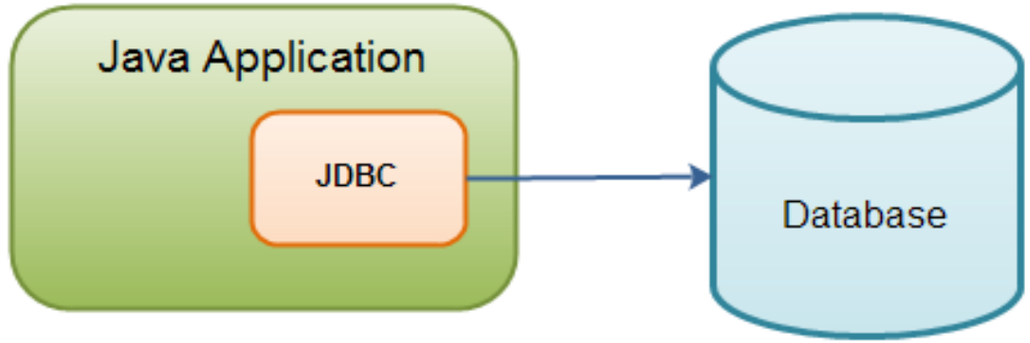
JDBC Object

Presented By: Pankaj Kuma

Java Database Connectivity (JDBC)

- **JDBC** is an **Application Programming Interface(API)** used to connect Java application with Database.
- JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL



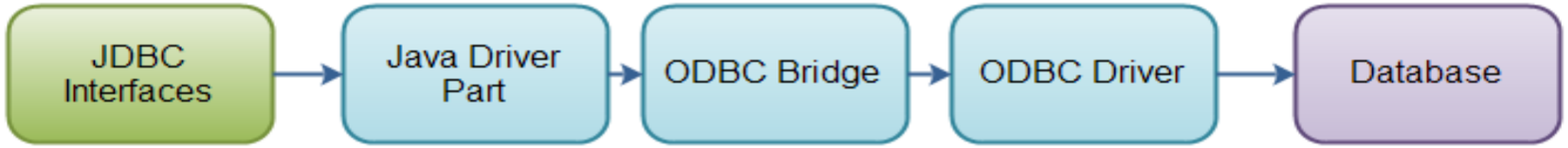
JDBC Driver Types

JDBC Driver Types

1. Type-1 Driver or JDBC-ODBC bridge
2. Type-2 Driver or Native API Partly Java Driver
3. Type-3 Driver or Network Protocol Driver
4. Type-4 Driver or Thin Driver

1. Type-1 Driver or JDBC-ODBC bridge

- Type 1 JDBC driver consists of a Java part that translates the JDBC interface calls to ODBC calls
- JDBC-ODBC bridge driver uses ODBC driver to connect to the database.
- JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.



Advantage and Disadvantage of type 1 driver

Advantages:

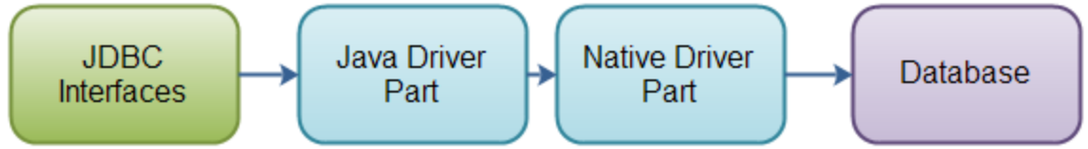
- ❖ easy to use.
- ❖ can be easily connected to any database.

Disadvantages:

- ❖ Performance degraded
- ❖ The ODBC driver needs to be installed on the client machine.

2. Type 2 JAVA / Native Code Driver

- Type 2 JDBC driver convert JDBC calls into database calls by using native API provided by database.
- Native API driver uses the client -side libraries of the database.
- In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.



