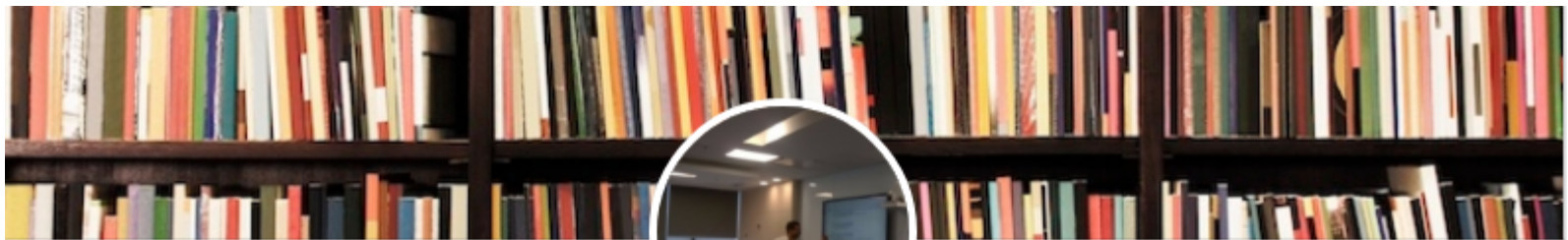


JDBC Java Training

Rajeev Gupta

Java Trainer & Consultant (MTech CS)


rgupta.mtech@gmail.com



Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

freelance • Institution of Electronics and Telecommunication Engineers IETE

New Delhi Area, India • 500+ 

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7, Spring 5, Hibernate 5, Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Topics

- Introduction
 - Type of drivers
 - JDBC API
 - Hello World Ex.
 - CRUD operations
- Statement, prepare Statement, callable statement
- ResultSetMetaData & DatabaseMetaData
- Scrollable ResultSet
- Auto Generated Keys
- Transaction Management
- Connection Pooling concept
- Data Access Object (DAO) Pattern
- Introduction to `javax.sql.RowSet`



Introduction

Type of drivers

JDBC API

Hello World Ex.

