UNIT 4

WEB TECHNOLOGY

EJB & JDBC

What is JavaBean?

- JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.
- In simple terms, JavaBeans are classes which encapsulate several objects into a single object.
- It helps in accessing these object from multiple places. JavaBeans contains several elements like Constructors, Getter/Setter Methods and much more.

EJB

EJB is an acronym for *enterprise java bean*. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.

To get information about distributed applications, visit RMI Tutorial first.

To run EJB application, you need an *application server* (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. It performs:

- life cycle management,
- security,
- transaction management, and
- object pooling.

EJB application is deployed on the server, so it is called server side component also.

EJB is like COM (*Component Object Model*) provided by Microsoft. But, it is different from Java Bean, RMI and Web Services.

When use Enterprise Java Bean?

Application needs Remote Access. In other words, it is distributed.
Application needs to be scalable. EJB applications supports load balancing, clustering and fail-over.
Application needs encapsulated business logic. EJB application is separated from presentation and persistent layer

Introduction to EJB

EJB(Enterprises Java Bean)

• It is the server side component of the Java Enterprise Edition specification where the business logic of an application resides.

• They required developers to create multiple interfaces and XML files just to define a single bean.

• The EJB container provides services such as security, transactions, resource management, dependency injection, concurrency and lookup so that the developer is free to focus on business logic.

Introduction to EJB(cont..)

Preparing a Class to be a JavaBeans

- It is a reusable software component.

- A bean encapsulates many objects into one object so that we can access this object from multiple places.

- It should have a no-arg constructor.

- It should be Serializable.

- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Introduction to EJB(cont..)

- **Example of Java Bean Class**

```
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

Introduction to EJB(cont..)

- **To Access the Java Bean Class**

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

Previous Topics: Recap

- The previous topic was mainly focused on using java bean classes as a reusable software components

UNIT 4

Topics: Enterprise Java Bean: Creating a JavaBeans. **(CO4)**

Objective of the above topics:

- To understand the concepts of creating Enterprises java bean classes.

Introduction to EJB(cont..)

- **Creating a Java Bean Class**

  - **Put this source code into a file named "SimpleBean.java"**

```
import java.awt.*;
import java.io.Serializable;
public class SimpleBean extends Canvas
 implements Serializable{
//Constructor sets inherited properties
 public SimpleBean(){
  setSize(60,40);
setBackground(Color.red);
}
}
```

Introduction to EJB(cont..)

- **Creating a Java Bean Class(cont..)**

  - **Compile the file**
    javac SimpleBean.java

  - **Create a manifest file, named "manifest.tmp":**
    Name: SimpleBean.class
    Java-Bean: True

  - **Create the JAR file, named "SimpleBean.jar":**

  - **Start the Bean Box.**
    c:\Program Files\BDK1.1\beanbox\.
    Load JAR into Bean Box by selecting "LoadJar..." under the File menu.

Introduction to EJB(cont..)

- **Creating a Java Bean Class(cont..)**

▪ **After the file selection dialog box is closed, change focus to the "ToolBox" window.**
You'll see "SimpleBean" appear at the bottom of the toolbox window. Select SimpleBean.jar.

▪ **Try changing the red box's color with the Properties windows.**

▪ **Chose "Events" under the "Edit" menu in the middle window to see what events SimpleBean can send. These events are inherited from java.awt.Canvas.**

Introduction to EJB(cont..)

- **Creating a Java Bean Class(cont..)**

```
import java.awt.*;
    import java.io.Serializable;
    public class MediumBean
extends Canvas
                    implements
Serializable {
      private Color color =
Color.green;
      //property getter method
      public Color getColor(){
        return color;
      }
```

```
public void setColor(Color
newColor){
        color = newColor;
        repaint();
      }
```

Introduction to EJB(cont..)

- **Creating a Java Bean Class(cont..)**

