JDBC

Q.What is JDBC ? Explain Types Of Drivers with Advantage and Disadvantage ?

Java Database Connectivity(JDBC) defines how a java program can communicate with a database. JDBC API has two major packages java.sql and javax.sql.   
There are 4 types of JDBC drivers available.   
1)    Type 1 Driver- the JDBC-ODBC bridge :   
The JDBC type 1 driver, also known as the JDBC-ODBC bridge is a database driver implementation that the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database.   
The driver is implemented in the sun.jdbc.odbc.JdbcOdbcDriver class .   
The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the operating system.   
Advantage :   
Almost any database for which ODBC driver is installed, can be accessed.   
Disadvantage :   
a)    Performance overhead since the calls have to go through the JDBC overhead bridge to the ODBC driver.   
b)    The ODBC driver needs to be installed on the client machine   
c)    considering the client-side software needed, this might not be suitable for applets.   
  
  
2)    Type 2 Driver - the Native-API Driver :   
The JDBC type 2 driver, also known as the Native-API driver is a database driver implementation that uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.   
The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls.   
  
A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for ORACLE, DB2 or other . Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.   
However the type 2 driver provides more functionality and performance than the type 1 driver as it does not have the overhead of the additional ODBC function calls.   
  
Advantage:   
Better performance than Type 1 since no jdbc to odbc translation is needed   
  
Disadvantage :   
a)    The vendor client library needs to be installed on the client machine.   
b)    Cannot be used in internet due the client side software needed   
c)    Not all databases give the client side library   
  
3)    Type 3 driver - the Network-Protocol Driver:   
The JDBC type 3 driver, also known as the network-protocol driver is a database driver implementation which makes use of a middle-tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.   
  
Advantages:   
a)    Since the communication between client and the middleware server is database independent, there is no need for the vendor db library on the client machine.   
b)    The Middleware Server (Can be a full fledged J2EE Application server) can provide typical middleware services like caching (connections, query results, and so on), load balancing etc.   
Disadvantages :   
a)    Requires database-specific coding to be done in the middle tier.   
b)    An extra layer added may result in a time-bottleneck   
  
  
4)    Type 4 - the Native-Protocol Driver :   
The JDBC type 4 driver, also known as the native-protocol driver is a database driver implementation that converts JDBC calls directly into the vendor-specific database protocol.   
The type 4 driver is written completely in Java and is hence platform independent. It is installed inside the Java Virtual Machine of the client. It provides better performance over the type 1 and 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 1 and 2 drivers, it does not need associated software to work..   
  
Advantages :   
a) These drivers don't translate the requests into db request to ODBC or pass it to client api for the db, nor do they need a middleware layer for request indirection. Thus the performance is considerably improved.   
b) Web application mainly used this driver.   
Disadvantage:   
At client side, a separate driver is needed for each database. ex- classes12.zip (for ORACLE)

Image upload from JSP to DataBase ?

Image upload from JSP to Data Base step by step :   
  
step 1 :   
In the JSP   
<form name="regform2" method="post" enctype="multipart/form-data">   
<input type="file" name="ImageFile" id="ImageFile" onChange="uploadImage()"/>   
</form>   
  
java script :   
function uploadImage(){   
document.regform.action ="<%=request.getContextPath()%>/dfdmin?cmd=uploadimage";   
document.regform.submit();   
}   
  
  
Step 2.   
In the servlet - add the below code.   
String rtempfile = File.createTempFile("temp","1").getParent();   
MultipartRequest multi = new MultipartRequest(request, rtempfile, 500000 \* 1024);   
File rnewfile=null;   
rnewfile = new File(CommonArt.IMAGE\_PATH+"jsp"+File.separator+"images"+File.separator+"uploadImage"+File.separator);   
if(rnewfile.exists()){   
}else{   
rnewfile.mkdirs();   
}   
  
File f = multi.getFile("ImageFile");   
System.out.println(f.getName());   
FileInputStream fin =new FileInputStream(f);   
RandomAccessFile r = new RandomAccessFile(rnewfile+File.separator+f.getName(),"rw");   
filename = f.getName();   
// FileOutputStream fos =new FileOutputStream(rnewfile);   
byte sizefile[] = new byte[5000000];   
fin.read(sizefile);   
// fos.write(sizefile);   
r.write(sizefile);   
//fos.close();   
r.close();   
fin.close();   
  
  
Step 3.   
Insert into Database   
  
InputStream is = new FileInputStream(f);   
String sql = " INSERT INTO image\_upload (IMAGE) VALUES (?) ";   
pStmt = conn.prepareStatement(sql);   
pStmt.setBinaryStream(1, is, (int)(f.length()));   
pStmt.execute();   
conn.commit();   
  
  
  
This will upload multipart file to your data base.   
  
