RS rs = pst.executeQuery();**

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

**2) {**

**3) //reading name**

**4) String name=rs.getString(1);**

**5) //reading dob**

**6) java.sql.Date sdate=rs.getDate(2);**

**7) String dob=sdf.format(sdate);**

**8) //reading BLOB(image)**

**9) InputStream is = rs.getBinaryStream(3);**

**10) byte[] b = new byte[1024];**

**11) while(is.read(b)>0)**

**12) {**

**13) fos.write(b);**

**14) }**

**15) fos.flush();**

**16) //reading CLOB(txt file)**

**118**



**17) Reader r = rs.getCharacterStream(4);**

**18) char[] ch = new char[1024];**

**19) while(r.read(ch)>0)**

**20) {**

**21) pw.write(ch);**

**22) }**

**23) pw.flush();**

**24) }**

**119**



**Connection Pooling**

**If we required to communicate with database multiple times then it is not recommended to create**

**separate Connection object every time, b'z creating and destroying Connection object every time**

**creates performance problems.**

**To overcome this problem, we should go for Connection Pool.**

**Connection Pool is a pool of already created Connection objects which are ready to use.**

**If we want to communicate with database then we request Connection pool to provide**

**Connection. Once we got the Connection, by using that we can communicates with database. After**

**completing our work, we can return Connection to the pool instead of destroying.**

**Hence the main advantage of Connection Pool is we can reuse same Connection object multiple**

**times, so that overall performance of application will be improved.**

**Process to implement Connection Pooling:**

**1. Creation of DataSource object**

**DataSource is responsible to manage connections in Connection Pool.**

**DataSource is an interface present in javax.sql package.**

**Driver Software vendor is responsible to provide implementation.**

**Oracle people provided implementation class name is :**

**OracleConnectionPoolDataSource.**

**This class present inside oracle.jdbc.pool package and it is the part of ojdbc6.jar.**

**OracleConnectionPoolDataSource ds= new OracleConnectionPoolDataSource();**

**2. Set required JDBC Properties to the DataSource object:**

**ds.setURL("jdbc:oracle:thin:@localhost:1521:XE");**

**ds.setUser("scott");**

**ds.setPassword("tiger");**

**3. Get Connection from DataSource object:**

**Connection con = ds.getConnection();**

**Once we got Connection object then remaining process is as usual.**

**120**



**Program to Demonstrate Connection Pooling for Oracle Database:**

**1) import java.sql.\*;**

**2) import javax.sql.\*;**

**3) import oracle.jdbc.pool.\*;// ojdbc6.jar**

**4) class ConnectionPoolDemoOracle**

**5) {**

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

**7) {**

**8) OracleConnectionPoolDataSource ds = new OracleConnectionPoolDataSource();**

**9) ds.setURL("jdbc:oracle:thin:@localhost:1521:XE");**

**10) ds.setUser("scott");**

**11) ds.setPassword("tiger");**

**12) Connection con=ds.getConnection();**

**13) Statement st =con.createStatement();**

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

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

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

**17) while(rs.next())**

**18) {**

**19) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**20) }**

**21) con.close();**

**22) }**

**23) }**

**Program to Demonstrate Connection Pooling for MySQL Database:**

**1) import java.sql.\*;**

**2) import javax.sql.\*;**

**3) import com.mysql.jdbc.jdbc2.optional.\*;**

**4) class ConnectionPoolDemoMySql**

**5) {**

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

**7) {**

**8) MysqlConnectionPoolDataSource ds = new MysqlConnectionPoolDataSource();**

**9) ds.setURL("jdbc:mysql://localhost:3306/elanzadb");**

**10) ds.setUser("root");**

**11) ds.setPassword("root");**

**12) Connection con=ds.getConnection();**

**13) Statement st =con.createStatement();**

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

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

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

**17) while(rs.next())**

**18) {**

**121**



**19) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**20) }**

**21) con.close();**

**22) }**

**23) }**

**Note: This way of implementing Connection Pool is useful for Standalone applications. In the case**

**of web and enterprise applications, we have to use server level connection pooling. Every web and**

**application server can provide support for Connection Pooling.**

**Q. What is the difference Between getting Connection object by using DriverManager and**

**DataSource object?**

**In the case of DriverManager.getConnection(), always a new Connection object will be created**

**and returned.**

**But in the case of DataSourceObject.getConnection(), a new Connection object won't be created**

**and existing Connection object will be returned from Connection Pool.**

**122**



**Properties**

**In Java Program if anything which changes frequently(like jdbc url, username, pwd etc)is not**

**recommended to hard code in our program.**

**The problem in this approach is if there is any change in java program,to reflect that change we**

**have to recompile,rebuild and redeploy total application and even some times server restart also**

**required,which creates a big business impact to the client.**

**To overcome this problem, we should go for Properties file. The variable things we have to**

**configure in Properties file and we have to read these properties from java program.**

**The main advantage of this approach is if there is any change in Properties file and to reflect that**

**change just redeployment is enough, which won't create any business impact to the client.**

**Program to Demonstrate use of Properties file:**

**db.properties:**

**url= jdbc:oracle:thin:@localhost:1521:XE**

**user= scott**

**pwd= tiger**

**JdbcPropertiesDemo.java:**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) import java.io.\*;**

**4) class JdbcPropertiesDemo**

**5) {**

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

**7) {**

**8) Properties p = new Properties();**

**9) FileInputStream fis = new FileInputStream("db.properties");**

**10) p.load(fis);// to load all properties from properties file into java Properties object**

**11) String url=p.getProperty("url");**

**12) String user=p.getProperty("user");**

**13) String pwd=p.getProperty("pwd");**

**14) Connection con=DriverManager.getConnection(url,user,pwd);**

**15) Statement st =con.createStatement();**

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

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

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

**123**



**19) while(rs.next())**

**20) {**

**21) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**22) }**

**23) con.close();**

**24) }**

**25) }**

**If we change the properties in properties file for mysql database then the program will fetch data**

**from mysql database.**

**db.properties:**

**url= jdbc:mysql://localhost:3306/elanzadb**

**user= root**

**pwd= root**

**Program to Demonstrate use of Properties file:**

**db1.properties:**

**user=scott**

**password=tiger**

**JdbcPropertiesDemo1.java:**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) import java.io.\*;**

**4) class JdbcPropertiesDemo1**

**5) {**

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

**7) {**

**8) Properties p = new Properties();**

**9) FileInputStream fis = new FileInputStream("db1.properties");**

**10) p.load(fis);// to load all properties from properties file into java Properties object**

**11) Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE"**

**,p);**

**12) Statement st =con.createStatement();**

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

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

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

**16) while(rs.next())**

**17) {**

**18) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**19) }**

**124**



**20) con.close();**

**21) }**

**22) }**

**Q. How many getConnection() methods are available in DriverManager class.**

**1. Connection con=DM.getConnection(url,user,pwd);**

**2. Connection con=DM.getConnection(url,Properties);**

**3. Connection con=DM.getConnection(url);**

**Eg:**

**Connection**

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

**Eg:**

**Connection con=DriverManager.getConnection("jdbc:oracle:thin:scott/tiger@localhost:1521:XE");**

**125**



**Transaction Management in JDBC**

**Process of combining all related operations into a single unit and executing on the rule "either all**

**or none", is called transaction management.**

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

**Case-1: Funds Transfer**

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

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

**All operations should be performed as a single unit only. If debit from sender's account completed**

**and credit into receiver's account fails then there may be a chance of data inconsistency problems.**

**Case-2: Movie Ticket Reservation**

**1. Verify the status**

**2. Reserve the tickets**

**3. Payment**

**4. issue tickets.**

**All operations should be performed as a single unit only. If some operations success and some**

**operations fails then there may be data inconsistency problems.**

**Transaction Properties:**

**Every Transaction should follow the following four ACID properties.**

**1. A** ➔ **Atomiticity**

**Either all operations should be done or None.**

**2. C** ➔ **Consistency(Reliabile Data)**

**It ensures bringing database from one consistent state to another consistent state.**

**3. I** ➔ **isolation (Sepatation)**

**Ensures that transaction is isolated from other transactions**

**4. D** ➔ **Durability**

**It means once transaction committed, then the results are permanent even in the case of**

**system restarts, errors etc..**

**126**



**Types of Transactions:**

**There are two types of Transactions**

**1. Local Transactions**

**2. Global Transactions**

**1. Local Transactions:**

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

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

**2. Global Transactions:**

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

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

**banks.**

**Note:**

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

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

**Process of Transaction Management in JDBC:**

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

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

**committed automatically in the database.**

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

**con.setAutoCommit(false);**

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

**con.commit();**

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

**rollback() method.**

**con.rollback();**

**127**



**Program: To demonstrate Transactions**

**1) create table accounts(name varchar2(10),balance number);**

**2)**

**3) insert into accounts values('elanza',100000);**

4) **insert into accounts values('sunny',10000);**

**TransactionDemo1.java**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) public class TransactionDemo1**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**8) Statement st = con.createStatement();**

**9) System.out.println("Data before Transaction");**

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

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

**12) while(rs.next())**

**13) {**

**14) System.out.println(rs.getString(1)+"..."+rs.getInt(2));**

**15) }**

**16) System.out.println("Transaction begins...");**

**17) con.setAutoCommit(false);**

**18) st.executeUpdate("update accounts set balance=balance-**

**10000 where name='elanza'");**

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

**");**

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

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

**22) String option = sc.next();**

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

**24) {**

**25) con.commit();**

**26) System.out.println("Transaction Commited");**

**27) }**

**28) else**

**29) {**

**30) con.rollback();**

**31) System.out.println("Transaction Rolled Back");**

**32) }**

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

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

**35) ResultSet rs1 =st.executeQuery("select \* from accounts");**

**36) while(rs1.next())**

**128**



**37) {**

**38) System.out.println(rs1.getString(1)+"..."+rs1.getInt(2));**

**39) }**

**40) con.close();**

**41) }**

**42) }**

**Savepoint(I):**

**Savepoint is an interface present in java.sql package.**

**Introduced in JDBc 3.0 Version.**

**Driver Software Vendor is responsible to provide implementation.**

**Savepoint concept is applicable only in Transactions.**

**Within a transaction if we want to rollback a particular group of operations based on some**

**condition then we should go for Savepoint.**

**We can set Savepoint by using *setSavepoint()* method of Connection interface.**

**Savepoint sp = con.setSavepoint();**

**To perform rollback operation for a particular group of operations wrt Savepoint, we can use**

**rollback() method as follows.**

**con.rollback(sp);**

**We can release or delete Savepoint by using release Savepoint() method of Connection interface.**

**con.releaseSavepoint(sp);**

**Case Study:**

**con.setAutoCommit(false);**

**Operation-1;**

**Operation-2;**

**Operation-3;**

**Savepoint sp = con.setSavepoint();**

**Operation-4;**

**Operation-5;**

**if(balance<10000)**

**{**

**con.rollback(sp);**

**}**

**else**

**{**

**con.releaseSavepoint(sp);**

**}**

**operation-6;**

**129**



**con.commit();**

**At line-1 if balance <10000 then operations 4 and 5 will be Rollback, otherwise all operations will**

**be performed normally.**

**Program to Demonstrate Savepoint:**

**create table politicians(name varchar2(10),party varchar2(10));**

**1) import java.sql.\*;**

**2) public class SavePointDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) Statement st = con.createStatement();**