Advantage and Disadvantage of type 2 driver

Advantage:

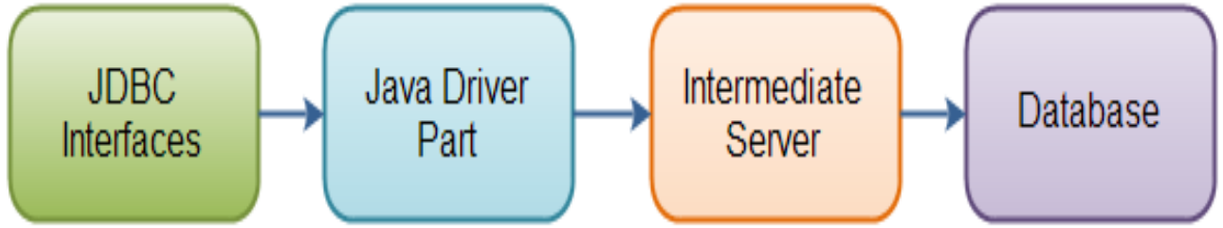
- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- Type 2 driver is platform dependent.

3. Type-3 Driver or Network Protocol Drive

- Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol



Advantage and Disadvantage of type 3 driver

Advantage

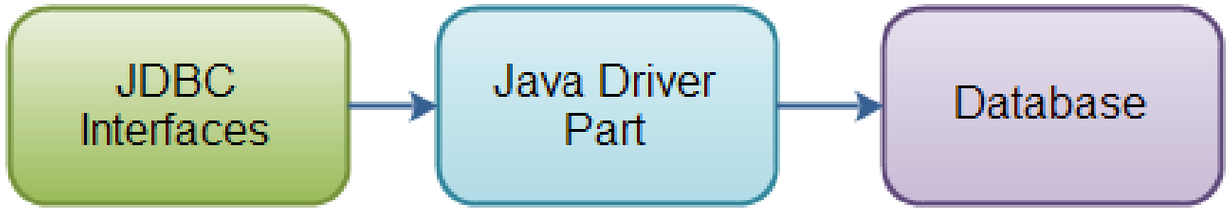
- Does not require any native library to be installed.
- Database Independency.

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.

4. Type-4 Driver or Thin Driver

- This is Driver called Pure Java Driver because this driver interact directly with database.
- It does not require any native database library, that is why it is also known as Thin Driver



Advantage and Disadvantage of type 4 driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database.

JDBC Process

JDBC Process

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Loading the JDBC Driver
- Connecting to the DBMS
- Creating and executing a statement
- Processing data returned by the DBMS
- Terminating data returned by the DBMS

1. Loading the JDBC Driver

- JDBC driver must be loaded before the J2EE component can connect to the DBMS.
- The **Class.forName()** method is used to load the JDBC driver.

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

2. Connect to the DBMS

- Once driver is loaded the J2EE component must connect to the DBMS using DriverManager.getConnection() method.
- getConnection() returns Connection interface.
- For getConnection() method we need to pass three parameters.
    - url
    - username
    - Password
- Syntax: Connection connect=DriverManger.getConnection(String url, Sting user, String password)

3. Creating and Executing a statement

- After the JDBC is loaded and connection is successfully made with a particular database managed by the dbms, is to send a particular query to the DBMS for processing.
- SQL query consists series of SQL command that direct DBMS to do something example Return rows of data.
- **Statement stmt= connect.createStatement()** method is used to create a statement Object.
- The statement object is then used to execute a query and return result object that contain response from the DBMS

Contd.. Retrieves all the rows from customer table

```
try {

String query="select * from Customers";

Statement stmt= connect.createStatement()

ResultSet Results = stmt.executeQuery(query);

}
```

4. Process data returned by the DBMS

- **java.sql.ResultSet** object is assigned the result received from the DBMS after the query is processed.
- **java.sql.ResultSet** contain method to interact with data that is returned by the DBMS to the J2EE Component.
- call **next()** method of **ResultSet.**
- If returned value **next() is 0 it indicates that no** row are present in the table.
- **getString()** of ResultSet object is used to copy the value of a specified in the current row of the ResultSet to aString object