```
public void paint(Graphics g) {
        g.setColor(color);
        g.fillRect(20, 5, 20, 30);
       }
       //Constructor sets inherited properties
       public MediumBean(){
        setSize(60,40);
        setBackground(Color.red);  } }
```

Previous Topics: Recap

• It was discussed about creating a java bean class and setter and getter properties.

UNIT 4

Topics: Enterprise Java Bean: JavaBeans Properties, Types of beans. **(CO4)**

Objective of the above topics:

- To understand the various types of beans and their properties.
- To understand about types of session and entity bean with example

Introduction to EJB(cont..)

- **JavaBeans Properties**

  - A JavaBean property is a named feature that can be accessed by the user of the object.

  - A JavaBean property may be read, write, read-only, or write-only.

  - JavaBean features are accessed through two methods in the JavaBean's implementation class

    - **getPropertyName ()**

    - **setPropertyName ()**

Introduction to EJB(cont..)

- **JavaBeans Properties(cont..)**
  - **getPropertyName ()**
    - For example, if the property name is firstName, the method name would be getFirstName() to read that property.
    - This method is called the accessor.
  - **setPropertyName ()**
    - For example, if the property name is firstName, the method name would be setFirstName() to write that property.
    - This method is called the mutator.

Introduction to EJB(cont..)

- **Types of Beans**
  - **Session Bean**
  - **Entity Bean**

Introduction to EJB(cont..)

- **Session Bean**

  - A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views.

  - To access an application that is deployed on the server, the client invokes the session bean's methods.

  - The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

Introduction to EJB(cont..)

- **Types Session Bean**
  - **Stateful Session Beans**
    - The state of an object consists of the values of its instance variables.
    - In a stateful session bean, the instance variables represent the state of a unique client/bean session.
    - The client interacts (“talks”) with its bean, this state is often called the conversational state.

Introduction to EJB(cont..)

- **Types Session Bean(cont..)**
  - **Stateless Session Beans**
    - A stateless session bean does not maintain a conversational state with the client.
    - When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation.

Introduction to EJB(cont..)

Example:-{Stateless Session Beans}

```
import javax.ejb.Stateless.*;

@Stateless(name="CalculateEJB")
public class CalculateEJBBean implements
CalculateEJB
{
int value = 0;
public String incrementValue()
{
value++;
return "value incremented by 1";
}
}
```

Introduction to EJB(cont..)

Example:-{Stateful Session beans}

```
import javax.ejb.Stateful;

@Stateful
public class MyStatefulBean
{
  public String echo(String name)
  {
    return "Hello " + name;
  }
}
```

Introduction to EJB(cont..)

- **Entity Bean**

  - An entity bean is an object that manages persistent data, potentially uses several dependent Java objects, and can be uniquely identified by a primary key.

  - Entity beans represent persistent data from the database, such as a row in a customer table, or an employee record in an employee table.

  - Entity beans are sharable across multiple clients.

  - Specific fields of the entity bean object can be made persistent.

Introduction to EJB(cont..)

- **Entity Bean Example**

```
import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
@Entity
@Table(name = "EMPLOYEES")
public class Employee implements java.io.Serializable
{
private int empId;
private String eName;
private double sal;

@Id
@Column(name="EMPNO", primaryKey=true)
```

Introduction to EJB(cont..)

- **Entity Bean Example(cont..)**

```
public int getEmpId()
{
return empId;
}
public void setEmpId(int empId)
{
this.empId = empId;
}
public String getEname()
{
return eName;
}
```

Introduction to EJB(cont..)

- **Entity Bean Example(cont..)**

```
public void setEname(String
eName)
{
this.eName = eName;
}
public double getSal()
{
return sal;
}
public void setSal(double sal)
{
this.sal = sal;
}
```