**8) con.setAutoCommit(false);**

**9) st.executeUpdate("insert into politicians values ('kcr','trs')");**

**10) st.executeUpdate("insert into politicians values ('babu','tdp')");**

**11) Savepoint sp = con.setSavepoint();**

**12) st.executeUpdate("insert into politicians values ('siddu','bjp')");**

**13) System.out.println("oops ..wrong entry just rollback");**

**14) //con.rollback(sp);**

**15) con.releaseSavepoint(sp);**

**16) //System.out.println("Operations are roll back from Savepoint");**

**17) con.commit();**

**18) con.close();**

**19) }**

**20) }**

**Note:**

**Some drivers won't provide support for Savepoint. Type-1 Driver won't provide support, but Type-**

**4 Driver can provide support.**

**Type-4 Driver of Oracle provide support only for *setSavepoint()* and *rollback()* methods but not for**

***releaseSavepoint()* method.**

**Transaction Concurrency Problems:**

**Whenever multiple transactions are executing concurrently then there may be a chance of**

**transaction concurrency problems.**

**The following are the most commonly occurred concurrency problems.**

**1. Dirty Read Problem**

**2. Non Repeatable Read Problem**

**3. Phantom Read Problem**

**130**



**1. Dirty Read Problem:**

**Also known as uncommitted dependency problem.**

**Before committing the transaction, if its intermediate results used by any other transaction then**

**there may be a chance of Data inconsistency problems. This is called Dirty Read Problem.**

**elanza:50000**

**T1:update accounts set balance=balance+50000 where name='elanza'**

**T2:select balance from accounts where name='elanza'**

**T1: con.rollback();**

**At the end, T1 point of view, elanza has 50000 balance and T2 point of view elanza has 1Lakh. There**

**may be a chance of data inconsistency problem. This is called Dirty Read Problem.**

**2. Non-Repeatable Read Problem:**

**For the same Read Operation, in the same transaction if we get different results at different times,**

**then such type of problem is called Non-Repeatable Read Problem.**

**Eg:**

**T1: select \* from employees;**

**T2: update employees set esal=10000 where ename='elanza';**

**T1: select \* from employees;**

**In the above example Transaction-1 got different results at different times for the same query.**

**3. Phantom Read Problem:**

**A phantom read occurs when one transaction reads all the rows that satisfy a where condition and**

**second transaction insert a new row that satisfy same where condition. If the first transaction**

**reads for the same condition in the result an additional row will come. This row is called phantom**

**row and this problem is called phantom read problem.**

**T1: select \* from employees where esal >5000;**

**T2: insert into employees values(300,'ravi',8000,'hyd');**

**T1: select \* from employees where esal >5000;**

**In the above code whenever transaction-1 performing read operation second time, a new row will**

**come in the result.**

**To overcome these problems we should go for Transaction isolation levels.**

**Connection interface defines the following 4 transaction isolation levels.**

**1. TRANSACTION\_READ\_UNCOMMITTED** ➔ **1**

**2. TRANSACTION\_READ\_COMMITTED** ➔ **2**

**3. TRANSACTION\_REPEATABLE\_READ** ➔ **4**

**4. TRANSACTION\_SERIALIZABLE** ➔ **8**

**131**



**1. TRANSACTION\_READ\_UNCOMMITTED:**

**It is the lowest level of isolation.**

**Before committing the transaction its intermediate results can be used by other transactions.**

**Internally it won't use any locks.**

**It does not prevent Dirty Read Problem, Non-Repeatable Read Problem and Phantom Read**

**Problem.**

**We can use this isolation level just to indicate database supports transactions.**

**This isolation level is not recommended to use.**

**2. TRANSACTION\_READ\_COMMITTED:**

**This isolation level ensures that only committed data can be read by other transactions.**

**It prevents Dirty Read Problem. But there may be a chance of Non Repeatable Read Problem and**

**Phantom Read Problem.**

**3. TRANSACTION\_REPEATABLE\_READ:**

**This is the default value for most of the databases. Internally the result of SQL Query will be locked**

**for only one transaction. If we perform multiple read operations, then there is a guarantee that for**

**same result.**

**It prevents Dirty Read Problem and Non Repeatable Read Problems. But still there may be a**

**chance of Phantom Read Problem.**

**4. TRANSACTION\_SERIALIZABLE:**

**It is the highest level of isolation.**

**The total table will be locked for one transaction at a time.**

**It prevents Dirty Read, Non-Repeatable Read and Phantom Read Problems.**

**Not Recommended to use because it may creates performance problems.**

**Connection interface defines the following method to know isolation level.**

**getTransactionIsolation()**

**Connection interface defines the following method to set our own isolation level.**

**setTransactionIsolation(int level)**

**132**



**Eg:**

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

**con.setTransactionIsolation(8);**

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

**Note:**

**For Oracle database, the default isolation level is: 2(TRANSACTION\_READ\_COMMITED).**

**Oracle database provides support only for isolation levels 2 and 8.**

**For MySql database, the default isolation level is: 4(TRANSACTION\_REPEATABLE\_READ).**

**MySql database can provide support for all isolation levels (1, 2, 4 and 8).**

**Program to demonstrate Oracle database Isolation levels:**

**1) import java.sql.\*;**

**2) public class TransactionDemo2**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) System.out.println(con.getTransactionIsolation());//2**

**8) con.setTransactionIsolation(8);**

**9) System.out.println(con.getTransactionIsolation());**

**10)**

**11) }**

**12) }**

**Program to demonstrate MySQL database Isolation levels:**

**1) import java.sql.\*;**

**2) public class TransactionDemo3**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/elanzad**

**b","root","root");**

**7) System.out.println(con.getTransactionIsolation());//4**

**8) con.setTransactionIsolation(8);**

**9) System.out.println(con.getTransactionIsolation());//8**

**10)**

**11) }**

**12) }**

**133**



**Q. In JDBC how many transaction isolation levels are defined?**

**The following 5 isolation levels are defined.**

**1) TRANSACTION\_NONE** ➔ **0**

**It indicates that database won't provide support for transactions.**

**2) TRANSACTION\_READ\_UNCOMMITTED** ➔ **1**

**3) TRANSACTION\_READ\_COMMITTED** ➔ **2**

**4) TRANSACTION\_REPEATABLE\_READ** ➔ **4**

**5) TRANSACTION\_SERIALIZABLE** ➔ **8**

**Summary Table of Isolation Levels**

**Isolation**

**Level**

**Is Dirty**

**Problem**

**Prevented?**

**Is Non**

**Repeatable**

**Read**

**Problem**

**Prevented?**

**Is Phantom**

**Read Problem**

**Prevented?**

**TRANSACTION\_READ\_**

**UNCOMMITTED**

**No No No**

**TRANSACTION\_READ\_**

**COMMITTED**

**Yes No No**

**TRANSACTION\_**

**REPEATABLE\_READ**

**Yes Yes No**

**TRANSACTION\_**

**SERIALIZABLE**

**Yes Yes Yes**

**134**



**MetaData**

**Metadata means data about data. I.e. Metadata provides extra information about our original**

**data.**

**Eg:**

**Metadata about database is nothing but database product name, database version etc..**

**Metadata about ResultSet means no of columns, each column name, column type etc..**

**JDBC provides support for 3 Types of Metadata**

**1. DatabaseMetaData**

**2. ResultSetMetaData**

**3. ParameterMetaData**

**1. DatabaseMetaData**

**It is an interface present in java.sql package.**

**Driver Software vendor is responsible to provide implementation.**

**We can use DatabaseMetaData to get information about our database like database product**

**name, driver name, version, number of tables etc..**

**We can also use DatabaseMetaData to check whether a particular feature is supported by DB or**

**not like stored procedures, full joins etc..**

**We can get DatabaseMetaData object by using getMetaData() method of Connection interface.**

**public DatabaseMetaData getMetaData();**

**Eg: DatabaseMetaData dbmd=con.getMetaData();**

**Once we got DatabaseMetaData object we can call several methods on that object like**

**getDatabaseProductName()**

**getDatabaseProductVersion()**

**getMaxColumnsInTable()**

**supportsStoredProcedures()**

**etc...**

**135**



**App1: Program to display Database meta information by using DataBaseMetaData**

**1) import java.sql.\*;**

**2) class DatabaseMetaDataDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) DatabaseMetaData dbmd=con.getMetaData();**

**8) System.out.println("Database Product Name:"+dbmd.getDatabaseProductName());**

**9) System.out.println("DatabaseProductVersion:"+dbmd.getDatabaseProductVersion());**

**10) System.out.println("DatabaseMajorVersion:"+dbmd.getDatabaseMajorVersion());**

**11) System.out.println("DatabaseMinorVersion:"+dbmd.getDatabaseMinorVersion());**

**12) System.out.println("JDBCMajorVersion:"+dbmd.getJDBCMajorVersion());**

**13) System.out.println("JDBCMinorVersion:"+dbmd.getJDBCMinorVersion());**

**14) System.out.println("DriverName:"+dbmd.getDriverName());**

**15) System.out.println("DriverVersion:"+dbmd.getDriverVersion());**

**16) System.out.println("URL:"+dbmd.getURL());**

**17) System.out.println("UserName:"+dbmd.getUserName());**

**18) System.out.println("MaxColumnsInTable:"+dbmd.getMaxColumnsInTable());**

**19) System.out.println("MaxRowSize:"+dbmd.getMaxRowSize());**

**20) System.out.println("MaxStatementLength:"+dbmd.getMaxStatementLength());**

**21) System.out.println("MaxTablesInSelect"+dbmd.getMaxTablesInSelect());**

**22) System.out.println("MaxTableNameLength:"+dbmd.getMaxTableNameLength());**

**23) System.out.println("SQLKeywords:"+dbmd.getSQLKeywords());**

**24) System.out.println("NumericFunctions:"+dbmd.getNumericFunctions());**

**25) System.out.println("StringFunctions:"+dbmd.getStringFunctions());**

**26) System.out.println("SystemFunctions:"+dbmd.getSystemFunctions());**

**27) System.out.println("supportsFullOuterJoins:"+dbmd.supportsFullOuterJoins());**

**28) System.out.println("supportsStoredProcedures:"+dbmd.supportsStoredProcedures());**

**29) con.close();**

**30) }**

**31) }**

**App2: Program to display Table Names present in Database by using DataBaseMetaData**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class DatabaseMetaDataDemo2**

**4) {**

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

**6) {**

**7) int count=0;**

**8) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**9) DatabaseMetaData dbmd=con.getMetaData();**

**136**



**10) String catalog=null;**

**11) String schemaPattern=null;**

**12) String tableNamePattern=null;**

**13) String[] types=null;**

**14) ResultSet rs = dbmd.getTables(catalog,schemaPattern,tableNamePattern,types);**

**15) //the parameters can help limit the number of tables that are returned in the ResultSe**

**t**

**16) //the ResultSet contains 10 columns and 3rd column represent table names.**

**17) while(rs.next())**

**18) {**

**19) count++;**

**20) System.out.println(rs.getString(3));**

**21) }**

**22) System.out.println("The number of tables:"+count);**

**23) con.close();**

**24) }**

**25) }**

**Note: Some driver softwares may not capture complete information. In that case we will get**

**default values like zero.**

**Eg: getMaxRowSize()** ➔ **0**

**ResultSetMetaData:**

**It is an interface present in java.sql package.**

**Driver software vendor is responsible to provide implementation.**

**It provides information about database table represented by ResultSet object.**

**Useful to get number of columns, column types etc..**

**We can get ResultSetMetaData object by using getMetaData() method of ResultSet interface.**

**public ResultSetMetaData getMetaData()**

**Eg: ResultSetMetaData rsmd=rs.getMetaData();**

**Once we got ResultSetMetaData object, we can call the following methods on that object like**

**getColumnCount()**

**getColumnName()**

**getColumnType()**

**etc...**

**App3: Program to display Columns meta information by using ResultMetaData**

**1) import java.sql.\*;**

**2) class ResultSetMetaDataDemo**

**3) {**

**137**



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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) Statement st = con.createStatement();**