Retrieving data from ResultSet

```
ResultSet Result;
String FirstName;
String LastName;
String printrow;
boolean Records=Results.next();
if(!Records) {
S.O.P(“no data found”);
return; }
else {
FirstName= Results.getString(FirstName);
LastName= Results.getString(LastName);
printrow=FirstName + “  “ +LastName;;
S.O.P(printrow);     }
```

5. Terminate the Connection to the DBMS

- The connection to the DBMS is terminated by using the **close()** method of the **Connection** object once the J2EE component is finished accessing the DBMS.
- **Db.close();**

Database Connection

Database Connection

A Database connection is a facility in computer science that allows client software to talk to database server software, whether on the same machine or not.

Code to load and register driver class

```
Class Demo{

p.s.v.m(String args[]){

try {

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

S.O.P(“Driver loaded & Registered Succesfully”);

}

Catch(ClassNotFoundException e){

S.O.P(e);

}}}
```

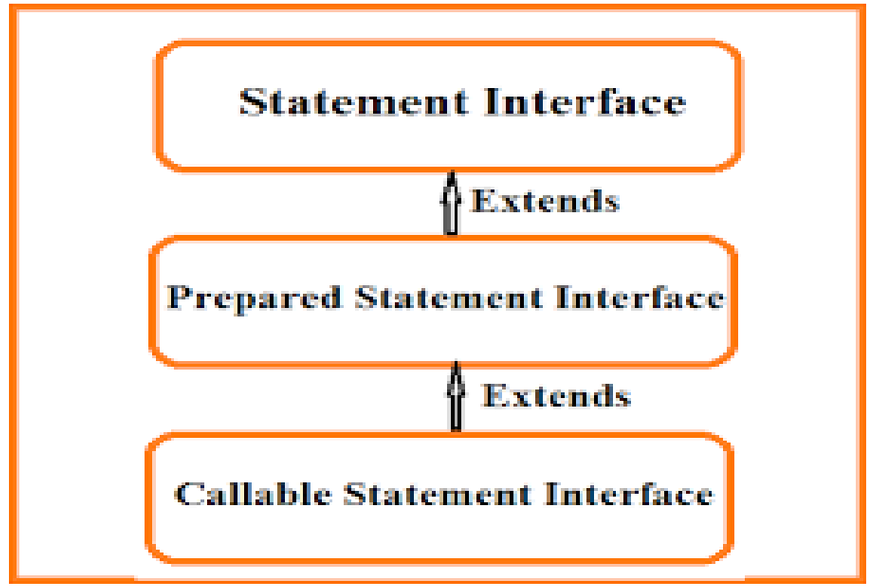
Connecting the database using url, userId, password

```
Class Demo{

p.s.v.m(String args[]){

try {

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

S.O.P("Driver loaded & Registered Succesfully");

Connection connect=DriverManger.getConnection(("jdbc:mysql//localhost:3306?  User=root &
password = manoj )

}

Catch(ClassNotFoundException e){

S.O.P(e);

}}}
```

Statement Objects

Three Statement objects are used to execute the query:

- **Statement**: Executes a query immediately.
- **PreparedStatement**: used to execute a compiled query.
- **CallableStatement**: Used to execute store procedures.



Statement Object

Once we created a Statement object, we can then use it to execute an SQL statement with one of its three execute methods

1. **ResultSet execute Query(String query):** It executes the given SQL statement, which returns ResultSet object.
2. **int execute Update (String query):** Returns the number of rows affected by the execution of the SQL statement. It execute the given SQL which may be an Insert, Update or Delete statement (DML Statements) and DDL statements
3. **boolean execute (String query):** Returns a boolean value of true if a ResultSet object can be retrieved.Used to execute any kind of SQL quries