Note : get cos.jar from oreilly website

Q. What is Rowset ? CachedRowSet,JDBCRowSet and WebRowSet ?

A RowSet object is a java bean component and extends the ResultSet interface.   
The RowSet is majorly classified into two types :   
  
1. Connected Rowset :   
The connected rowset as the name suggests in connected to the database connection object like the resultset.   
The JDBCRowSet is the example of the connected RowSet.   
  
2. Disconnected RowSet:   
The disconnected RowSet only connects to the database whenever required and after finishing the interaction they close the database connection.   
  
There are three types of RowSet :   
  
A CachedRowSet class?   
a disconnected rowset that caches its data in memory; not suitable for very large data sets, but an ideal way to provide thin Java clients.   
  
Example :   
CachedRowSet cachedRs = new CachedRowSetImpl();   
cachedRs.setUsername("scott");   
cachedRs.setPassword("tiger");   
cachedRs.setUrl("jdbc:oracle://localhost:3306/test");   
cachedRs.setCommand("select \* from employee");   
cachedRs.setPageSize(4);   
cachedRs.execute();   
while (cachedRs.nextPage()) {   
while (cachedRs.next()) {   
System.out.println(cachedRs.getString("emp\_name"));   
}   
}   
  
A JDBCRowSet class?   
a connected rowset that serves mainly as a thin wrapper around a ResultSet object to make a JDBC driver look like a JavaBeans component.   
Example:   
JdbcRowSet jdbcRs = new JdbcRowSetImpl();   
jdbcRs.setUsername("scott");   
jdbcRs.setPassword("tiger");   
jdbcRs.setUrl("jdbc:oracle://localhost:3306/test");   
jdbcRs.setCommand("select \* from employee");   
jdbcRs.execute();   
while(jdbcRs.next()) {   
System.out.println(jdbcRs.getString("emp\_name"));   
}   
  
  
A WebRowSet class?   
a connected rowset that uses the HTTP protocol internally to talk to a Java servlet that provides data access;   
used to make it possible for thin web clients to retrieve and possibly update a set of rows.

### **Q. What is Batch Updates Using Statements in JDBC ?**

Batch Updates calls to database as a chunk.   
If you want to run more than one sql statement in a single database call then you have to go for Batch Update.   
All Insert in single data base call and single transaction ( if one fail the all fail)   
  
For Example :   
public class TestDB {   
public static void main(String[] args) {   
try {   
  
/\*\* Loading the driver\*/   
  
Class.forName("com.oracle.jdbc.Driver");   
  
/\*\* Getting Connection\*/   
Connection con = DriverManager.getConnection("jdbc:oracle://localhost:3306/test","test","test");   
  
/\*\* Creating Statement\*/   
con.setAutoCommit(false); // This means after all insert then commit at last   
Statement stmt = con.createStatement();   
        stmt.addBatch("INSERT INTO EMP VALUES(1,'Ram1')");   
        stmt.addBatch("INSERT INTO EMP VALUES(2,'Ram2')");   
        stmt.addBatch("INSERT INTO EMP VALUES(3,'Ram3')");   
        stmt.addBatch("INSERT INTO EMP VALUES(4,'Ram4')");   
        int [] updateCounts= stmt.executeBatch();   
        con.commit();   
        con.setAutoCommit(true);   
        /// All Insert in single data base call and single transaction ( if one fail the all fail)   
  
stmt.close();   
con.close();   
  
} catch (Exception e) {   
  
e.printStackTrace();   
}   
     }   
}

Q.Handling Blob and CLOB data using JDBC ? Insert CLOB and Retrive CLOB , Convert into String ?

BLOB (Binary Large Objects ) and CLOB(Character large objects) are special datatypes and can hold the large chunks of data in form of objects or text.   
Blob and Clob objects persist the data of the objects into the database as a stream.   
  
Example :   
  
Java code to execute procedure   
  
public class TestDB {   
public static void main(String[] args) {   
try {   
  
/\*\* Loading the driver\*/   
  
Class.forName("com.oracle.jdbc.Driver");   
  
/\*\* Getting Connection\*/   
Connection con = DriverManager.getConnection("jdbc:oracle://localhost:3306/test","test","test");   
  
PreparedStatement pstmt = con.prepareStatement("insert into Emp(id,name,description)values(?,?,?)");   
pstmt.setInt(1,5);   
pstmt.setString(2,"Das");   
// Create a big CLOB value...AND inserting as a CLOB   
StringBuffer sb = new StringBuffer(400000);   
  
sb.append("This is the Example of CLOB ..");   
String clobValue = sb.toString();   
pstmt.setString(3, clobValue);   
int i= pstmt.executeUpdate();   
System.out.println("Done Inserted");   
pstmt.close();   
     con.close();   
  