```
public String toString()
{
StringBuffer buf = new
StringBuffer();
buf.append("Class:")
.append(this.getClass().getNa
me()).append(" :: ").append("
empId:").append(getEmpId())
.append("
ename:").append(getEname())
.append("sal:").append(getSal
());
return buf.toString();  }  }
```

Previous Topics: Recap

- It was focused on various types beans such as: session and entity bean.

- It was also focused stateless and stateful session bean with example

UNIT 4

Topics: Database Connectivity (JDBC): Merging Data from Multiple Tables: Joining, Manipulating ,Databases with JDBC. **(CO4)**

Objective of the above topics:

- To understand the concepts of Java Database Connectivity(JDBC)
- To understand the concepts of using various types of SQL query in JDBC

Introduction to JDBC

- **JDBC**

  •JDBC stands for Java Database Connectivity.

  •JDBC is a Java API to connect and execute the query with the database.

  •JDBC API uses JDBC drivers to connect with the database.

Introduction to JDBC(cont..)

- **JDBC Drivers**
  - JDBC-ODBC Bridge Driver
  - Native Driver
  - Network Protocol Driver
  - Thin Driver or Oracle Driver

JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

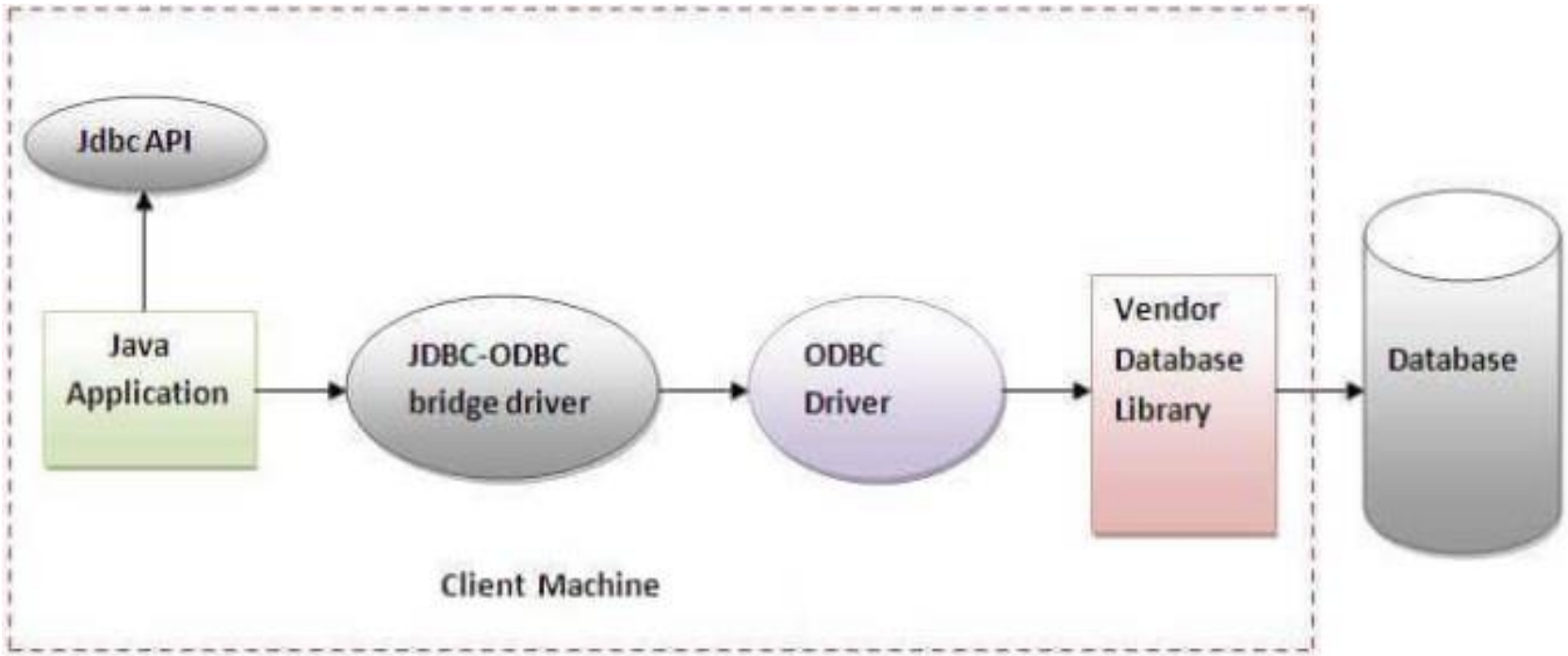


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

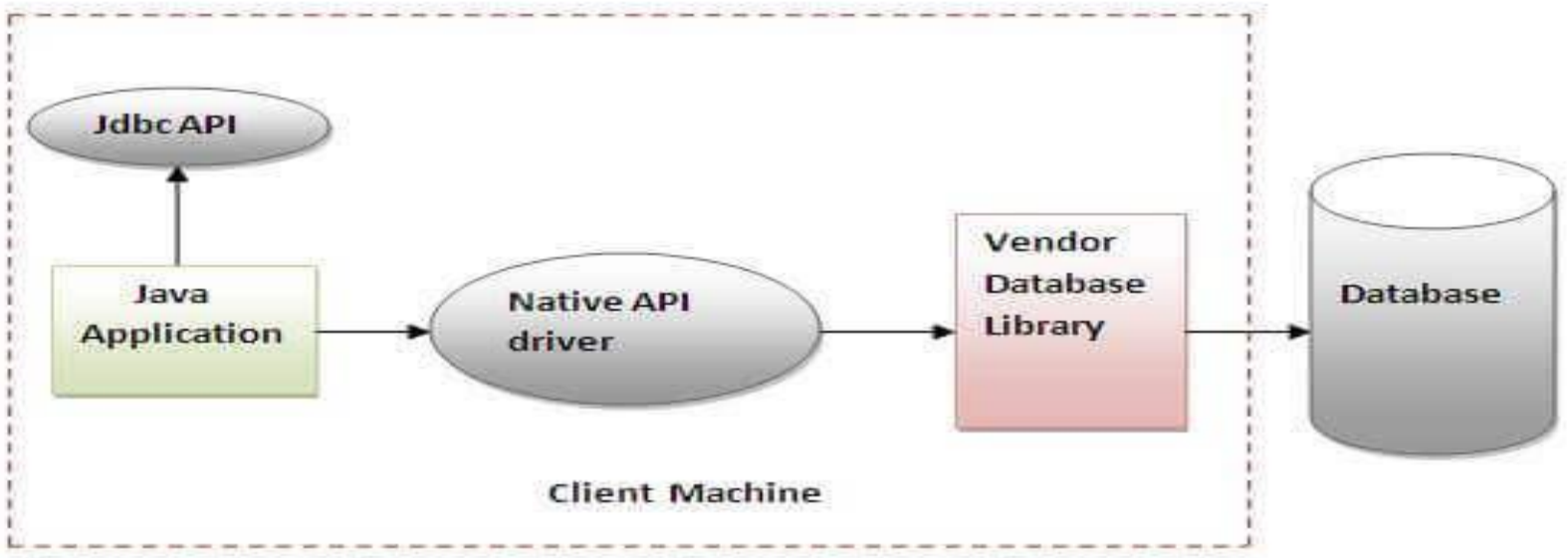


Figure- Native API Driver

Advantage:

performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

The Native driver needs to be installed on the each client machine.
The Vendor client library needs to be installed on client machine.

Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

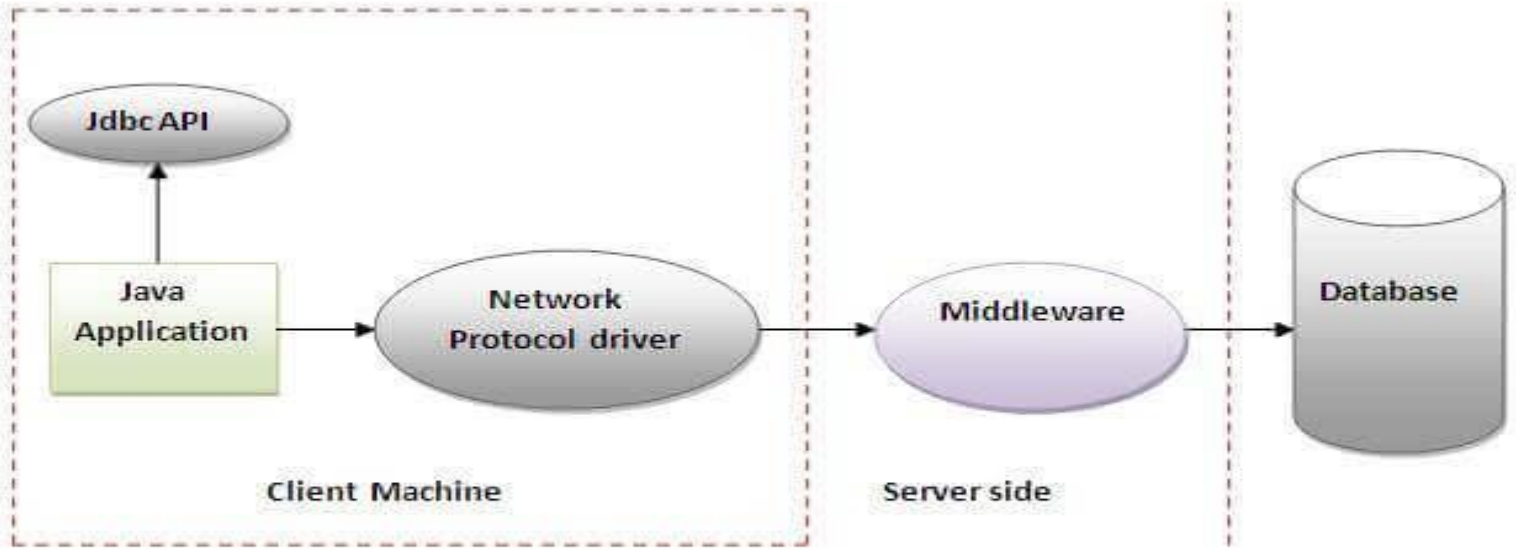


Figure- Network Protocol Driver

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

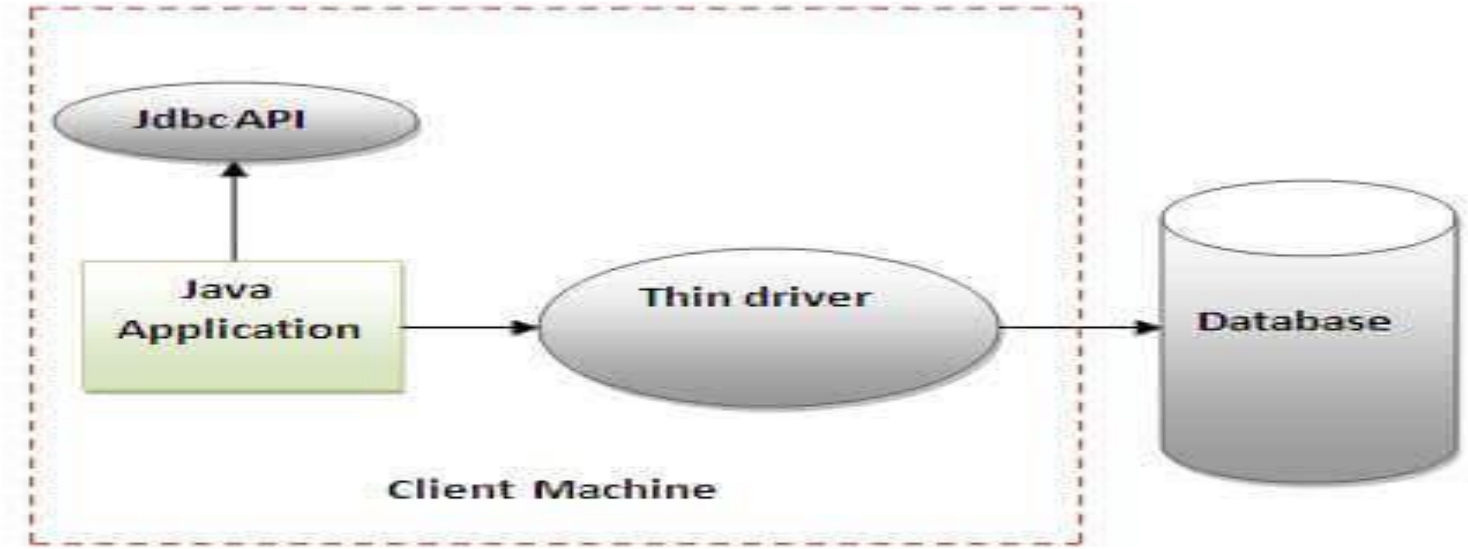


Figure- Thin Driver

Advantage:
Better performance than all other drivers.
No software is required at client side or server side.

Disadvantage:
Drivers depend on the Database.

Introduction to JDBC(cont..)

- **Steps for Java Database Connectivity**

  - Import JDBC packages.
  - Load and register the JDBC driver.
  - Open a connection to the database.
  - Create a statement object to perform a query.
  - Execute the statement object and return a query resultset.
  - Process the resultset.

Introduction to JDBC(cont..)

- **Merging Data from Multiple Tables**

  - Whenever we need to select data from two or more tables, we have to perform a join operation.

  - Tables in a database can be related to each other with keys.

  - A primary key is a column with a unique value for each row.

  - Each primary key value must be unique within the table.

Introduction to JDBC(cont..)

Joining Table

• The join operation works on the basis of common column between both table.

• The dept_id is the common between these two tabl

•Steps to create Join Application

- Create “Emp” table
- Create “Dept” table
- Develop the Java Application

Introduction to JDBC(cont..)

Joining Table Example