Structure of the program

```
import java.sql.*;
public class Test    {
    public static void main(String args []) throws Exception     {
          //step-l load the driver (Type-l)
          Class.forName ( “com.mysql.jdbc.Driver" ) ;
          //step-2 get the connection
          Connection cn=DriverManager.getConnection (url, username, password);
          //step-3 create Statement, object
          Statement st=cn.createStatement();
         //step-4 execute sql statement
         ResultSet  rs=st .executeQuery ("select * from emp");
         //step-5 print the output
        }
}
```

Text Book Example:

Using statement object to execute a query

```
String url=”jdbc:odbc:CustomerInformation”;
String userId=”jim”
String password=”Keogh”;
Statement DataRequest;
Connection Db;
ResultSet Results;
try {
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
Db=DriverManager.getConnection(url, userId, password);
}
catch(ClassNotFoundException error) {
S.O.P(“Unable to load JDBC/ODBC bridge”+error);
System.exit(1);
}
Catch(SQLException error) {
S.O.P(“cannot connect to the database”+error);
System.exit(2);   }
```

```
try{
String query=”SELECT * FROM Customers;
DataRequest=Db.createStaement();
Results=DataRequest.executeQuery(query);
DataRequest.close();
} catch(SQLException err)
{
System.err.println(“Error”);
System.exit(3);
}
Db.close();}
```

Example:

Another method is used when DML and DDL operations are used for processing query is executeUpdate(). This returns no of rows as integer.

Another method is used when DML and DDL operations are used for processing query is executeUpdate(). This returns no of rows as integer.

```
// write code to load driver
//write code to connect to the database
try{
String query=”UPDATE Customer set PAID=’Y’ where BALANCE =’0’;
DataRequest=Db.createStaement();
int rowUpdated=DatRequest.executeUpdate(query);// returns no of rows updated
S.O.P(rowUpdated);
}catch(SQLException err)
{
System.err.println(“Error”);
System.exit(1);
}
```

Statement interface Example

(other source)

```
class Demo {
public static void main(String args[]) {
        Connection con=null;
        Statement stmt=null;
        String qry="insert into student values(5,"manoj","cs&e");
try{
Class.forName("com.mysql.jdbc.Driver");
S.O.P("driver loaded successfully");
con=DriverManager.getConnection("jdbc:mysql//localhost:3306? User=root & password = manoj);
S.O.P("connection established");
Stmt=con.createStatement();
S.O.P("platform established");
stmt.executeUpdate(qry);
S.O.P("Data inserted sucessully");
}
```

```
Catch(ClassNotFoundException / SQLException e)
{
S.O.P(e);
}
// Close the resources
finally{
if(stmt!=null){
                    try{
                              stmt.close;
                         }  Catch(SQLException e)    {   s.o.p(e)(e);   }  }
 if(con!=null) {
                     try
                               {   con.close();   }    catch(SQLException e)    {   s.o.p(e);   } }}
```

PreparedStatement

PreparedStatement

- PreparedStatement is used to execute dynamic or parameterized SQL queries.
- PreparedStatement extends Statement interface.
- We can pass the parameters to SQL query at run time.
- It is recommended to use PreparedStatement if you are executing a particular SQL query multiple times.
- It gives better performance than Statement interface. Because, PreparedStatement are precompiled and the query plan is created only once irrespective of how many times you are executing that query. This will save lots of time.

Place Holder in PreaparedStatement

- It is the parameter which holds the value dynamically at the run time in prepared statement.
- In the JDBC place holder is represented as **?(Question mark)**
- It takes two arguments one is position of question mark and other is value to the filed

Example:

String qry="insert into student values(?,?,?);

String qry2="delete from student where id=?"

String qry3="update student set name=? where dept=?"