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

**9) ResultSetMetaData rsmd=rs.getMetaData();**

**10) int count=rsmd.getColumnCount();**

**11) for(int i=1;i<= count;i++)**

**12) {**

**13) System.out.println("Column Number:"+i);**

**14) System.out.println("Column Name:"+rsmd.getColumnName(i));**

**15) System.out.println("Column Type:"+rsmd.getColumnType(i));**

**16) System.out.println("Column Size:"+rsmd.getColumnDisplaySize(i));**

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

**18) }**

**19) con.close();**

**20) }**

**21) }**

**App3: Program to display Table Data including Column Names by using ResultMetaData**

**1) import java.sql.\*;**

**2) class ResultSetMetaDataDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) Statement st = con.createStatement();**

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

**9) ResultSetMetaData rsmd=rs.getMetaData();**

**10) String col1=rsmd.getColumnName(1);**

**11) String col2=rsmd.getColumnName(2);**

**12) String col3=rsmd.getColumnName(3);**

**13) String col4=rsmd.getColumnName(4);**

**14) System.out.println(col1+"\t"+col2+"\t"+col3+"\t"+col4);**

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

**16) while(rs.next())**

**17) {**

**18) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get**

**String(4));**

**19) }**

**20) }**

**21) }**

**138**



**ParameterMetaData (I):**

**It is an interface and present in java.sql package.**

**Driver Software vendor is responsible to provide implementation.**

**In General we can use positional parameters(?) while creating PreparedStatement object.**

**PreparedStatement pst=con.prepareStatement("insert into employees values(?,?,?,?)");**

**We can use ParameterMetaData to get information about positional parameters like parameter**

**count,parameter mode, and parameter type etc...**

**We can get ParameterMetaData object by using getParameterMetaData() method of**

**PreparedStatement interface.**

**ParameterMetaData pmd=pst.getParameterMetaData();**

**Once we got ParameterMetaData object, we can call several methods on that object like**

**1. int getParameterCount()**

**2. int getParameterMode(int param)**

**3. int getParameterType(int param)**

**4. String getParameterTypeName(int param)**

**etc..**

**Important Methods of ParameterMetaData:**

**1. int getParameterCount()**

**Retrieves the number of parameters in the PreparedStatement object for which this**

**ParameterMetaData object contains information.**

**2.int getParameterMode(int param)**

**Retrieves the designated parameter's mode.**

**3. int getParameterType(int param)**

**Retrieves the designated parameter's SQL type.**

**4. String getParameterTypeName(int param)**

**Retrieves the designated parameter's database-specific type name.**

**5. int getPrecision(int param)**

**Retrieves the designated parameter's specified column size.**

**6. int getScale(int param)**

**Retrieves the designated parameter's number of digits to right of the decimal point.**

**139**



**7. int isNullable(int param)**

**Retrieves whether null values are allowed in the designated parameter.**

**8. boolean isSigned(int param)**

**Retrieves whether values for the designated parameter can be signed numbers.**

**App14: Program to display Parameter meta information by using ParameterMetaData**

**1) import java.sql.\*;**

**2) class ParameterMetaDataDemo**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","system","elanza");**

**7) PreparedStatement pst = con.prepareStatement("insert into employees values(?,?,?,?)**

**");**

**8) ParameterMetaData pmd=pst.getParameterMetaData();**

**9) int count=pmd.getParameterCount();**

**10) for(int i=1;i<= count;i++)**

**11) {**

**12) System.out.println("Parameter Number:"+i);**

**13) System.out.println("Parameter Mode:"+pmd.getParameterMode(i));**

**14) System.out.println("Parameter Type:"+pmd.getParameterType(i));**

**15) System.out.println("Parameter Precision:"+pmd.getPrecision(i));**

**16) System.out.println("Parameter Scale:"+pmd.getScale(i));**

**17) System.out.println("Parameter isSigned:"+pmd.isSigned(i));**

**18) System.out.println("Parameter isNullable:"+pmd.isNullable(i));**

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

**20) }**

**21) con.close();**

**22) }**

**23) }**

**Note: Most of the drivers won't provide support for ParameterMetaData.**

**140**



**JDBC with Excel Sheets**

**1) We can read and write Data w.r.t Excel Sheet by using JDBC.**

**To work with Excel it is recommended to use Type - 1 Driver.**

**While configuring DSN we have to specify Driver Name as "Driver do Microsoft Excel"**

**2) We have to browse our Excel File by using "Select Work Book" Button.**

**3) Each Excel Work Book contains several Sheets.**

**While writing Query we have to specify Sheet Name also.**

**ResultSet rs=st.executeQuery("select \* from [Sheet1$]");**

**Program to Read Data From Excel Sheet by using JDBC:**

**1) import java.sql.\*;**

**2) public class ExcelDemo1**

**3) {**

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

**5) {**

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

**7) Connection con = DriverManager.getConnection("jdbc:odbc:demoexcel11");**

**8) Statement st = con.createStatement();**

**9) ResultSet rs=st.executeQuery("select \* from [Sheet1$]");**

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

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

**12) while(rs.next())**

**13) {**

**141**



**14) System.out.println(rs.getInt(1)+"..."+rs.getString(2)+"..."+rs.getFloat(3)+"..."+rs.get**

**String(4));**

**15) }**

**16) con.close();**

**17) }**

**18) }**

**Progaram to read data from excel and copy into Oracle Database:**

**1) import java.sql.\*;**

**2) public class ExcelDemo2**

**3) {**

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

**5) {**

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

**7) Connection con = DriverManager.getConnection("jdbc:odbc:demodsnforexcel2");**

**8) Statement st = con.createStatement();**

**9) ResultSet rs=st.executeQuery("select \* from [Sheet1$]");**

**10) Connection con2=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**11) PreparedStatement pst=con2.prepareStatement("insert into employees values(?,?,?,?**

**)");**

**12) while(rs.next())**

**13) {**

**14) pst.setInt(1,rs.getInt(1));**

**15) pst.setString(2,rs.getString(2));**

**16) pst.setFloat(3,rs.getFloat(3));**

**17) pst.setString(4,rs.getString(4));**

**18) pst.executeUpdate();**

**19) }**

**20) System.out.println("Data Inserted Successfully from Excel to Oracle");**

**21) con.close();**

**22) con2.close();**

**23) }**

**24) }**

**142**



**ResultSet Types**

**Division-1:**

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

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

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

**1.Read Only ResultSets:**

**We can perform only read operations on the ResultSet by using corresponding getter methods and**

**we cannot perform any updations.**

**By default ResultSet is Read Only.**

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

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

**2. Updatable ResultSets:**

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

**Updatable ResultSets.**

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

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

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

**Division-2:**

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

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

**2. Scrollable ResultSets**

**1. Forward Only ResultSets:**

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

**sequentially.**

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

**143**



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

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

**2. Scrollable ResultSets:**

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

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

**move to anywhere.**

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

**1. Scroll Insensitive ResultSet**

**2. Scroll Sensitive ResultSet**

**1. Scroll Insensitive ResultSet:**

**After getting ResultSet if we are performing any change in Database and if those changes are not**

**reflecting to the ResultSet, such type of ResultSets are called scroll insensitive ResultSets.**

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

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

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

**2.Scroll sensitive ResultSets:**

**After getting the ResultSet if we perform any change in the database and if those changes are**

**visible to ResultSet, such type of ResultSet is called Scroll sensitive ResultSet.**

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

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

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

**144**



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

**Differnences between Forward only and Scrollable ResultSets**

**How to get Required ResultSet:**

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

**Statement st =con.createStatement(int type,int mode);**

**PreparedStatement pst=con.prepareStatement(query,int type,int mode);**

**Allowed values for type are:**

**ResultSet.TYPE\_FORWARD\_ONLY** ➔ **1003**

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

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

**Allowed values for mode are:**

**ResultSet.CONCUR\_READ\_ONLY** ➔ **1007**

**ResultSet.CONCUR\_UPDATABLE** ➔ **1008**

**Eg: for Scroll sensitive and updatable ResultSet:**

**Statement st =con.createStatement(1005,1008);**

**Scroll Insensitive Scroll Sensitive**

**After getting ResultSet if we perform any**

**updation in the DB then those updation are**

**not visible to the ResultSet i.e. ResultSet is**

**insensitive to DB updations.**

**After getting ResultSet if we perform any**

**updation in the DB then those updation are**

**by default available to the to the ResultSet**

**i.e. ResultSet is sensitive to DB updations.**

**As insensitive ResultSet is like snapshot of**

**Data in DB when Query will be executed.**

**A Sensitive ResultSet doesn't represent**

**snap shot of Data. It contains Pointers to**

**Rows of DB directly, which satisfy Query**

**Condition.**

**Relatively Performance is High. Relatively Performance is low because for**

**get Operation a Trip is required to DB.**

**Non Scrollable (Forward only) Scrollable**

**Cursor can move only in Forward**

**Direction.**

**Cursor can move in both Forward**

**and Backward Direction.**

**This Cursor can't move randomly. Scrollable ResultSet Cursor can move**

**randomly.**

**By using Non Scrollable ResultSet**

**Cursor if we want to move Nth Record**

**(N + 1) Iterations are required.**

**Performance is high because Cursor**

**can move randomly to any Record.**

**145**



**Note: Default type is forward only and default mode is read only.**

**Note:**

**To use various types of ResultSets underlying database support and driver support must be**

**required**

**Some databases and some driver softwares wont provide proper support.**

**We can check whether the database supports a particular type of ResultSet or not by using the**

**following methods of DatabaseMetaData.**

**1. public boolean supportsResultSetConcurrency(int type, int concurrency)**

**Retrieves whether this database supports the given concurrency type in combination with the**

**given result set type.**

**2. public boolean supportsResultSetType(int type)**

**Retrieves whether this database supports the given result set type.**

**Program to Check whether database supports particular type of ResultSet or not**

**1) import java.sql.\*;**

**2) class ResultSetTypeTest**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) DatabaseMetaData dbmd=con.getMetaData();**