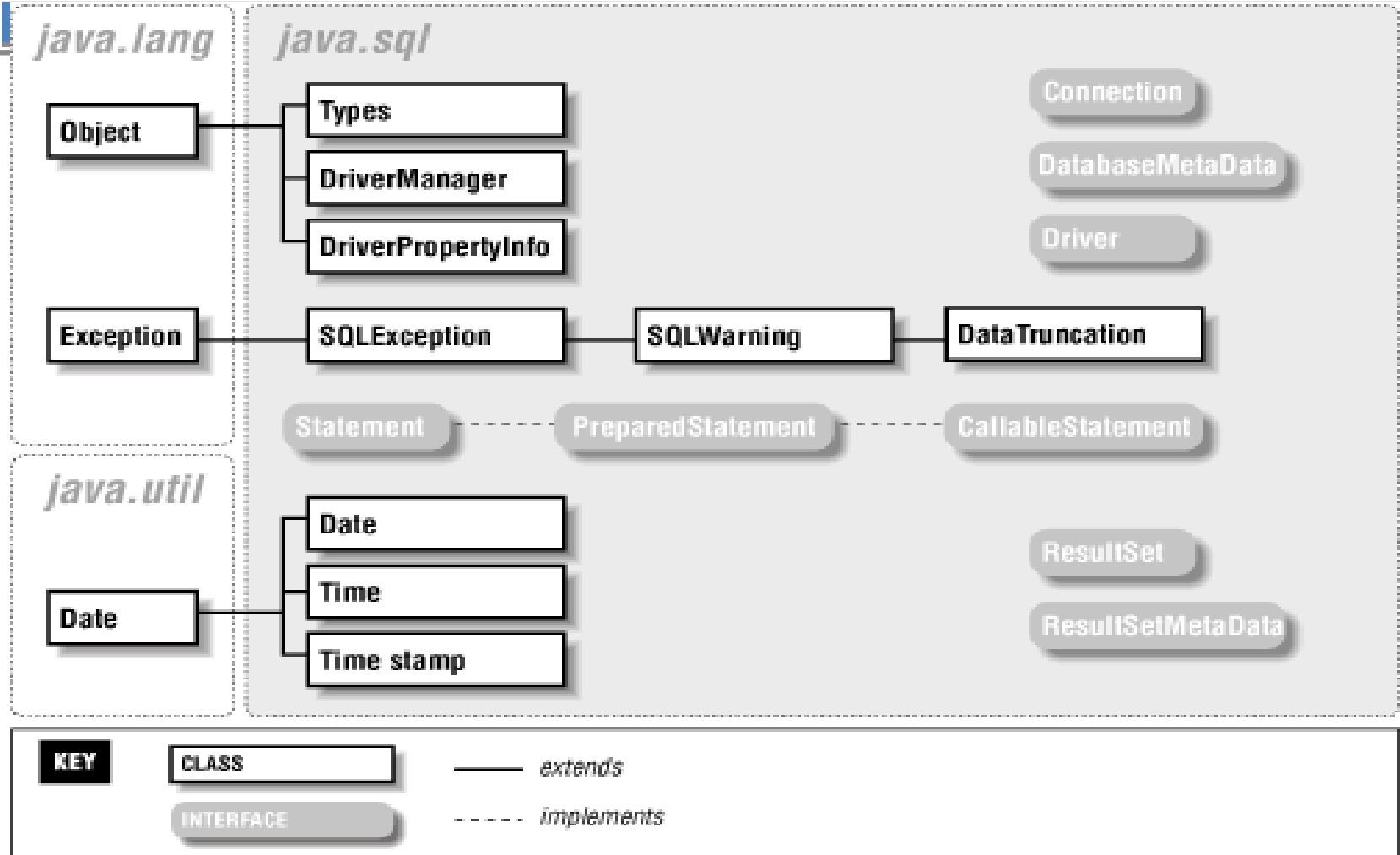
CRUD operations

Auto Generated Keys

JDBC introduction

- JDBC is a specification of Sun Microsystems for database connectivity , Vender is going to implement it.
- JDBC API has two major packages
 - java.sql
 - javax.sql.
- JDBC architecture consist of different layers and drivers which are capable of working with any database.

java.sql.* packages



java.sql.* packages

□ JDBC Interfaces

- Connection
- Driver
- Statement->PreparedStatement->CallableStatement
- ResultSet
- DatabaseMetaData
- ResultSetMetaData
- ResultSet