```
import java.sql.*;
class JoinExample
{
    public static final String DBURL =
"jdbc:oracle:thin:@localhost:1521:XE";
    public static final String DBUSER = "local";
    public static final String DBPASS = "test";
    public static void main(String args[])
    {
try
    {
     Class.forName("oracle.jdbc.driver.OracleDriver");
```

Introduction to JDBC(cont..)

Joining Table Example(cont..)

```
Connection con = DriverManager.getConnection(DBURL,
DBUSER, DBPASS)
        String sql = "select empname, city, department from
      emp e inner join dept d on e.dept_id = d.dept_id";
        Statement stmt = con.createStatement();
        ResultSet result = stmt.executeQuery(sql);
        System.out.println("EmpName  City   Department");
      while (result.next())
        {
            System.out.println (
                result.getString(1)+"   "+
                result.getString(2)+"    "+
                result.getString(3));
        }
```

Introduction to JDBC(cont..)

Joining Table Example(cont..)

```
}
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Introduction to JDBC(cont..)

- **Manipulating Databases with JDBC**

  - Inserting Records into Database

  - Updating Records into Database

  - Deleting Records into Database

Introduction to JDBC(cont..)

- **Inserting Records with JDBC**

```
try
{
Statement stmt = connection.createStatement(); // Prepare a
statement to insert a record
 String sql = "INSERT INTO my_table (col_string)
VALUES('a string')";
// Execute the insert statement
stmt.executeUpdate(sql);
 }
catch (SQLException e) { }
```

Introduction to JDBC(cont..)

- **Updating Records with JDBC**

```
try
{
Statement stmt = connection.createStatement(); // Prepare a
statement to update a record
String sql = "UPDATE my_table SET col_string='a new
string' WHERE col_string = 'a string'"; // Execute the insert
statement
int updateCount = stmt.executeUpdate(sql); // updateCount
contains the number of updated rows
}
catch (SQLException e) { }
```

Introduction to JDBC(cont..)

- **Deleting Records with JDBC**

```
try
 {
 Statement stmt = connection.createStatement(); // Use
TRUNCATE
String sql = "TRUNCATE my_table";
 // Use DELETE
 sql = "DELETE FROM my_table"; // Execute deletion
stmt.executeUpdate(sql);
}
catch (SQLException e) { }
```

Previous Topics: Recap

- The previous topic was focused on the concepts of Java Database Connectivity(JDBC) with its four types of driver in details.

- It was basically focused on using various types of SQL query in JDBC

UNIT 4

Topics: Database Connectivity (JDBC): Prepared Statements, Transaction Processing, Stored Procedures **(CO4)**

Objective of the above topics:

- To understand the concepts of Java Database Connectivity(JDBC)
- To understand the concepts of using various types of SQL query in JDBC
- To understand the concepts of stored Procedure in JDBC.
- To understand the joining multiple table in JDBC.

Introduction to JDBC(cont..)

- **PreparedStatement interface**

  - The PreparedStatement interface is a subinterface of Statement.

  - It is used to execute parameterized query.

  - The performance of the application will be faster if you use PreparedStatement interface

  - Easy to insert parameters into the SQL statement.

  - Enables easier batch updates.

Introduction to JDBC(cont..)

- **PreparedStatement interface Example**

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:th
in:@localhost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Em
p values(?,?)");
```

Introduction to JDBC(cont..)

- **PreparedStatement interface Example(cont..)**

```
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

Introduction to JDBC(cont..)

- **Transaction Processing Concepts**

  - **Transaction**

    - The transaction is a set of logically related operation.

    - It contains a group of tasks.

    - A transaction is an action or series of actions.

    - It is performed by a single user to perform operations for accessing the contents of the database.

Introduction to JDBC(cont..)

- **Transaction Processing Example**

  - Suppose an employee of bank transfers Rs 800 from X's account to Y's account.

  - This small transaction contains several low-level tasks:

X's Account

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

Introduction to JDBC(cont..)

- **Transaction Processing Example(cont..)**

Y's Account

Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)