**8) System.out.println(dbmd.supportsResultSetConcurrency(1003,1007));**

**9) System.out.println(dbmd.supportsResultSetConcurrency(1003,1008));**

**10) System.out.println(dbmd.supportsResultSetType(1003));**

**11) System.out.println(dbmd.supportsResultSetType(1004));**

**12) System.out.println(dbmd.supportsResultSetType(1005));**

**13) }**

**14) }**

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

**1.rs.next()**

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

**2.rs.getXxx()**

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

**3.rs.getRow()**

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

**146**



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

**1.rs.next()**

**2.rs.getXxx()**

**3.rs.getRow()**

**4.rs.previous()**

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

**to that record position**

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

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

**6.rs.afterLast()**

**moves the cursor after last record position**

**7.rs.first()**

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

**8.rs.last()**

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

**9.rs.absolute(int x)**

**The argument can be either positive or negative.**

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

**If the argument is negative then it will be moved to the specified record position from last.**

**10.rs.relative(int x)**

**The argument can be either positive or negative**

**If the argument is positive then the cursor will move to forward direction to specified number of**

**records from the current position. If the argument is negative then the cursor will move to**

**backward direction to the specified number of records from the current position.**

**11. rs.isFirst()**

**returns true if the cursor is locating at first record position**

**12. rs.isLast()**

**13. rs.isBeforeFirst()**

**14. rs.isAfterLast()**

**15. rs.refreshRow()**

**We can use this method in scroll sensitive ResultSets to update row with latest values from**

**Database.**

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

**absolute() method will always work either from BFR or from ALR.**

**relative() method will work wrt to current position.**

**147**



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

**we have to move backward direction.**

**Note:**

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

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

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

**SCROLLABLE ResultSet**

**1) import java.sql.\*;**

**2) class ResultSetTypesDemo1**

**3) {**

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

**5) {**

**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.C**

**ONCUR\_READ\_ONLY);**

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

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

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

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

**12) while(rs.next())**

**13) {**

**14) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**15) }**

**16) System.out.println("Records in Backword Direction");**

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

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

**19) while(rs.previous())**

**20) {**

**21) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**22) }**

**23) con.close();**

**24) }**

**25) }**

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

**1) import java.sql.\*;**

**2) class ResultSetTypesDemo2**

**3) {**

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

**5) {**

**148**



**6) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**7) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.C**

**ONCUR\_READ\_ONLY);**

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

**9) System.out.println("Records in Original Order");**

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

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

**12) while(rs.next())**

**13) {**

**14) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**15) }**

**16) rs.first();// First Record**

**17) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**18) rs.last();//Last Record**

**19) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**20) rs.relative(-1);//2nd Record from the last**

**21) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**22) rs.absolute(2);//2nd Record**

**23) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**24) con.close();**

**25) }**

**26) }**

**Application-3: Reflecting Database updations by using SCROLL SENSITIVE**

**ResultSet (Type-1 Driver)**

**1) import java.sql.\*;**

**2) class ResultSetTypesDemo3**

**3) {**

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

**5) {**

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

**7) Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");**

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

**9) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

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

**11) System.out.println("Records Before Updation");**

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

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

**14) while(rs.next())**

**15) {**

**149**



**16) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**17) }**

**18) System.out.println("Application is in pausing state,please update database..");**

**19) System.in.read();**

**20) rs.beforeFirst();**

**21) System.out.println("Records After Updation");**

**22) while(rs.next())**

**23) {**

**24) rs.refreshRow();**

**25) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**26) }**

**27) con.close();**

**28) }**

**29) }**

**Note: Very few Drivers provide support for SCROLL\_SENSITIVE Result Sets. Type-1 Driver will**

**provide support for this feature. But it supports only update operation, but not delete and insert**

**operations.**

**Type-2 driver also can provide support for SCROLL\_SENSITIVE ResultSets. But we should not use \***

**in select query. we should use only column names. It supports only update operation, but not**

**delete and insert operations.**

**Application-4: Reflecting Database updations by using SCROLL SENSITIVE**

**ResultSet (Type-2 Driver)**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetTypesDemo3T2**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:oci8:@XE","scott","tiger**

**");**

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

**9) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

**10) ResultSet rs=st.executeQuery("select eno,ename,esal,eaddr from employees");**

**11) System.out.println("Records Before Updation");**

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

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

**14) while(rs.next())**

**15) {**

**16) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**17) }**

**150**



**18) System.out.println("Application is in pausing state,please update database..");**

**19) System.in.read();**

**20) rs.beforeFirst();**

**21) System.out.println("Records After Updation");**

**22) while(rs.next())**

**23) {**

**24) rs.refreshRow();**

**25) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**26) }**

**27) con.close();**

**28) }**

**29) }**

**Note: Very few Drivers provide support for SCROLL\_SENSITIVE Result Sets. Type-1 Driver will**

**provide support for this feature. But it supports only update operation, but not delete and insert**

**operations.**

**Type-2 driver also can provide support for SCROLL\_SENSITIVE ResultSets. But we should not use \***

**in select query. we should use only column names. It supports only update operation, but not**

**delete and insert operations.**

**Updatable ResultSets:**

**If we perform any changes to the ResultSet and if those changes are reflecting to the Database,**

**such type of ResultSets are called Updatable ResultSets.**

**By default ResultSet is Read only. But we can specify explicitly as updatable by using the following**

**constant.**

**CONCUR\_UPDATABLE** ➔ **1008**

**For Updatable ResultSets, we have to create Statement object as follows..**

**Statement st**

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

**Delete Record From ResultSet:**

**Sample Code:**

**rs.last();**

**rs.deleteRow();**

**151**



**Application-7: Performing Database updations (DELETE operation) by using**

**UPDATABLE ResultSet (Type-1 Driver)**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetTypesDemo5**

**4) {**

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

**6) {**

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

**8) Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");**

**9) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

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

**11) rs.last();**

**12) rs.deleteRow();**

**13) con.close();**

**14) }**

**15) }**

**Note: Very few Drivers provide support for CONCUR\_UPDATABLE Result Sets. Type-1 Driver will**

**provide support for this feature.**

**Type-2 driver also can provide support for CONCUR\_UPDATABLE ResultSets. But we should not use**

**\* In select query. we should use only column names.**

**Update Record of ResultSet:**

**Sample Code Eg 2:**

**1) rs.absolute(3);**

**2) rs.updateString(2,"KTR");**

**3) rs.updateFloat(3,10000);**

4) **rs.updateRow();**

**Sample Code Eg 2:**

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

**2) {**

**3) float esal = rs.getFloat(3);**

**4) if(esal<5000)**

**5) {**

**6) float incr\_sal=esal+777;**

**7) rs.updateFloat(3,incr\_sal);**

**8) rs.updateRow();**

**9) }**

10) **}**

**152**



**Application-5: Performing Database updations (UPDATE operation) by using**

**UPDATABLE ResultSet (Type-1 Driver)**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetTypesDemo4**

**4) {**

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

**6) {**

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

**8) Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");**

**9) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

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

**11) while(rs.next())**

**12) {**

**13) float esal = rs.getFloat(3);**

**14) if(esal<5000)**

**15) {**

**16) float incr\_sal=esal+777;**

**17) rs.updateFloat(3,incr\_sal);**

**18) rs.updateRow();**

**19) }**

**20) }**

**21) con.close();**

**22) }**

**23) }**

**Note: Very few Drivers provide support for CONCUR\_UPDATABLE Result Sets. Type-1 Driver will**

**provide support for this feature.**

**Type-2 driver also can provide support for CONCUR\_UPDATABLE ResultSets. But we should not use**

**\* In select query. we should use only column names.**

**Application-6: Performing Database updations (UPDATE operation) by using**

**UPDATABLE ResultSet (Type-2 Driver)**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetTypesDemo4T2**

**4) {**

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

**6) {**

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

**8) Connection con = DriverManager.getConnection("jdbc:oracle:oci8:@XE","scott","tiger");**

**9) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

**10) ResultSet rs=st.executeQuery("select eno,ename,esal,eaddr from employees");**

**153**



**11) while(rs.next())**

**12) {**

**13) float esal = rs.getFloat(3);**

**14) if(esal<5000)**

**15) {**

**16) float incr\_sal=esal+777;**

**17) rs.updateFloat(3,incr\_sal);**

**18) rs.updateRow();**

**19) }**

**20) }**

**21) con.close();**

**22) }**

**23) }**

**Note: Very few Drivers provide support for CONCUR\_UPDATABLE Result Sets. Type-1 Driver will**

**provide support for this feature.**

**Type-2 driver also can provide support for CONCUR\_UPDATABLE ResultSets. But we should not use**

**\* In select query. we should use only column names.**

**Insert operation:**

**Sample code:**

**1) rs.moveToInsertRow();**

**2) rs.updateInt(1,1010);**

**3) rs.updateString(2,"sunny");**

**4) rs.updateFloat(3,3000);**

**5) rs.updateString(4,"Mumbai");**

**6) rs.insertRow();**

**Application-8: Performing Database updations (INSERT operation) by using**

**UPDATABLE ResultSet (Type-1 Driver)**

**1) import java.sql.\*;**

**2) class ResultSetTypesDemo6**

**3) {**

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

**5) {**

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

**7) Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");**

**8) Statement st =con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CON**

**CUR\_UPDATABLE);**

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

**10) rs.moveToInsertRow();//creates an empty record**

**11) rs.updateInt(1,900);**

**12) rs.updateString(2,"katrina");**

**13) rs.updateFloat(3,3000);**

**154**



**14) rs.updateString(4,"Hyd");**

**15) rs.insertRow();**

**16) con.close();**

**17) }**

**18) }**

**Note: Very few Drivers provide support for CONCUR\_UPDATABLE Result Sets. Type-1 Driver will**

**provide support for this feature.**

**Type-2 driver also can provide support for CONCUR\_UPDATABLE ResultSets. But we should not use**

**\* In select query. we should use only column names.**

**Conclusions:**

**1.Updatable ResultSets allows the programmer to perform following operations on ResultSet.**

**select**

**insert**

**delete**

**update**

**2. Updatable ResultSets allows the programmer to perform insert, update and delete database**

**operations without using SQL Queries.**

**3.Very few drivers provide support for Updatable ResultSets.**

**Type-1 Driver provides support**

**Type-2 Driver provides support but we should not use \* in SQL Query and we should use column**

**names.**

**4.ResultSet cannot be updatable if we are using joins and aggregate functions**

**5.It is not recommended to perform database updations by using updatable ResultSets, b'z most**

**of the drivers and most of the databases won't provide support for Updatable ResultSets.**

**ResultSet Holdability:**

**The ResultSet holdability represents whether the ResultSet is closed or not whenever we call**

**commit() method on the Connection object.**

**There are two types of Holdability**

**HOLD\_CURSORS\_OVER\_COMMIT** ➔ **1**

**CLOSE\_CURSORS\_AT\_COMMIT** ➔ **2**

**HOLD\_CURSORS\_OVER\_COMMIT:**

**It means the ResultSet will be opened for further operations even after calling con.commit()**

**method.**

**155**



**CLOSE\_CURSORS\_AT\_COMMIT:**

**It means that ResultSet will be closed automatically whenever we are calling con.commit()**

**method.**

**We can get Current Holdability of the ResultSet as follows.**

**SOP(rs.getHoldability());**

**For most of the databases default holdabiltiy is 1**

**We can check whether database provides support for a particular holdability or not by using the**

**following method of DatabaseMetaData.**

**supportsResultSetHoldability()**

**We can create Statement object for our required Holdability as follows...**

**Statement st = con.createStatement(1005,1008,2);**

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

**con.commit();**

**rs.aboslute(3);** ➔ **SQLException**

**Note: Most of the databases like Oracle, MySQL won't provide support for holdability 2.**

**Program to check ResultSet Holdability:**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetHoldabilityDemo1**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**8) DatabaseMetaData dbmd=con.getMetaData();**