□ JDBC Classes

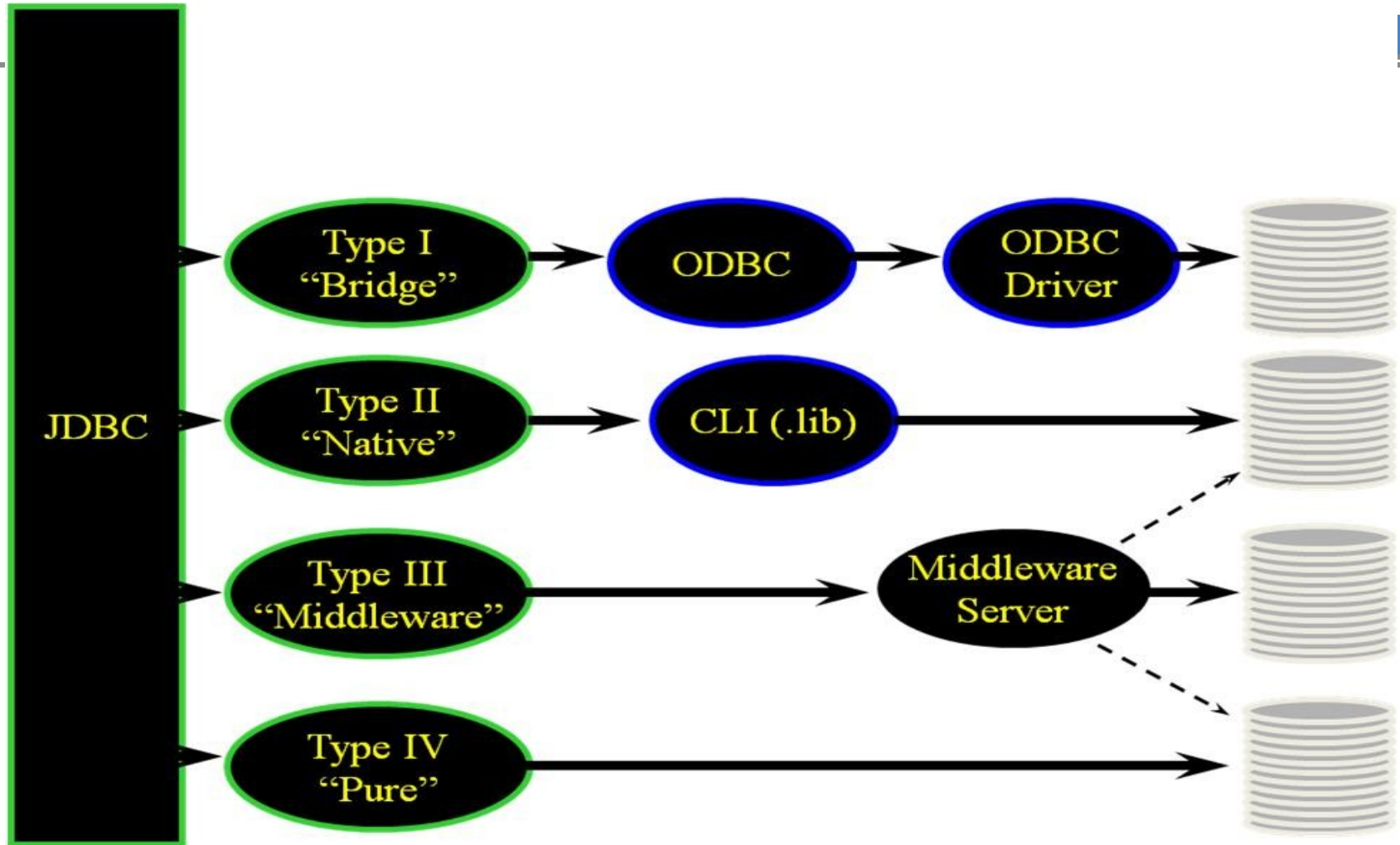
- Throwable==>Exception==>SQLException==>SQLWarning
- Date(java.lang.*)==>Date,Time,TimeStamp
- DriverManager
- DriverPropertyInfo

Driver & DriverManager

- Driver:
 - Ensure consistent and uniform access to any database
 - Driver act as a translator bw client request and database access

- Driver Manager
 - Class that manages establishment of connection
 - Driver Manager need to told which JDBC driver it should try to connect

Type of Drivers



Type of Drivers

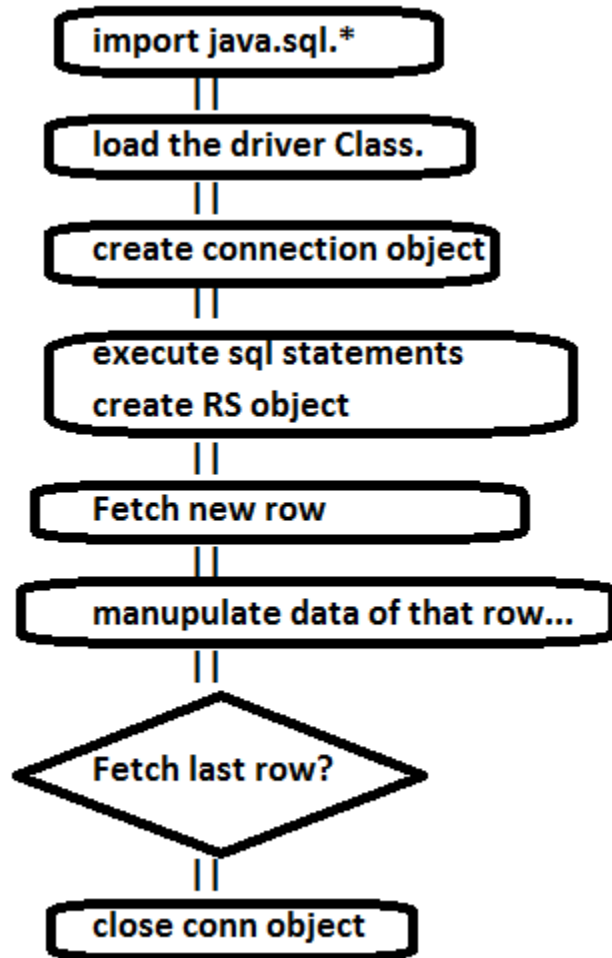
- 1. JDBC-ODBC Bridge (Type 1)
 - ODBC must be configured on client machine
 - OS dependents: ODBC need to purchases for other OS