Setting the data for placeholder in PreparedStatement

- The data must be set to the placeholder before execution.
- The number of data must exactly match the number of placeholder.
- To set the data for placeholder we have to use setXXX method.

syntax: public void setXXX(int placeholder_index_number, XXXdata)

Example:

void setString(int placeholder_index_number, String_data);

void setInt(int placeholder_index_number, int_data);

Example 1 for PreparedStatement

String qry="insert into student values(?,?,?)";

try{

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

S.O.P("driver loaded successfully");

Connection con=DriverManager.getConnection("jdbc:mysql//localhost:3306? User=root &

password = manoj);

S.O.P("connection established");

PreparedStatement pstmt=con. prepareStatement(qry);

S.O.P("platform established");

```
pstmt.setInt(1,101);

pstmt.setString(2,"Ratan");

pstmt.setDouble(3,44.44);

pstmt.executeUpdate();

pstmt.setInt(1,102);

pstmt.setString(2,“Tata");

pstmt.setDouble(3,55.44);

pstmt.executeUpdate();
} catch Catch(ClassNotFoundException / SQLException e)
{
S.O.P(e);
}
```

```
// Close the resources
finally{
if(pstmt!=null){
        try{
            pstmt.close;
          }  Catch(SQLException e)    {   s.o.p(e);   }  }
 if(con!=null) {
          try {
              con.close();
            }    catch(SQLException e)    {   s.o.p(e);   } }}
```

Example 2 for PreparedStatement

```
try{
//code to load the drivers
//code to get the connection to database
        String query=”SELECT * FROM Customers where cno=?”;
        PreparedStatement pstatement=db.preparedStatement(query);
        pstatement.setString( 1,”123”); // 1 represents first place holder, 123 is value
        rs= pstatement.executeQuery();
}catch(SQLException err)
{
System.err.println(err); }
//Write a code Close the resources
```

Callable Statement

Callable Statement

- CallableStatement object is used to **call a stored procedure** within J2EE object.
- Stored procedure is block of code and is identified by a unique name
- The callableStatement uses three types of parameter when calling stored procedure. The parameters are **IN , OUT,INOUT.**
- IN parameter contains data that needs to be passed to the stored procedure whose value is assigned using setxxx() method.
- OUT parameter contains value returned by stored procedure.the OUT parameter should be registers by using **registerOutParameter()** method and then later retrieved by the J2EE component using **getxxx()** method.
- INOUT parameter is used to both pass information to the stored procedure and retrieve the information from the procedure.

```
try{
// code to load driver
//code to connect to the database
String lastOrderNumber;
        String query= "{ CALL LastOrderNumber (?) }" ;
CallableStatement Cstatement = Db.prepareCall(query);
Cstatement.registerOutParameter(1, Types. VARCHAR);
Cstatement.execute();
lastOrderNumber=Cstatement.getString(1);
Cstatement.close();
}catch(SQLException err)
{
System.err.println("Error");
System.exit(1);   }
```

ResultSet

ResultSet

- The JDBC ResultSet is used to handle the result returned from the SQL select statement.
- The SQL select statements reads data from a database and return the data in a result set.
- The result from a select statement is in a tabular form. It has columns and rows.
- A ResultSet object maintains a cursor that points to the current row in the result set.

Contd…

- getString() method is called to retrieve values from each of the columns ResultSet.
- Results.getString(1); to retrieve first column
- The next() method is called in the looping statement to move the virtual cursor to the next row in the ResultSet and determine if there is data in that row.

```
// code to load driver
//code to connect to the database
String query = "SELECT  Id, FirstName, LastName  FROM Employees";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);
while(rs.next()) {
                    int id = rs.getInt(“Id");        / / rs.getInt(1);
                    String first = rs.getString(“FirstName");
                    String last = rs.getString(“LastName");
                    System.out.print("ID: " + id);
                    System.out.print(", First: " + first);
                    System.out.println(", Last: " + last);
                     }
```

//Write a code to Close the resources

Scrollable ResultSet

Scrollable Result Set

- A scrollable ResultSet is one which allows us to retrieve the data in forward direction as well as backward direction
- There are six methods to position the cursor at specific location in addition to next() in scrollable result set.
  - first() …… position at first row.
  - last()……position at last row.
  - previous()…….position at previous row.
  - absolute()….positions the cursor to the row number specified in the absolute function
  - relative()…… move relative to current row. Positive and negative number can be given. Ex. relative(-4) … 4 position backward direction.
  - getRow() …… returns the number of current row

Contd…

- There are three constants can be passed to the createStatement()

  1.TYPE_FORWARD_ONLY.