**9) if(dbmd.supportsResultSetHoldability(1))**

**10) {**

**11) System.out.println("HOLD\_CURSORS\_OVER\_COMMIT");**

**12) }**

**13) if(dbmd.supportsResultSetHoldability(2))**

**14) {**

**15) System.out.println("CLOSE\_CURSORS\_AT\_COMMIT");**

**16) }**

**17) }**

**18) }**

**156**



**Program to display properties of ResultSet:**

**1) import java.sql.\*;**

**2) import java.util.\*;**

**3) class ResultSetHoldabilityDemo3**

**4) {**

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

**6) {**

**7) Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE**

**","scott","tiger");**

**8) Statement st =con.createStatement();**

**9) System.out.println("Type :"+st.getResultSetType());**

**10) System.out.println("Concurrency :"+st.getResultSetConcurrency());**

**11) System.out.println("Holdability:"+st.getResultSetHoldability());**

**12) }**

**13) }**

**Summary of ResultSet Types**

**ResultSet Type ResultSet Concurrency ResultSet Holadability**

**TYPE\_FORWARD\_ONLY [1003]**

**TYPE\_SROLL\_INSENSITIVE [1004]**

**TYPE\_ SROLL\_SENSITIVE [1005]**

**The Default Concurrency is**

**TYPE\_FORWARD\_ONLY**

**CONCUR\_READ\_ONLY**

**[1007]**

**CONCUR\_UPDATABLE**

**[1008]**

**The Default Concurrency is**

**CONCUR\_READ\_ONLY**

**HOLD\_CURSORS\_OVER\_**

**COMMIT [1]**

**CLOSE\_CURSORS\_AT\_**

**COMMIT [2]**

**The Default Holadability is**

**HOLD\_CURSORS\_OVER\_**

**COMMIT**

**157**



**RowSets**

**It is alternative to ResultSet.**

**We can use RowSet to handle a group of records in more effective way than ResultSet.**

**RowSet interface present in javax.sql package**

**RowSet is child interface of ResultSet.**

**RowSet implementations will be provided by Java vendor and database vendor.**

**By default RowSet is scrollable and updatable.**

**By default RowSet is serializable and hence we can send RowSet object across the network. But**

**ResultSet object is not serializable.**

**ResultSet is connected i.e to use ResultSet compulsary database Connection must be required.**

**RowSet is disconnected. ie to use RowSet database connection is not required.**

**Types of RowSets:**

**There are two types of RowSets**

**1.Connected RowSets**

**2. Disconnected RowSets**

**Connected RowSets:**

**Connected RowSets are just like ResultSets.**

**To access RowSet data compulsary connection should be available to database.**

**We cannot serialize Connected RowSets**

**Eg: JdbcRowSet**

**Disconnected RowSets:**

**Without having Connection to the database we can access RowSet data.**

**We can serialize Disconnected RowSets.**

**Eg:**

**CachedRowSet**

**WebRowSet**

**158**



**FilteredRowSet**

**JoinRowSet**

**How to create RowSet objects:**

**We can create different types of RowSet objects as follows**

**RowSetFactory rsf = RowSetProvider.newFactory();**

**JdbcRowSet jrs = rsf.createJdbcRowSet();**

**CachedRowSet crs = rsf.createCachedRowSet();**

**WebRowSet wrs = rsf.createWebRowSet();**

**JoinRowSet jnrs = rsf.createJoinRowSet();**

**FilteredRowSet frs = rsf.createFilteredRowSet();**

**Application-1: To create Different RowSet Objects:**

**1) import javax.sql.rowset.\*;**

**2) public class Test**

**3) {**

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

**5) {**

**6) RowSetFactory rsf=RowSetProvider.newFactory();**

**7) JdbcRowSet jrs=rsf.createJdbcRowSet();**

**8) CachedRowSet crs=rsf.createCachedRowSet();**

**9) WebRowSet wrs=rsf.createWebRowSet();**

**10) JoinRowSet jnrs=rsf.createJoinRowSet();**

**11) FilteredRowSet frs=rsf.createFilteredRowSet();**

**12)**

**Connected Disconnected**

**ResultSet (I)**

**RowSet (I)**

**JdbcRowSet (I) CachedRowSet (I)**

**WebRowSet (I)**

**JoinRowSet (I) FilteredRowSet (I)**

**159**



**13) System.out.println(jrs.getClass().getName());**

**14) System.out.println(crs.getClass().getName());**

**15) System.out.println(wrs.getClass().getName());**

**16) System.out.println(jnrs.getClass().getName());**

**17) System.out.println(frs.getClass().getName());**

**18) }**

**19) }**

**1.JdbcRowSet(I):**

**It is exactly same as ResultSet except that it is scrollable and updatable.**

**JdbcRowSet is connected and hence to access JdbcRowSet compulsary Connection must be**

**required.**

**JdbcRowSet is non serializable and hence we cannot send RowSet object across the network.**

**Application-2: To Retrieve records from JdbcRowSet:**

**1) import javax.sql.rowset.\*;**

**2) public class JdbcRowSetRetrieveDemo {**

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

**4) RowSetFactory rsf=RowSetProvider.newFactory();**

**5) JdbcRowSet rs=rsf.createJdbcRowSet();**

**6) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**7) rs.setUsername("root");**

**8) rs.setPassword("root");**

**9) rs.setCommand("select \* from employees");**

**10) rs.execute();**

**11) System.out.println("Employee Details In Forward Direction");**

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

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

**14) while(rs.next()){**

**15) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**16) }**

**17) System.out.println("Employee Details In Backward Direction");**

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

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

**20) while(rs.previous()){**

**21) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**22) }**

**23) System.out.println("Accessing Randomly...");**

**24) rs.absolute(3);**

**25) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**26) rs.first();**

**27) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**160**



**28) rs.last();**

**29) System.out.println(rs.getRow()+"---**

**>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));**

**30) rs.close();**

**31) }**

**32) }**

**Application-3: To Insert Records by using JdbcRowSet**

**1) import java.util.Scanner;**

**2) import javax.sql.rowset.\*;**

**3) public class JdbcRowSetInsertDemo {**

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

**5) RowSetFactory rsf=RowSetProvider.newFactory();**

**6) JdbcRowSet rs=rsf.createJdbcRowSet();**

**7) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**8) rs.setUsername("root");**

**9) rs.setPassword("root");**

**10) rs.setCommand("select \* from employees");**

**11) rs.execute();**

**12) Scanner s=new Scanner(System.in);**

**13) rs.moveToInsertRow();**

**14) while(true){**

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

**16) int eno=s.nextInt();**

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

**18) String ename=s.next();**

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

**20) float esal=s.nextFloat();**

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

**22) String eaddr=s.next();**

**23)**

**24) rs.updateInt(1, eno);**

**25) rs.updateString(2, ename);**

**26) rs.updateFloat(3, esal);**

**27) rs.updateString(4, eaddr);**

**28) rs.insertRow();**

**29)**

**30) System.out.println("Employee Inserted Successfully");**

**31) System.out.print("Do You Want to insert One more Employee[yes/no]? :");**

**32) String option=s.next();**

**33) if(option.equalsIgnoreCase("No")){**

**34) break;**

**35) }**

**36) }**

**37) rs.close();**

**38) }**

**39) }**

**161**



**Application-4: To Update Records by using JdbcRowSet**

**1) import javax.sql.rowset.\*;**

**2) public class JdbcRowSetUpdateDemo {**

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

**4) RowSetFactory rsf=RowSetProvider.newFactory();**

**5) JdbcRowSet rs=rsf.createJdbcRowSet();**

**6) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**7) rs.setUsername("root");**

**8) rs.setPassword("root");**

**9) rs.setCommand("select \* from employees");**

**10) rs.execute();**

**11) while(rs.next()){**

**12) float esal=rs.getFloat(3);**

**13) if(esal<10000){**

**14) float new\_Esal=esal+500;**

**15) rs.updateFloat(3, new\_Esal);**

**16) rs.updateRow();**

**17) }**

**18) }**

**19) System.out.println("Records Updated Successfully");**

**20) rs.close();**

**21) }**

**22) }**

**Application-5: To Delete Records by using JdbcRowSet**

**1) import javax.sql.rowset.\*;**

**2) public class JdbcRowSetDeleteDemo {**

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

**4) RowSetFactory rsf=RowSetProvider.newFactory();**

**5) JdbcRowSet rs=rsf.createJdbcRowSet();**

**6) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**7) rs.setUsername("root");**

**8) rs.setPassword("root");**

**9) rs.setCommand("select \* from employees");**

**10) rs.execute();**

**11) while(rs.next()){**

**12) float esal=rs.getFloat(3);**

**13) if(esal>5000){**

**14) rs.deleteRow();**

**15) }**

**16) }**

**17) System.out.println("Records Deleted Successfully");**

**18) rs.close();**

**19) }**

**20) }**

**162**



**CachedRowSet:**

**It is the child interface of RowSet.**

**It is bydefault scrollable and updatable.**

**It is disconnected RowSet. ie we can use RowSet without having database connection.**

**It is Serializable.**

**The main advantage of CachedRowSet is we can send this RowSet object for multiple people**

**across the network and all those people can access RowSet data without having DB Connection.**

**If we perform any update operations(like insert,delete and update) to the CachedRowSet,to**

**reflect those changes compulsary Connection should be established.**

**Once Connection established then only those changes will be reflected in Database.**

**Application-6: To Demonstrate Disconnected CachedRowSet**

**1) import java.sql.\*;**

**2) import javax.sql.rowset.\*;**

**3) public class CachedRowSetDemo {**

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

**5) Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/elanzadb**

**","root","root");**

**6) Statement st =con.createStatement();**

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

**8) RowSetFactory rsf=RowSetProvider.newFactory();**

**9) CachedRowSet crs=rsf.createCachedRowSet();**

**10) crs.populate(rs);**

**11) con.close();**

**12) //Now we cannot access RS but we can access CRS**

**13) //if(rs.next()){}RE:SQLException:Operation not allowed after ResultSet closed**

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

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

**16) while(crs.next()){**

**17) System.out.println(crs.getInt(1)+"\t"+crs.getString(2)+"\t"+crs.getFloat(3)+"\t"+crs.**

**getString(4));**

**18) }**

**19) }**

**20) }**

**Application-7: To Retrieve Records by using CachedRowSet**

**1) import javax.sql.rowset.\*;**

**2) public class CachedRowSetRetrieveDemo {**

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

**4) RowSetFactory rsf=RowSetProvider.newFactory();**

**5) CachedRowSet rs=rsf.createCachedRowSet();**

**6) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**163**



**7) rs.setUsername("root");**

**8) rs.setPassword("root");**

**9) rs.setCommand("select \* from employees");**

**10) rs.execute();**

**11) System.out.println("Data In Forward Direction");**

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

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

**14) while(rs.next()){**

**15) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**16) }System.out.println("Data In Backward Direction");**

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

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

**19) while(rs.previous()){**

**20) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**21) }**

**22) }**

**23) }**

**Application-8: To Insert Records by using CachedRowSet**

**1) import java.util.\*;**

**2) import javax.sql.rowset.\*;**

**3)**

**4) public class CachedRowSetInsertDemo {**

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

**6) RowSetFactory rsf=RowSetProvider.newFactory();**

**7) CachedRowSet rs=rsf.createCachedRowSet();**

**8) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb?relaxAutoCommit=true");**

**9) rs.setUsername("root");**

**10) rs.setPassword("root");**

**11) rs.setCommand("select \* from employees");**

**12) rs.execute();**

**13) Scanner s=new Scanner(System.in);**

**14) while(true){**

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