- 2. Type 2 Driver
 - No ODBC is required
 - We need native client files(script file) form vendor to make db call
 - Platform dependent

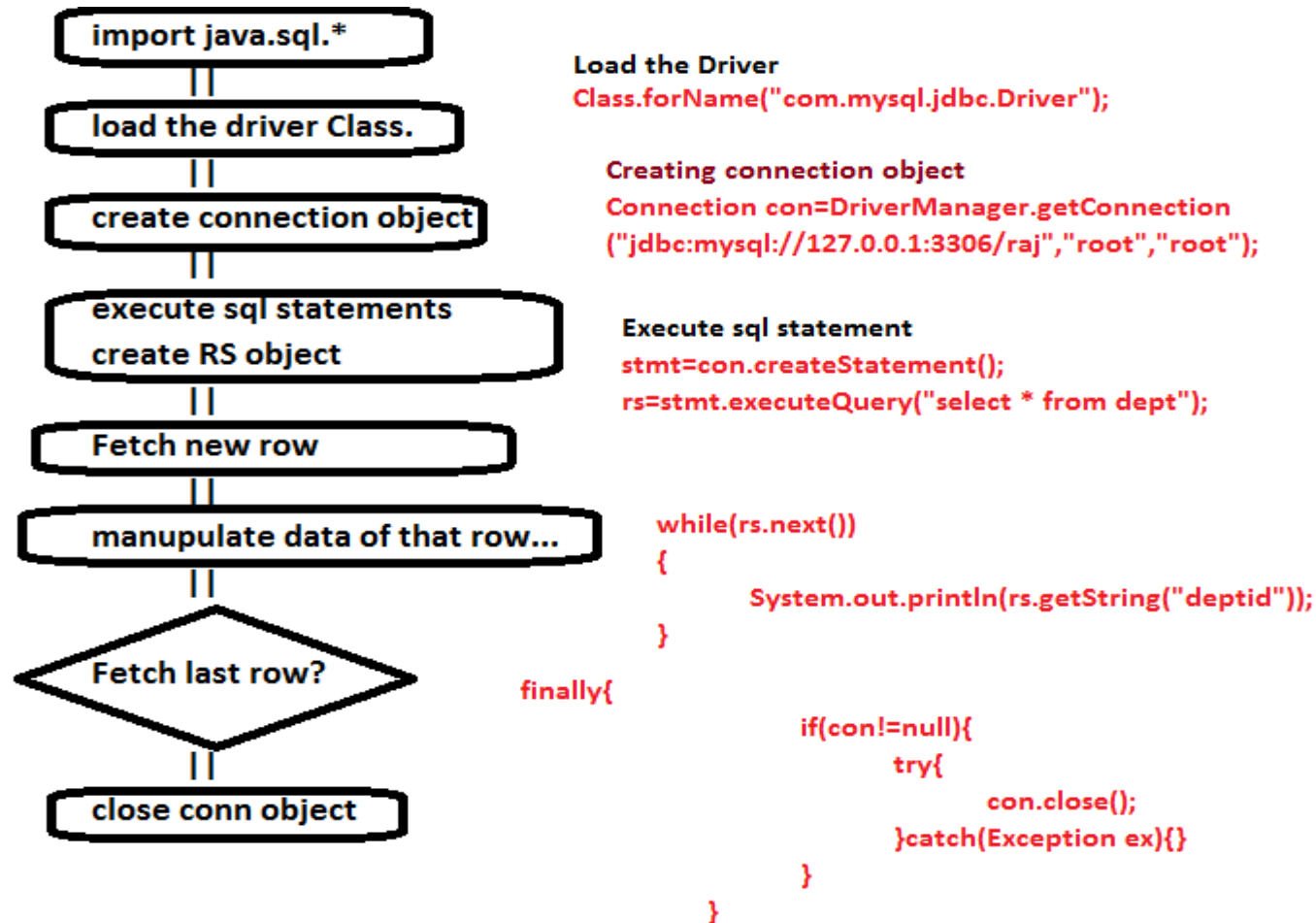
Rajeev Gupta Java / JEE Corporate trainer