  2.TYPE_SCROLL_INSENSITIVE

  3.TYPE_SCROLL_SENSITIVE

- TYPE_SCROLL_ INSENSITIVE and TYPE_SCROLL_ SENSITIVE makes cursor to move both direction.

- INSENSITIVE makes changes made by J2EE component will not reflect. SENSITIVE means changes by J2EE will reflect in the result set.

```
try {
// code to load driver          //code to connect to the database
String query= "SELECT FirstName, LastName FROM Customer;
Statement stmt=con.createStatement(TYPE_SCROLL_INSENSITIVE);
ResultSet rs=stmt.executeQuery(query);
while(rs.next()) {
                        rs.first();
                        rs.last();
                        rs.previous();
                        rs.absolute(10);
                        rs.relative(-2);
                        rs.relative(2);
                        String FirstName=rs.getString(1);
                        String LastName=rs.getString(2);
                        String printrow= FirstName + " " +LastName;
                        S.O.P(printrow);
                    }
}
//Write a code to Close the resources
```

Updatable Result Set

Updatable Result Set

- An updatable result set allows modification to data in a table through the result set.
- This is possible by sending CONCUR_UPDATABLE
- Three ways in which result set can be changed.
    - updating row
    - deleting a row
    - inserting a new row

Update ResultSet

- Once the executeQuery() method of the statement object returns a result set. updatexxx() method is used to change the value of column in the current row of result set.
- It requires two parameters,

    1. position of the column in query.

    2.parameter is value

- updateRow() method is called after all the updatexxx() methods are called.

```
String selectSql = "SELECT * FROM employee_details";
ResultSet resultSet = null;
try {
Class.forName(“com.mysql.jdbc.Driver”);
Connection  connection=DriverManager.getConnection(“jdbc:mysql//localhost:3306? User=root & password = manoj);
Statement  statement = connection.createStatement(ResultSet.CONCUR_UPDATABLE);
resultSet = statement.executeQuery(selectSql);
while (resultSet.next()) {
int salary = resultSet.getInt("salary");
salary = salary + 500;
resultSet.updateInt("salary", salary);
resultSet.updateRow();
}  } catch (SQLException e) {
System.out.println("An exception occured.Exception is :: " + e);
```