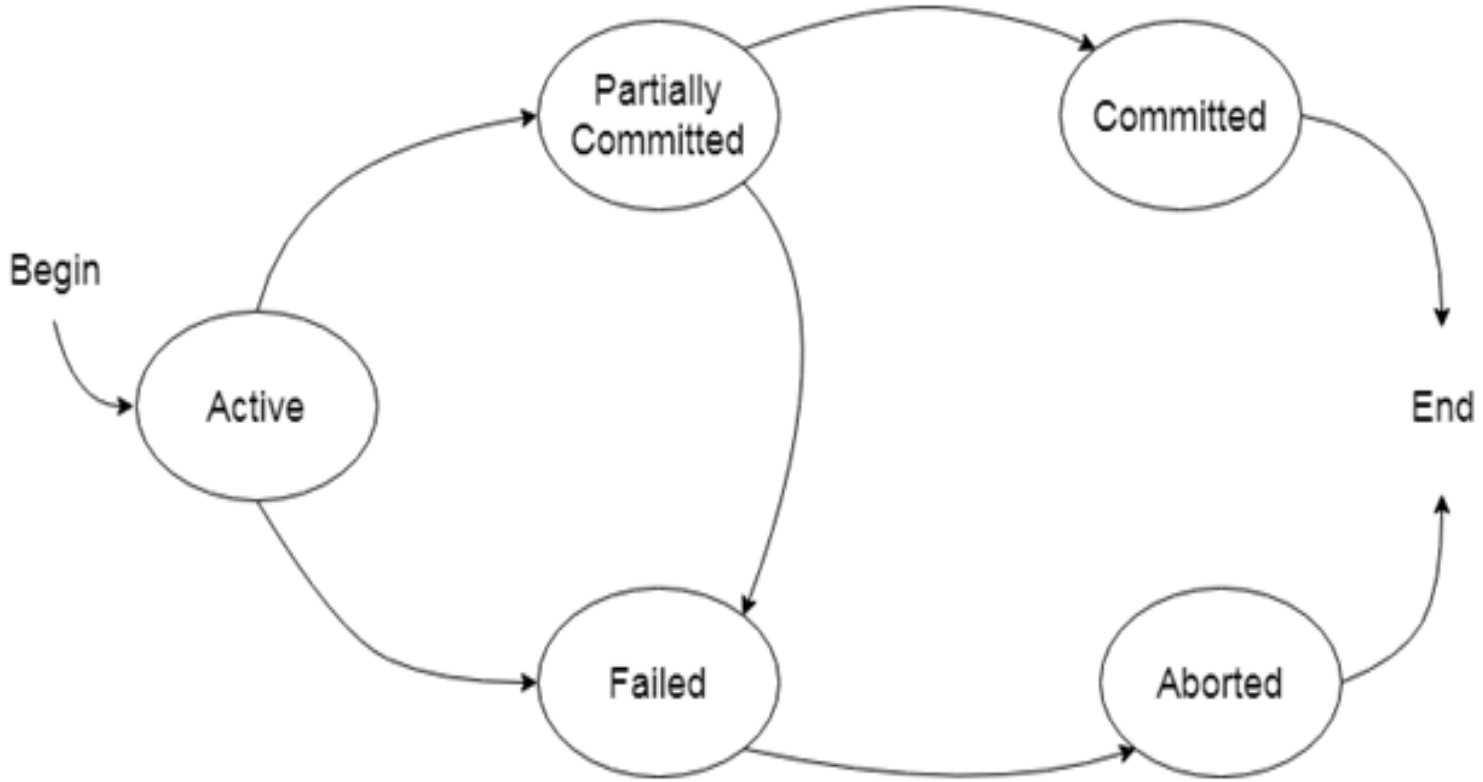
- **Operation of Transaction**

  - Read(x)

  - Write(x)

Introduction to JDBC(cont..)

- **States of Transaction**



Introduction to JDBC(cont..)

- **Transaction Management in JDBC**

•**Atomicity** means either all successful or none.

•**Consistency** ensures bringing the database from one consistent state to another consistent state.

•**Isolation** ensures that transaction is isolated from other transaction.

•**Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Introduction to JDBC(cont..)

- **Stored Procedure in JDBC**

  - CallableStatement interface is used to call the stored procedures and functions.

  - We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Introduction to JDBC(cont..)

- **Stored Procedure and Function**

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

Introduction to JDBC(cont..)

- **Example to call stored Procedure using JDBC**

```
create or replace procedure "INSERTR"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user420 values(id,name);
end;
/
```

Introduction to JDBC(cont..)

- **Complete Example of Stored Procedure using JDBC**

```
import java.sql.*;
public class Proc {
public static void main(String[] ar
gs) throws Exception{

Class.forName("oracle.jdbc.driver.
OracleDriver");
Connection con=DriverManager.g
etConnection(
"jdbc:oracle:thin:@localhost:1521:
xe","system","oracle");

CallableStatement stmt=c
on.prepareCall("{call inser
tR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();

System.out.println("succes
s");
}
}
```

Previous Topics: Recap

- It was discussed about the concepts of stored Procedure in JDBC.
- It was also discussed about joining multiple table in JDBC.