**16) int eno=s.nextInt();**

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

**18) String ename=s.next();**

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

**20) float esal=s.nextFloat();**

**21) System.out.print("EMployee Address :");**

**22) String saddr=s.next();**

**23)**

**24) rs.moveToInsertRow();**

**25) rs.updateInt(1, eno);**

**26) rs.updateString(2, ename);**

**27) rs.updateFloat(3, esal);**

**164**



**28) rs.updateString(4, saddr);**

**29) rs.insertRow();**

**30)**

**31) System.out.println("Employee Inserted Successfully");**

**32) System.out.print("Do you want to insert One more Employee[Yes/No]? :");**

**33) String option=s.next();**

**34) if(option.equalsIgnoreCase("No")){**

**35) break;**

**36) }**

**37) }**

**38) rs.moveToCurrentRow();**

**39) rs.acceptChanges();**

**40) rs.close();**

**41) }**

**42) }**

**Application-9: To Update Records by using CachedRowSet**

**1) import javax.sql.rowset.\*;**

**2)**

**3) public class CachedRowSetUpdateDemo {**

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

**5)**

**6) RowSetFactory rsf=RowSetProvider.newFactory();**

**7) CachedRowSet rs=rsf.createCachedRowSet();**

**8) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb?relaxAutoCommit=true");**

**9) rs.setUsername("root");**

**10) rs.setPassword("root");**

**11) rs.setCommand("select \* from employees");**

**12) rs.execute();**

**13) while(rs.next()){**

**14) float esal=rs.getFloat(3);**

**15) if(esal<10000){**

**16) esal=esal+500;**

**17) rs.updateFloat(3, esal);**

**18) rs.updateRow();**

**19) }**

**20) }**

**21) rs.moveToCurrentRow();**

**22) rs.acceptChanges();**

**23) System.out.println("Records Updated Successfully");**

**24) rs.close();**

**25) }**

**26) }**

**165**



**Application-10: To Delete Records by using CachedRowSet**

**1) import javax.sql.rowset.\*;**

**2)**

**3) public class CachedRowSetDeleteDemo {**

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

**5)**

**6) RowSetFactory rsf=RowSetProvider.newFactory();**

**7) CachedRowSet rs=rsf.createCachedRowSet();**

**8) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb?relaxAutoCommit=true");**

**9) rs.setUsername("root");**

**10) rs.setPassword("root");**

**11) rs.setCommand("select \* from employees");**

**12) rs.execute();**

**13) while(rs.next()){**

**14) float esal=rs.getFloat(3);**

**15) if(esal>6000){**

**16) rs.deleteRow();**

**17) }**

**18) }**

**19) rs.moveToCurrentRow();**

**20) rs.acceptChanges();**

**21) rs.close();**

**22) System.out.println("Records deleted successfully");**

**23) }**

**24) }**

**WebRowSet(I):**

**It is the child interface of CachedRowSet.**

**It is bydefault scrollable and updatable.**

**It is disconnected and serializable**

**WebRowSet can publish data to xml files,which are very helpful for enterprise applications.**

**FileWriter fw=new FileWriter("emp.xml");**

**rs.writeXml(fw);**

**We can read XML data into RowSet as follows**

**FileReader fr=new FileReader("emp.xml");**

**rs.readXml(fr);**

**Application-11: To Retrieve Records by using WebRowSet**

**1) import java.io.\*;**

**2) import javax.sql.rowset.\*;**

**3)**

**166**



**4) public class WebRowSetRetrieveDemo {**

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

**6) RowSetFactory rsf=RowSetProvider.newFactory();**

**7) WebRowSet rs=rsf.createWebRowSet();**

**8) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**9) rs.setUsername("root");**

**10) rs.setPassword("root");**

**11) rs.setCommand("select \* from employees");**

**12) rs.execute();**

**13) FileWriter fw=new FileWriter("emp.xml");**

**14) rs.writeXml(fw);**

**15) System.out.println("Employee Data transfered to emp.xml file");**

**16) fw.close();**

**17) }**

**18) }**

**Application-12: To Insert Records by using WebRowSet**

**1) import java.io.\*;**

**2) import javax.sql.rowset.\*;**

**3) public class WebRowSetInsertDemo {**

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

**5) RowSetFactory rsf=RowSetProvider.newFactory();**

**6) WebRowSet rs=rsf.createWebRowSet();**

**7) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb?relaxAutoCommit=true");**

**8) rs.setUsername("root");**

**9) rs.setPassword("root");**

**10) rs.setCommand("select \* from employees");**

**11) rs.execute();**

**12) FileReader fr=new FileReader("emp.xml");**

**13) rs.readXml(fr);**

**14) rs.acceptChanges();**

**15) System.out.println("emp data inserted successfully");**

**16) fr.close();**

**17) rs.close();**

**18) }**

**19) }**

**Note: In emp.xml file, <insertRow> tag must be provided under <data> tag**

**Application-13: To Delete Records by using WebRowSet**

**1) import java.io.\*;**

**2) import javax.sql.rowset.\*;**

**3) public class WebRowSetDeleteDemo {**

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

**5) RowSetFactory rsf=RowSetProvider.newFactory();**

**6) WebRowSet rs=rsf.createWebRowSet();**

**167**



**7) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb?relaxAutoCommit=true");**

**8) rs.setUsername("root");**

**9) rs.setPassword("root");**

**10) rs.setCommand("select \* from employees");**

**11) rs.execute();**

**12) FileReader fr=new FileReader("emp.xml");**

**13) rs.readXml(fr);**

**14) rs.acceptChanges();**

**15) System.out.println("emp data deleted successfully");**

**16) fr.close();**

**17) rs.close();**

**18) }**

**19) }**

**Note: In emp.xml file, <deleteRow> tag must be provided under <data> tag**

**JoinRowSet:**

**It is the child interface of WebRowSet.**

**It is by default scrollable and updatable**

**It is disconnected and serializable**

**If we want to join rows from different rowsets into a single rowset based on matched**

**column(common column) then we should go for JoinRowSet.**

**We can add RowSets to the JoinRowSet by using addRowSet() method.**

**addRowSet(RowSet rs,int commonColumnIndex);**

**Application-14: To Retrieve Records by using JoinRowSet**

**1) import java.sql.\*;**

**2) import javax.sql.rowset.\*;**

**3) public class JoinRowSetRetriveDemo {**

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

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

**6) Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/elanzadb**

**","root", "root");**

**7) RowSetFactory rsf=RowSetProvider.newFactory();**

**8)**

**9) CachedRowSet rs1=rsf.createCachedRowSet();**

**10) rs1.setCommand("select \* from student");**

**11) rs1.execute(con);**

**12)**

**13) CachedRowSet rs2=rsf.createCachedRowSet();**

**14) rs2.setCommand("select \* from courses");**

**15) rs2.execute(con);**

**16)**

**17) JoinRowSet rs=rsf.createJoinRowSet();**

**168**



**18) rs.addRowSet(rs1, 4);**

**19) rs.addRowSet(rs2, 1);**

**20) System.out.println("SID\tSNAME\tSADDR\tCID\tCNAME\tCCOST");**

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

**22) while(rs.next()){**

**23) System.out.print(rs.getString(1)+"\t");**

**24) System.out.print(rs.getString(2)+"\t");**

**25) System.out.print(rs.getString(3)+"\t");**

**26) System.out.print(rs.getString(4)+"\t");**

**27) System.out.print(rs.getString(5)+"\t");**

**28) System.out.print(rs.getString(6)+"\n");**

**29) }**

**30) con.close();**

**31) }**

**32) }**

**Note: students and courses tables must require in database with a matched column[Join column]**

**cid.**

**students courses**

**|SID(PK)|SNAME|SADDR|CID| |CID(PK)|CNAME|CCOST|**

**addRowSet(RowSet rowset, int columnIdx)**

**Adds the given RowSet object to this JoinRowSet object and sets the designated column as the**

**match column for the RowSet object.**

**FilteredRowSet(I):**

**It is the child interface of WebRowSet.**

**If we want to filter rows based on some condition then we should go for FilteredRowSet.**

**We can define the filter by implementing Predicate interface.**

**1) public class EmpSalFilter implements Predicate**

**2) {**

**3) evaluate(Object value,String columnName)**

**4) {**

**5) This method will be called at the time of insertion**

**6) }**

**7) evaluate(Object value,int columnIndex)**

**8) {**

**9) this method will be called at the time of insertion**

**10) }**

**11) evaluate(RowSet rs)**

**12) {**

**13) filtering logic**

**14) }**

**15) }**

**169**



**We can set Filter to the FilteredRowSet as follows...**

**EmployeeSalaryFilter f=new EmployeeSalaryFilter(2500,4000);**

**rs.setFilter(f);**

**Application-15: To Retrieve Records by using FilteredRowSet**

**1) import java.sql.\*;**

**2) import javax.sql.\*;**

**3) import javax.sql.rowset.\*;**

**4) class EmployeeSalaryFilter implements Predicate{**

**5) float low;**

**6) float high;**

**7) public EmployeeSalaryFilter(float low,float high) {**

**8) this.low=low;**

**9) this.high=high;**

**10) }**

**11) //this method will be called at the time of row insertion**

**12) public boolean evaluate(Object value, String columnName) throws SQLException {**

**13) return false;**

**14) }**

**15) //this method will be called at the time of row insertion**

**16) public boolean evaluate(Object value, int column) throws SQLException {**

**17) return false;**

**18) }**

**19) public boolean evaluate(RowSet rs) {**

**20) boolean eval=false;**

**21) try{**

**22) FilteredRowSet frs=(FilteredRowSet)rs;**

**23) float esal=frs.getFloat(3);**

**24) if((esal>=low) && (esal<=high)){**

**25) eval=true;**

**26) }else{**

**27) eval=false;**

**28) }**

**29) }catch(Exception e){**

**30) e.printStackTrace();**

**31) }**

**32) return eval;**

**33) }**

**34) }**

**35) public class FilteredRowSetRetriveDemo {**

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

**37) RowSetFactory rsf=RowSetProvider.newFactory();**

**38) FilteredRowSet rs=rsf.createFilteredRowSet();**

**39) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**40) rs.setUsername("root");**

**41) rs.setPassword("root");**

**42) rs.setCommand("select \* from employees");**

**170**



**43) rs.execute();**

**44) System.out.println("Data Before Filtering");**

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

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

**47) while(rs.next())**

**48) {**

**49) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**50) }**

**51) EmployeeSalaryFilter f=new EmployeeSalaryFilter(100,5000);**

**52) rs.setFilter(f);**

**53) rs.beforeFirst();**

**54) System.out.println("Data After Filtering");**

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

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

**57) while(rs.next())**

**58) {**

**59) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**60) }**

**61) rs.close();**

**62) }**

**63) }**

**Event Handling Mechanism for RowSets:**

**We can implement event handling for RowSets.**

**To perform event handling,we have to implement RowSetListener interface.**

**1) class RowSetListenerImpl implements RowSetListener**

**2) {**

**3) rowSetChanged(RowSetEvent e)**

**4) {**

**5) this method will be executed whenever total RowSet content changed**

**6) }**

**7) rowChanged(RowSetEvent e)**

**8) {**

**9) this method will be executed whenever any change performed in rows of RowSet like in**

**sertion,deletion and updation**

**10) }**

**11) cursorMoved(RowSetEvent e)**

**12) {**

**13) this method will be executed whenever cursor moved from one row to another row**

**14) }**

**15) }**

**We can add RowSetListener to the RowSet by using addRowSetListener() method.**

**Eg: rs.addRowSetListener(new RowSetListenerImpl());**

**171**



**Application-16: To Demonstrate Event Handling by using JdbcRowSet**

**1) import javax.sql.\*;**

**2) import javax.sql.rowset.\*;**

**3) class RowSetListenerImpl implements RowSetListener{**

**4)**

**5) public void rowSetChanged(RowSetEvent event) {**

**6) System.out.println("RowSetChanged");**

**7) }**

**8)**

**9) public void rowChanged(RowSetEvent event) {**

**10) System.out.println("RowChanged");**

**11) }**

**12)**

**13) public void cursorMoved(RowSetEvent event) {**

**14) System.out.println("CursorMoved");**

**15) }**

**16) }**

**17) public class RowSetListenerDemo {**

**18)**

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