- 3. Type 3 Driver

JDBC Steps



JDBC Steps



```

public class HelloWorld
{
    public static void main(String[] args)
    {
        Connection con=null;
        Statement stmt=null;
        ResultSet rs=null;

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

            }
            catch(ClassNotFoundException ex){}

            con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/raj","root","root");
            stmt=con.createStatement();
            rs=stmt.executeQuery("select * from dept");
            while(rs.next()){
                System.out.print(rs.getString(1));
                System.out.println(rs.getString(2));
            }
        }
        catch(SQLException ex){
            ex.printStackTrace();
        }
        catch(Exception ex){
            ex.printStackTrace();
        }
        finally{
            if(con!=null){
                try{
                    con.close();
                }
                catch(Exception ex){}
            }
        }
    }
}

```

CRUD Operations

```
stmt.executeUpdate("Create Table  
dept (deptid varchar(20) NOT  
NULL,name varchar(45))");  
System.out.println("Table  
Created");
```

/** inserting Rows */

```
stmt=con.createStatement();  
int i=stmt.executeUpdate("insert into  
dept(deptid,deptName)values(234,'f  
oo')");
```

Prepared statements

- Refer to data values in the query string using placeholders
- Then tell JDBC to bind data values to the placeholder and it will handles any special char accordingly
- Being a subclass of Statement, PreparedStatement inherits all the functionality of Statement.
- The three methods are execute(), executeQuery() and executeUpdate()

Callable Statement

- Sometimes the it is not possible to retrieve the whole manipulated data with a single query from database. So many queries clubbed together to form a single block. This block is known as procedures in database.
- So if you want to call these blocks from your java programs so you need to use a special statement i.e Callable Statement.
- Creating hello world procedure in mysql:

```
DELIMITER //  
CREATE PROCEDURE Test(IN num1 INT, IN num2 INT ,OUT param1 INT)  
BEGIN  
set param1 := num1 + num2;  
END  
//  
DELIMITER :
```

Rajeev Gupta Java / JEE Corporate trainer


```
public class DemoStoredProcedure {
    public static void main(String[] args) {
        Connection con=null;
        CallableStatement cstmt=null;
        ResultSet rs=null;

        try{
            try{
                Class.forName("com.mysql.jdbc.Driver");
            }
            catch(ClassNotFoundException ex){}

            con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/raj","root","root");
            cstmt=con.prepareCall("call test(?,?,?)");
            cstmt.setInt(1, 3);
            cstmt.setInt(2, 44);
            cstmt.registerOutParameter(3, Types.INTEGER);
            cstmt.execute();
            System.out.println("Result:"+cstmt.getInt(3));
        }
        catch(Exception ex){}
        finally{
            if(con!=null){
                try{
                    con.close();
                }
                catch(Exception ex){}
            }
        }
    }
}
```



ResultSetMetaData & DatabaseMetaData

Metadata from DB

- A Connection's database is able to provide schema information describing its tables, its supported SQL grammar, its stored procedures the capabilities of this connection, and so on
- This information is made available through a DatabaseMetaData object.