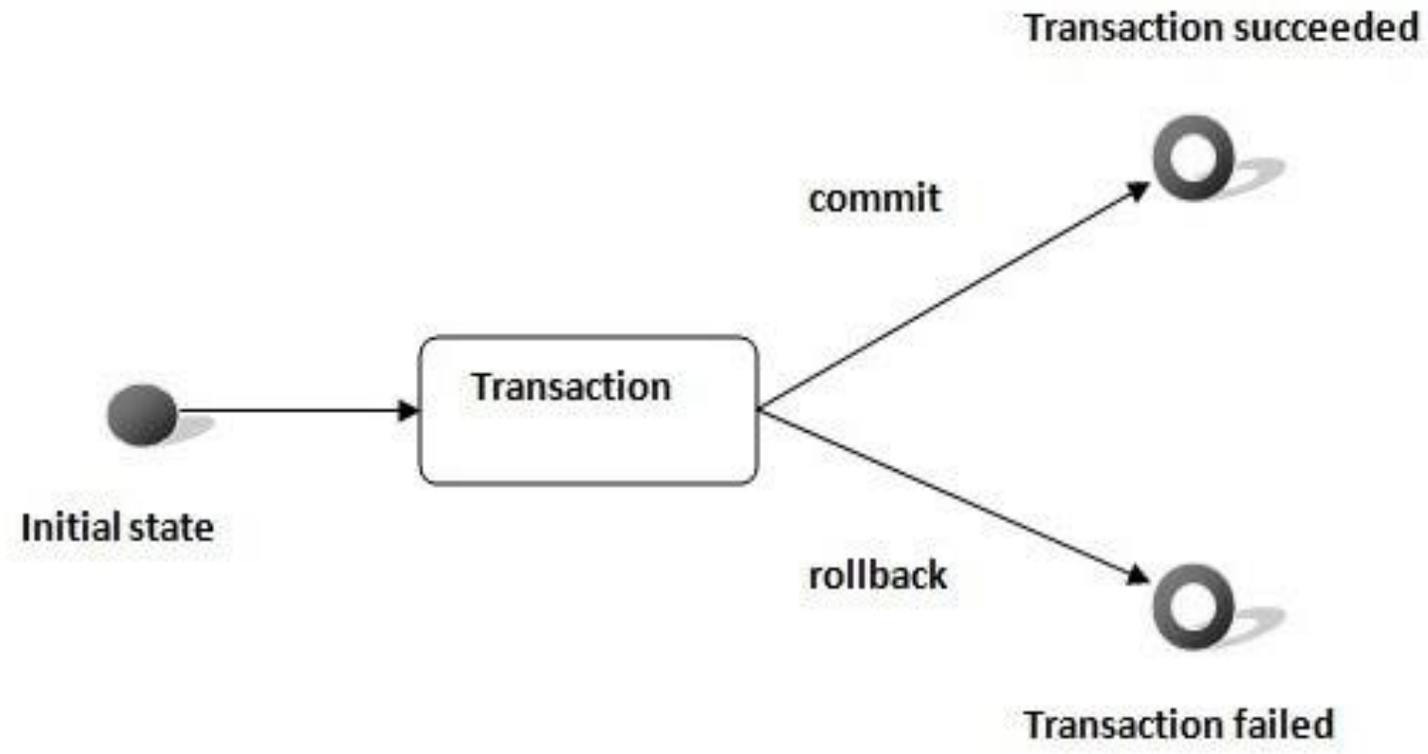
//Write a code to Close the resources

Transaction Processing

Presented By: Pankaj Kum

Transaction Processing

- A transaction is a group of operations used to perform a particular task.
- In a transaction if one operation fails then all operations of the transaction gets cancelled. Finally, the transaction status fails.
- If all operations in a transaction are successfully executed, then only the transaction status is successful.
- A transaction follows the principle of -All or Nothing.
- For example, booking a movie ticket online is a transaction.


Transaction succeeded
commit
Transaction
Initial state
rollback
Transaction failed

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

- **Atomicity** : means either all successful or none.
- **Consistency :** After the transaction is completed successfully or fails, the data left in the database, should be reliable. Reliable data is called consistent data.
- **Isolation** : Isolation means, if two transaction are going on same data then one transaction will not disturb another transaction.
- **Durability :** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

In JDBC, **Connection interface** provides methods to manage transaction.

Method	**Description**
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Example for Executing a database transaction

```
try {
Class.forName(“com.mysql.jdbc.Driver”);
Connection  Database=DriverManager.getConnection(“jdbc:mysql//localhost:3306? User=root & password = manoj);
Statement  DataRequest1,DataRequest2;
Database.setAutoCommit(false);
String query1=“UPDATE Customers SET Street=‘5 Main Street’” +” WHERE FirstName=‘tin’  “;
String query2=“UPDATE Customers SET Street=‘10 Main Street’” +” WHERE FirstName=‘bob’  “;
DataRequest1=Database.createStatement();
DataRequest2=Database.createStatement();
DataRequest1.excuteUpdate(query1);
DataRequest2.excuteUpdate(query2);
Database.commit();
DataRequest1.close();
DataRequest2.close();
Database.close(); }
```

```
Catch(SQLException ex) {
S.O.P(ex);
if(con!=null) {
try {
S.O.P("transaction is being rolled back");
con.rollback();
}
catch(SQLException excep) {
S.O.P(excep); }
} }
```