**20) RowSetFactory rsf=RowSetProvider.newFactory();**

**21) JdbcRowSet rs=rsf.createJdbcRowSet();**

**22) rs.setUrl("jdbc:mysql://localhost:3306/elanzadb");**

**23) rs.setUsername("root");**

**24) rs.setPassword("root");**

**25) rs.setCommand("select \* from employees");**

**26) rs.addRowSetListener(new RowSetListenerImpl());**

**27) rs.execute();**

**28) while(rs.next()){**

**29) System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS**

**tring(4));**

**30) }**

**31) rs.moveToInsertRow();**

**32) rs.updateInt(1, 777);**

**33) rs.updateString(2, "malli");**

**34) rs.updateFloat(3,9000);**

**35) rs.updateString(4, "Hyd");**

**36) rs.insertRow();**

**37) rs.close();**

**38) }**

**39) }**

**172**



**Methods of RowSetListener**

**void cursorMoved(RowSetEvent event)**

**Notifies registered listeners that a RowSet object's cursor has moved.**

**void rowChanged(RowSetEvent event)**

**Notifies registered listeners that a RowSet object has had a change in one of its rows.**

**void rowSetChanged(RowSetEvent event)**

**Notifies registered listeners that a RowSet object in the given RowSetEvent object has changed its**

**entire contents.**

**Differences Between ResultSet and RowSet**

**ResultSet RowSet**

**1) ResultSet present in java.sql Package. 1) RowSet present in javax.sql Package.**

**2) By Default ResultSet is Non Scrollable and**

**Non Updatable (Forward only and Read only).**

**2) By Default RowSet is Scrollable and**

**Updatable.**

**3) ResultSet Objects are Non Serializable and**

**we can't send over Network.**

**3) RowSet Objects are Serializable and hence**

**we can send over Network.**

**4) ResultSet Objects are Connection oriented**

**i.e. we can access ResultSet Data as long as**

**Connection is available once Connection closed**

**we can't access ResultSet Data.**

**4) RowSet Objects are Connection Less**

**Objects i.e. we can access RowSet Data**

**without having Connection to DB (except**

**JdbcRowSet).**

**5) ResultSet Object is used to store Records**

**returned by Select Query.**

**5) RowSet Object is also used to store Records**

**returned by Select Query.**

**6) We can createResultSet Object as follows**

**Connection con = DriverManager.getConnection**

**(url, uname, pwd);**

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

**ResultSet rs = st.executeQuery(SQLQuery);**

**6) RowSetFactory rsf =**

**RowSetProvider.newFactory();**

**JdbcRowSet rs = rsf.createJdbcRowSet();**

**rs.setUserName(user);**

**rs.setUrl(jdbcurl);**

**rs.setPassword(pwd);**

**rs.setCommand(query);**

**rs.execute();**

**7) ResultSet Object is not having Event**

**Notification Model.**

**7) RowSet Object is having Event**

**Notification Model.**

**173**



**Top Most**

**Important**

**JDBC FAQ’s**

**174**



**Q1. What is Driver and how many types of drivers are there**

**in JDBC?**

**The Main Purpose of JDBC Driver is to convert Java (JDBC) calls into Database specific calls and**

**Database specific calls into Java calls. i.e. It acts as a Translator.**

**There are 4 Types of JDBC Drivers are available**

**1. Type-1 Driver (JDBC-ODBC Bridge Driver OR Bridge Driver)**

**2. Type-2 Driver (Native API-Partly Java Driver OR Native Driver)**

**3. Type-3 Driver (All Java Net Protocol Driver OR Network Protocol Driver OR Middleware Driver)**

**4. Type-4 Driver (All Java Native Protocol Driver OR Pure Java Driver OR Thin Driver)**

**Q2. Explain Differences between**

**executeQuery(), executeUpdate() and execute() methods?**

**We can use execute Methods to execute SQL Queries.**

**There are 4 execute Methods in JDBC.**

**1. executeQuery():**

**can be used for Select Queries**

**2. executeUpdate():**

**Can be used for Non-Select Queries (Insert|Delete|Update)**

**3. execute()**

**Can be used for both Select and Non-Select Queries**

**It can also be used to call Stored Procedures.**

**4. executeBatch()**

**Can be used to execute Batch Updates**

**executeQuery() vs executeUpdate() vs execute():**

**1. If we know the Type of Query at the beginning and it is always Select Query then we should use**

**executeQuery() Method.**

**2. If we know the Type of Query at the beginning and it is always Non-Select Query then we should**

**use executeUpdate() Method.**

**175**



**3. If we don't know the Type of SQL Query at the beginning and it is available dynamically at**

**Runtime (may be from Properties File OR from Command Prompt etc) then we should go for**

**execute() Method.**

**Q3. What is Statement and How many Types of Statements**

**are available?**

**To send SQL Query to the Database and to bring Results from Database some Vehicle must be**

**required. This Vehicle is nothing but Statement Object.**

**Hence, by using Statement Object we can send our SQL Query to the Database and we can get**

**Results from Database.**

**There are 3 Types of Statements**

**1. Statement:**

**If we want to execute multiple Queries then we can use Statement Object.**

**Every time Query will be compiled and executed. Hence relatively performance is low.**

**2. PreparedStatement:**

**If we want to execute same Query multiple times then we should go for PreparedStatement.**

**Here Query will be compiled only once even though we executed multiple times. Hence relatively**

**performance is high.**

**PreparedStatement is always associated with precompiled SQL Queries.**

**3. CallableStatement:**

**We can use CallableStatement to call Stored Procedures and Functions from the Database.**

**Statement (I)**

**PreparedStatement (I)**

**CallableStatement (I)**

**176**



**Q4. Explain differences between Statement and PreparedStatement?**

**Differences Between Statement And PreparedStatement**

**Statement PreparedStatement**

**1) At the time of creating Statement Object,**

**we are not required to provide any Query.**

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

**Hence Statement Object is not associated with**

**any Query and we can use for multiple Queries.**

**1) At the time of creating PreparedStatement,**

**we have to provide SQL Query compulsory and**

**will send to the Database and will be compiled.**

**PS pst = con.prepareStatement(query);**

**Hence PS is associated with only one Query.**

**2) Whenever we are using execute Method,**

**every time Query will be compiled and**

**executed.**

**2) Whenever we are using execute Method,**

**Query won't be compiled just will be executed.**

**3) Statement Object can work only for Static**

**Queries.**

**3) PS Object can work for both Static and**

**Dynamic Queries.**

**4) Relatively Performance is Low. 4) Relatively Performance is High.**

**5) Best choice if we want to work with**

**multiple Queries.**

**5) Best choice if we want to work with only**

**one Query but required to execute multiple**

**times.**

**6) There may be a chance of SQL Injection**

**Attack.**

**6) There is no chance of SQL Injection**

**Attack.**

**7) Inserting Date and Large Objects (CLOB and**

**BLOB) is difficult.**

**7) Inserting Date and Large Objects (CLOB and**

**BLOB) is easy.**

**Q5. Explain Steps to develop JDBC Application?**

**1. Load and Register Driver**

**2. Establish Connection b/w Java Application and Database**

**3. Create Statement Object**

**4. Send and Execute SQL Query**

**5. Process Results from ResultSet**

**6. Close Connection**

**Q6. Explain main Important components of JDBC?**

**The Main Important Components of JDBC are:**

**1. Driver**

**2. DriverManager**

**3. Connection**

**4. Statement**

**5. ResultSet**

**177**



**1.Driver(Translator):**

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

**2. DriverManager:**

**DriverManager is a Java class present in *java.sql* Package.**

**It is responsible to manage all Database Drivers available in our System.**

**DriverManager is responsible to register and unregister Database Drivers.**

**DriverManager.registerDriver(driver);**

**DriverManager.unregisterDriver(driver);**

**DriverManager is responsible to establish Connection to the Database with the help of Driver**

**Software.**

**Connection con=DriverManager.getConnection(jdbcurl,username,pwd);**

**3. Connection (Road):**

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

**4. Statement (Vehicle):**

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

**from Database.**

**To send SQL Query to the Database and to bring Results from Database some Vehicle must be**

**required. This Vehicle is nothing but Statement Object.**

**Hence, by using Statement Object we can send our SQL Query to the Database and we can get**

**Results from Database.**

**There are 3 types of Statements**

**1.Statement**

**2.PreparedStatement**

**3.CallableStatement**

**5. ResultSet:**

**Whenever we are executing Select Query, Database engine will provide Result in the form of**

**ResultSet. Hence ResultSet holds Results of SQL Query. By using ResultSet we can access the**

**Results.**

**178**



**Q7. Explain JDBC Architecture?**

**JDBC Architecture**

 **JDBC API provides DriverManager to our Java Application.**

 **Java Application can communicate with any Database with the help of DriverManager and**

**Database specific Driver.**

**DriverManager:**

 **It is the Key Component in JDBC Architecture.**

 **DriverManager is a Java Class present in java.sql Package.**

 **It is responsible to manage all Database Drivers available in our System.**

 **DriverManager is responsible to register and unregister Database Drivers.**

**DriverManager.registerDriver(Driver);**

**DriverManager.unregisterDriver(Driver);**

**Driver For**

**Oracle**

**Driver For**

**MySql**

**Driver For**

**Sybase**

**Java Application**

**DriverManager**

**JDBC API**

**Oracle**

**Database**

**MySql**

**Database**

**Sybase**

**Database**

**179**



 **DriverManager is responsible to establish Connection to the Database with the help of Driver**

**Software.**

**Connection con = DriverManager.getConnection (jdbcurl, username, pwd);**

**Database Driver:**

 **It is the very Important Component of JDBC Architecture.**

 **Without Driver Software we cannot Touch Database.**

 **It acts as Bridge between Java Application and Database.**

 **It is responsible to convert Java calls into Database specific calls and Database specific calls**

**into Java Calls.**

**Q8. Explain about BLOB and CLOB?**

**Sometimes as the Part of programming Requirement, we have to Insert and Retrieve Large Files**

**like Images, Video Files, Audio Files, Resume etc wrt Database.**

**Eg:**

**Upload Image in Matrimonial Web Sites**

**Upload Resume in Job related Web Sites**

**To Store and Retrieve Large Information we should go for Large Objects (LOBs).**

**There are 2 Types of Large Objects.**

**1. Binary Large Object (BLOB)**

**2. Character Large Object (CLOB)**

**1. Binary Large Object (BLOB):**

**A BLOB is a Collection of Binary Data stored as a Single Entity in the Database.**

**BLOB Type Objects can be Images, Video Files, Audio Files etc..**

**BLOB Data Type can store Maximum of "4GB" Binary Data.**

**2. Character Large Objects (CLOB):**

**A CLOB is a Collection of Character Data stored as a Single Entity in the Database.**

**CLOB can be used to Store Large Text Documents (May Plain Text OR XML Documents)**

**CLOB Type can store Maximum of 4 GB Data.**

**Eg: hydhistory.txt**

**180**



**Q9. Explain about Batch Updates?**

**When we Submit Multiple SQL Queries to the Database one by one then lot of time will be wasted**

**in Request and Response.**

**For Example our Requirement is to execute 1000 Queries. If we are trying to submit 1000 Queries**

**to the Database one by one then we need to communicate with the Database 1000 times. It**

**increases Network Traffic between Java Application and Database and even creates Performance**

**Problems also.**

**To overcome these Problems, we should go for Batch Updates. We can Group all related SQL**

**Queries into a Single Batch and we can send that Batch at a time to the Database.**

**Sample Code:**

**st.addBatch(sqlQuery-1);**

**st.addBatch(sqlQuery-2);**

**st.addBatch(sqlQuery-3);**

**st.addBatch(sqlQuery-4);**

**st.addBatch(sqlQuery-5);**

**st.addBatch(sqlQuery-6);**

**...**

**st.addBatch(sqlQuery-1000);**

**st.executeBatch();**

**181**



**JDBC**

**Interview**

**FAQ’s**

**182**



**1) What is JDBC?**

**2) What is Latest version of JDBC available?**

**3) Explain about JDBC Architecture?**

**4) Explain about common JDBC Components?**

**5) Explain about DriverManager?**

**6) What is JDBC API?**

**8) Who has provided JDBC API?**

**9) What are the classes and interfaces available in JDBC API?**

**10) Who has provided implementation of JDBC API?**

**11) What are the steps to write JDBC Program?**

**12) What is JDBC Driver?**

**13) How many types of JDBS Drivers available?**

**14) Explain TYPE I Driver?**

**15) Which version of Java has excluded TYPE I Driver?**

**16) I have loaded both Oracle and MySQL drivers, Which database connection will be**

**established when we call getConnection(...)method?**

**Ans: Based on jdbc url the Connection object will be created to the database.**

**17) I have loaded Oracle driver and trying to get the connection with MySQL URL What**

**will happen?**

**Code:**

**Class.for Name("oracle.jdbc.OracleDriver");**

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