Metadata from DB - example

```
...  
Connection con = .... ;
```

```
DatabaseMetaData dbmd = con.getMetaData();
```

```
String catalog = null;  
String schema = null;  
String table = "sys%";  
String[ ] types = null;
```

```
ResultSet rs =  
    dbmd.getTables(catalog , schema , table , types );  
...
```

JDBC – Metadata from RS

```
public static void printRS(ResultSet rs) throws SQLException
{
    ResultSetMetaData md = rs.getMetaData();
    // get number of columns
    int nCols = md.getColumnCount();
    // print column names
    for(int i=1; i < nCols; ++i)
        System.out.print( md.getColumnName( i)+",");
    // output resultset
    while ( rs.next() )
    {
        for(int i=1; i < nCols; ++i)
            System.out.print( rs.getString( i)+",");
        System.out.println( rs.getString(nCols) );
    }
}
```



Scrollable Result Set

Scrollable ResultSet

- The ResultSet has been enhanced to make them scrollable and updateable since JDBC 2.0
- A ResultSet can now have one of four possible scrolling attributes:
 - Forward and backward
 - Scroll insensitive
 - Scroll sensitive

Scrollable ResultSet

- So what is the meaning of Scroll-sensitive?
 - A ResultSet that's scroll-sensitive is “live.”
 - As other users modify the data reflected in the ResultSet, the ResultSet can be changed to reflect the revised view of the data.

- So what is the meaning of Scroll-Insensitive?
 - A ResultSet that's scroll-insensitive is a static view of the data.
 - ▶ If the data in the ResultSet is changed by other clients, the ResultSet isn't updated accordingly.

Scrollable ResultSet

- To create a scrollable ResultSet we use the overloaded createStatement() and prepareStatement() methods:
-
- Statement createStatement(int resultSetType,int resultSetConcurrency)
- throws SQLException
-
- PreparedStatement prepareStatement(String SQL,int resultSetType,int resultSetConcurrency)throws SQLException
- Legal ResultSet types include:

Scrollable ResultSet Example

```
con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/raj","root","root");
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
rs=stmt.executeQuery("select * from dept2");

while(rs.next()){
    System.out.print(rs.getInt(1)+": ");
    System.out.println(rs.getString(2));
}

rs.absolute(1);

rs.updateString(2,"New Dept name");
rs.updateRow();

rs.beforeFirst();

System.out.println("After Updation");

while(rs.next()){
    System.out.print(rs.getInt(1)+": ");
    System.out.println(rs.getString(2));
}
```



Auto Generated Keys

Auto Generated Keys

- Using `getGeneratedKey()` to get auto generated key (JDBC 3.0)

```
//create table emp3(id int not null AUTO_INCREMENT primary key,name varchar(20)not null);  
//populate some records
```

```
con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/raj","root","root");  
stmt=con.createStatement(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_UPDATABLE);
```

```
//insert one row that will generate an auto increment key in key field
```

```
stmt.executeUpdate("insert into emp3(name)values('foo')",Statement.RETURN_GENERATED_KEYS);
```

```
int getKey=-1;  
rs=stmt.getGeneratedKeys();  
if(rs.next()){  
    getKey=rs.getInt(1);  
}  
System.out.println(getKey);
```



Transaction Management

Transaction Management

□ Transaction

- **logical unit of related work**
- **ACID properties**
- **Transaction is a complete task which is a combination of the smaller tasks. For the major task to be completed, the smaller tasks need to be successfully completed. If any one task fails then all the previous tasks are reverted to the original state.**

□ Example:

Transaction Management Example

- To understand transactions we will use two tables Login (id, login, password) and Student (regno, name,degree,semester).
- Insertion of student data will require us to insert data either into both the tables or in none of them in case there is an error. Login will be same as regno.

```
con.setAutoCommit(false);  
st.executeUpdate("INSERT INTO Login  
VALUES(id, login, pass)");  
st.executeUpdate("INSERT INTO Student  
VALUES(login, name, 'B.E.')");  
con.commit();  
con.close();  
}  
catch(Exception e)  
{
```