SavePoint

- The SavePoint is a logical position in a transaction up to which we can rollback the transaction.
- When the save point is placed in the middle of the transaction, the logics placed before the save point will be committed and the logics placed after the save point will be rolled back

```
try {
Class.forName("com.mysql.jdbc.Driver");
Connection  Database=DriverManager.getConnection("jdbc:mysql//localhost:3306?  User=root & password = manoj);
Statement   DataRequest1,DataRequest2;
Database.setAutoCommit(false);
String query1="UPDATE Customers SET Street='5 Main Street'" +" WHERE FirstName='tin'  ";
String query2="UPDATE Customers SET Street='10 Main Street'" +" WHERE FirstName='bob'  ";
DataRequest1=Database.createStatement();
SavePoint s1=Database.setSavePoint("spl")     // spl is the parameter to rollback
DataRequest2=Database.createStatement();
DataRequest1.excuteUpdate(query1);
DataRequest2.excuteUpdate(query2);
Database.commit();
DataRequest1.close();    DataRequest2.close();    Database.close();  }
```

```
Catch(SQLException ex) {

S.O.P(ex);

try {

Database.rollback(spl);

S.O.P("transaction is being rolled back");

}

catch(SQLException excep) {

S.O.P(excep); }

} }
```

Metadata

- Metadata is data about data.
- MetaData is accessed by using the DatabaseMetaData interface.
- Database metadata interface is used to retrieve information about databases, tables, columns, indexes etc.

The method used to retrieve meta data informations are :

1. **getDatabaseProductname() :** Returns the product name of the database
2. **getUserName():** Returns the userName
3. **getURL():** Returns the URL of the Database
4. **getSchemas():** Returns all the scheme name available in this database
5. **getPrimaryKeys() :** Returns primary Keys
6. **getProcdures():** Returns stored procedures names
7. **getTables():** Returns names of thtables in the database

ResultSet MetaData

- Describes the result set
- Meta Data that describes the ResultSet is retrieved by calling getMetaData() method of ResultSet object.
- ResultsetMetaData rm=Result.getMetaData();

Commonly used Methods in ResultSetMetaData:

- **getColumn Count():** Returns the number of column contained in the ResultSrt
- **getColumnName(int number):** Returns the Name of the coumnspecified by the column number
- **getColumnType(int number)** Returns the Data type of the column specified by the column number

Data Types

- The setXXX() and getXXX() methods are used to set the value of specific data type and to retrieve the value of specific data type.
- The XXX in the names of these methods is replaced with the name of the data type.

SQL data type	Java types
CHARACTER	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float
FLOAT	Double
DOUBLE PRECISION	Double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

List of data types for use with setXXX() and getXXX()

Exceptions in jdbc

- There are three kind of exception thrown by jdbc methods.
    1. SQLException ,
    2. SQLWarnings
    3. DataTruncation
- SQLException which result in SQL syntax errors.
- getNextException() method of SQLExceptions object returns details about the SQL error.
- getErrorCode() method of SQLExceptions object is used to retrieves vendor specific error codes.

- **SQLWarnings :**

    1. it throws warnings related to connection from DBMS.
    2. **getWarnings()** method of connecction object retrieves the warnings and **getNextWarnings()** returns subsequent warnings.

- **DataTruncation :**

    1. Whenever data is lost due to truncation of the data value , a truncation exception is thrown.

Important Questions

1. List and Explain the various JDBC Driver Types
2. Explain the Basic steps involved in the JDBC process
3. Explain the various steps of JDBC with code snippets
4. List and explain the various Statement objects in JDBC
5. Explain the following with an example:
    - PreparedStatement object
    - CallableStatement object
6. What is ResultSet? Explain the types of ResultSets in JDBC
7. Explain the types of exception occurred in JDBC
8. Write a java program to execute a database transaction
9. Explain MetaData