**Ans: We will get ClassNotFoundException**

**183**



**18) What the DriverManager.getConnection() method doing?**

**In JDBC API or in java.sql package, SUN has given more interfaces like Connection,**

**Statement, ResultSet, Etc., How Instances will be created?**

**19) Can I register the Driver Explicitly?**

**20) Can I unregister the Driver?**

**21) How can i find list of drivers registered?**

**22) How many types of JDBC Drivers are available? Which is best?**

**23) Explain the cases when each driver should be used?**

**24) Which Type of JDBC Driver is the Fastest One?**

**25) Explain two important approaches to Register a Driver?**

**26) Whenever we are using Class.forName() method to load Driver class automatically**

**Driver will be Registered with DriverManager. Then what is the need of DriverManager**

**class registerDriver() method.**

**Ans: This method is useful whenever we are using Non-JDK Complaint Driver.**

**27) Can I establish two database connections at a time?**

**28) What are the difference among 3 getConnections() method?**

**1) public static Connection getConnection(String url)**

**2) public static Connection getConnection(String url,String uname, String pword)**

**3) public static Connection getConnection(String url,Properties info)**

**29) Can we specify the column name in the select statement or not?**

**30) What is the use of execute() if we have the executeUpdate() or executeQuery()?**

**31) What is Statement?**

**32) How many types of JDBC Statements are available?**

**33) In which package the statement is defined?**

**184**



**34) Is there any super type defined for statement?**

**35) Who is responsible to define implementation class for statement?**

**36) How to get /create the object of statement type?**

**37) While creating the statement do we need to provide any SQL statement?**

**38) What are the methods can be used from statement to submit the SQL Query to**

**database.?**

**39) What is the difference among executeUpdate(), executeQuery() and execute()**

**methods?**

**40) How many Queries we can submit by using one statement object?**

**41) How many types of queries I can submit using one statement object?**

**42) When exactly SQL statement will be submitted to the database?**

**43) When you submit the SQL statement to database using statement then how many**

**times the SQL statement will be compiled/verified?**

**44) How to use dynamic value to the SQL statement in the case of statement object?**

**45) What is the PreparedStatement?**

**46) In which package the PreparedStatement is defined?**

**47) Is there any super type defined for PreparedStatement?**

**48) Who is responsible to define implementation class for PreparedStatement?**

**49) How to get/create the object of PreparedStatement type?**

**50) While creating the prepared Statement do we need to provide any SQL Statement?**

**51) What are the methods can be used from Prepared Statement to submit the SQL Query**

**to database?**

**52) How many Queries we can submit using one Prepared Statement object?**

**53) How many types of queries We can submit using one PreparedStatement object?**

**185**



**54) When we submit the SQL statement to database using Prepared Statement then how**

**many times the SQL Statement will be compiled/vefified?**

**55) How to use dynamic value to the SQL statement in the case of PreparedStatement**

**object?**

**56) What is the difference between Statement and PreparedStatement ?**

**57) What is the benefit of PreparedStatement over Statement?**

**58) What is CallableStatement?**

**59) In Which package the CallableStatement is defined?**

**60) Is there any super type defined for CallableStatement?**

**61) Who is responsible to defined implementation class for CallableStatement?**

**62) How to get/create the object of CallableStatement type?**

**63) While creating the Callable Statement do we need to provide any SQL Statement?**

**64) What is the purpose/benefit of CallableStatement?**

**65) What are the methods can be used from CallableStatement to call the procedure from**

**database?**

**66) When we call the procedure from database using CallableStatement then how many**

**times the SQL Statement will be compiled/verified?**

**67) How to use dynamic value to the procedure in the case of CallableStatement object?**

**68) How can we call the procedure from Java application using input parameter?**

**69) How can you call the procedure from Java Application using output parameter of the**

**procedure?**

**70) How to get the value of output parameter of the procedure?**

**71) Can we write different types of SQL statement in procedure?**

**72) Can we submit select statement using batch update?**

**73) How to get the result from the callable statement if you invoke any stored function?**

**186**



**74) How can you access column information from ResultSet?**

**75) Can I access Statement and ResultSet after closing the connection?**

**76) What is the Batch Update? OR What is the advantage of Batch Update?**

**77) How to use Batch Update with Statement?**

**78) How to use Batch Update with Preparedstatement?**

**79) Can I submit insert Statement using Batch Update?**

**80) Can I submit update Statement using Batch Update?**

**81) Can I submit delete Statement using Batch Update?**

**82) Can I submit select Statement using Batch Update?**

**83) Can I submit different types of SQL statement with Batch Update using Statement?**

**84) Can I submit different types of SQL statement with Batch Update using Prepared**

**Statement?**

**85) What is Metadata?**

**86) What is DatabaseMetadata?**

**87) In Which package the DatabaseMetaData is available?**

**88) Who has defined the implementation class for DatabaseMetaData?**

**89) How can we get the object of DatabaseMetaData type?**

**90) What is the use of DatabaseMetaData?**

**91) How can I access the Database Product Name?**

**92) How can I access the Database Product version?**

**93) How can I access the Driver Name?**

**94) How can I access the Driver version?**

**95) How can I check whether Database supports batch update or not?**

**187**



**96) How can I check whether Database supports Full Outer Join or not?**

**97) What is ResultSetMetadata?**

**98) In Which package the ResultSetMetadata is available.?**

**99) Who has defined the implementation class for ResultSetMetadata?**

**100) How to get/create the object of ResultSetMetadata type?**

**101) What is the use of ResultSetMetadata type?**

**102) How can I get the number of columns available in Resultset?**

**103) How can I access the name & order of the columns available in Resultset?**

**104) How can I access the type of the columns available in Resultset?**

**105) What is transaction?**

**106) What is transaction management?**

**107) What is ACID properties?**

**108) What will happen when auto commit is true?**

**109) By using which methods we can implement Transactions in JDBC?**

**110) What are the Transactional concurrency problems?**

**111) Explain about Dirty Read Problem?**

**112) Explain about Repeatable Read Problem?**

**113) Explain about Phantom Read Problem?**

**114) What are the Transactional isolation levels?**

**115) Which isolation levels prevent Dirty Read Problem?**

**116) Which isolation levels prevent Repeatable Read Problem?**

**117) Which isolation levels prevent Phantom Read Problem?**

**118) What will happen when Iam not specifying the isolation Level with JDBC?**

**188**



**119) How can I get Database Vendor Specific Default Transactional Isolation Level?**

**120) What is the Default Transactional Isolation Level My SQL?**

**121) What is the Default Transactional Isolation Level Oracle?**

**122) What are the ways to manage the Connections in JDBC?**

**123) What are the advantages of DataSource Connections over Driver Manager**

**connections ?**

**124) What is ResultSet?**

**125) In Which package , ResultSet is available.?**

**126) Who has defined the implementation class of ResultSet?**

**127) How can we get the Object of ResultSet Type?**

**128) What does the ResultSet represent?**

**129) What are the types of ResultSet available as per Cursor movement?**

**130) What is forward only ResultSet?**

**131) How can you get the Forward Only ResultSet?**

**132) Can I call the following method with Forward Only ResultSet?**

**a. previous() b. first() c. last() d. absolute() e. relative()**

**133) What is Scrollable ResultSet?**

**134) How can I get the Scrollable ResultSet?**

**135) Can I call the following method with Scrollable ResultSet?**

 **previous()**

 **first()**

 **last()**

 **absolute()**

 **relative()**

**136) What are the types of Resultset available as per Operation?**

**189**



**137) What are the Read Only ResultSet?**

**138) How can you get the Read Only Resultset?**

**139) Can I call the following method with Read Only Resultset?**

 **moveTolnsertRow()**

 **updateRow()**

 **deleteRow()**

 **insertRow()**

 **updateX(int col\_lndex, X value)**

**140) What is updatable Resultset?**

**141) How can you get the updatable Resultset?**

**79. Can I call the following method with Updatable Resultset?**

 **moveTolnsertRow()**

 **updateRow()**

 **deleteRow()**

 **insertRow()**

 **updateX(int col\_lndex, X value)**

**142) What is the default type of Resultset?**

**143) What are the constants defined to specify the Resultset type?**

**144) What is the default concurrency of Resultset?**

**145) What are the constants defined to specify the Resultset concurrency?**

**146) What is difference between Scroll SENSITIVE and INSENSITIVE?**

**147) What are various Types of ResultSet based on cursor movement?**

**148) What are various Types of ResultSet based on operations?**

**149) What are various Types of ResultSet based on holdability?**

**150) What is Rowset?**

**151) What is the super type for RowSet?**

**152) How to get the object of RowSet?**

**190**



**153) How many types of RowSet available as per connection?**

**154) How many sub types of RowSet interface available?**

**155) What is the default type of RowSet?**

**156) What is the default concurrency RowSet?**

**157) Can I serialize the Cached RowSet?**

**158) Can I serialize the JDBC RowSet?**

**159) What is the difference between ResultSet and RowSet?**

**160) What is the use of RowSet Factory and RowSet Provider?**

**161) What are the new features of JDBC 4.0?**

**162) What are the new features of JDBC 4.1?**

**163) What is ResultSet holdability?**