Connection.setSavepoint

- There are some situations when you have to only rollback() to some specific statement and not all statements in a transaction.
- This can be achieved by introducing some labels to the statements in the transactions. These labels are known as savepoints.

```
con.setAutoCommit(false);
stmt= con.createStatement();

stmt.execute("insert into dept(deptid, deptname)values('1221','rajat')");
Savepoint sp1=con.setSavepoint("SP1");
stmt.execute("delete from dept");
//Before Commits the transaction rollback
System.out.println("Commit called ");

con.rollback(sp1);
con.commit();
```




Connection Pooling concept

Connection Pooling concept



- The javax.sql package defines several new interfaces to support the use of connection pools:
 - ConnectionPoolDataSource
 - PoolableConnection
 - ConnectionEventListener
- The use of these interfaces is vendor specific. We will do it with tomcat during servlet-jsp session

Connection Pooling concept

- Provides a cache, or pool, of prefabricated database connections.
- When a client needs a database connection, it goes to the pool.
- When the client is done with the connection, that connection is returned to the pool.
- Pools increase performance by avoiding the cost of creating new connections for each client.
- Pools decrease resource usage by using a small number of connections to service a large number of clients.



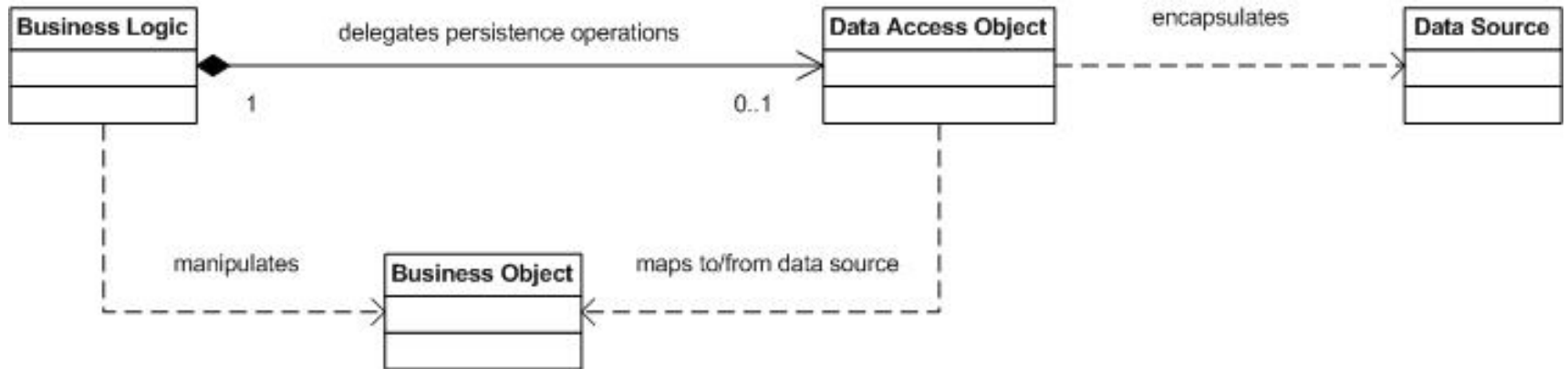
Data Access Object (DAO) Pattern

Why DOA pattern?

- Many components within an application need access to data
- Interfaces to data vary by technology and vendor
 - least common denominator option for portability may not be feasible in all cases
 - May make use of vendor extensions
- Impact of unique interfaces significant when exposed to many component and component types
 - components need more abstraction and shielding from the details of the persistent store

DAO solution

- Use a data access object (DAO) to abstract and encapsulate the data source



DAO solution

DAO Pattern Interactions



DAO Pattern Roles

- Business Logic
 - the object within the business domain that needs access to data (e.g., session bean)
 - knows when/why data is needed, but not how
- Data Access Object (DAO)
 - abstracts the access details from the business object
 - knows how data is accessed, but not when/why
- Business Object
 - an entity within the business logic
 - a data carrier of information to/from the DAO
- Data Source
 - physically stores the data (e.g., database)