     // Retrive CLOB values   
Connection con = DriverManager.getConnection("jdbc:oracle://localhost:3306/test","test","test");   
PreparedStatement pstmt = con.prepareStatement("select \* from Emp where id=5");   
ResultSet rs = pstmt.executeQuery();   
Reader instream = null;   
int chunkSize;   
if(rs.next()){   
String name = rs.getString("name");   
java.sql.Clob clob = result.getClob("description")   
StringBuffer sb1 = new StringBuffer();   
  
chunkSize = ((oracle.sql.CLOB)clob).getChunkSize();   
instream = clob.getCharacterStream();   
BufferedReader in = new BufferedReader(instream);   
String line = null;   
while ((line = in.readLine()) != null) {   
sb1.append(line);   
    }   
if(in != null){   
in.close();   
}   
  
String clobdata = sb1.toString(); // this is the clob data converted into string   
  
}   
  
  
} catch (Exception e) {   
  
e.printStackTrace();   
}   
     }   
}

Q. Calling Stored procedures with Callable Statement ?

Callable Statement is used to call stored procedure .   
  
Step 1:   
Write the procedure   
CREATE OR REPLACE PROCEDURE getEmpAge   
(ID IN NUMBER, AGE OUT NUMBER)   
  
IS   
Return\_AGE NUMBER;   
BEGIN   
SELECT AGE INTO Return\_AGE FROM EMP WHERE ID:=ID;   
Return\_AGE;   
END;   
  
Step 2:   
Java code to execute procedure   
  
public class TestDB {   
public static void main(String[] args) {   
try {   
  
/\*\* Loading the driver\*/   
  
Class.forName("com.oracle.jdbc.Driver");   
  
/\*\* Getting Connection\*/   
Connection con = DriverManager.getConnection("jdbc:oracle://localhost:3306/test","test","test");   
  
/\*\* Creating CallableStatement\*/   
  
CallableStatement call = con.prepareCall("call getEmpAge(?,?)");   
     call.setInt(1,24); // setting emp id=24   
call.registerOutParameter(2,Types.INTEGER);   
call.execute();   
System.out.println("Emp Age is "+call.getInt(2));   
  
  
stmt.close();   
con.close();   
  
} catch (Exception e) {   
  
e.printStackTrace();   
}   
     }   
}

Q.What is JDBC ? Explain Types Of Drivers with Advantage and Disadvantage ?

Java Database Connectivity(JDBC) defines how a java program can communicate with a database. JDBC API has two major packages java.sql and javax.sql.   
There are 4 types of JDBC drivers available.   
1)    Type 1 Driver- the JDBC-ODBC bridge :   
The JDBC type 1 driver, also known as the JDBC-ODBC bridge is a database driver implementation that the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database.   
The driver is implemented in the sun.jdbc.odbc.JdbcOdbcDriver class .   
The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the operating system.   
Advantage :   
Almost any database for which ODBC driver is installed, can be accessed.   
Disadvantage :   
a)    Performance overhead since the calls have to go through the JDBC overhead bridge to the ODBC driver.   
b)    The ODBC driver needs to be installed on the client machine   
c)    considering the client-side software needed, this might not be suitable for applets.   
  
  
2)    Type 2 Driver - the Native-API Driver :   
The JDBC type 2 driver, also known as the Native-API driver is a database driver implementation that uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.   
The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls.   
  
A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for ORACLE, DB2 or other . Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.   
However the type 2 driver provides more functionality and performance than the type 1 driver as it does not have the overhead of the additional ODBC function calls.   
  
Advantage:   
Better performance than Type 1 since no jdbc to odbc translation is needed   
  
Disadvantage :   
a)    The vendor client library needs to be installed on the client machine.   
b)    Cannot be used in internet due the client side software needed   
c)    Not all databases give the client side library   
  
3)    Type 3 driver - the Network-Protocol Driver:   
The JDBC type 3 driver, also known as the network-protocol driver is a database driver implementation which makes use of a middle-tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.   
  
Advantages:   
a)    Since the communication between client and the middleware server is database independent, there is no need for the vendor db library on the client machine.   
b)    The Middleware Server (Can be a full fledged J2EE Application server) can provide typical middleware services like caching (connections, query results, and so on), load balancing etc.   
Disadvantages :   
a)    Requires database-specific coding to be done in the middle tier.   
b)    An extra layer added may result in a time-bottleneck   
  
  
4)    Type 4 - the Native-Protocol Driver :   
The JDBC type 4 driver, also known as the native-protocol driver is a database driver implementation that converts JDBC calls directly into the vendor-specific database protocol.   
The type 4 driver is written completely in Java and is hence platform independent. It is installed inside the Java Virtual Machine of the client. It provides better performance over the type 1 and 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 1 and 2 drivers, it does not need associated software to work..   
  
Advantages :   
a) These drivers don't translate the requests into db request to ODBC or pass it to client api for the db, nor do they need a middleware layer for request indirection. Thus the performance is considerably improved.   
b) Web application mainly used this driver.   
Disadvantage:   
At client side, a separate driver is needed for each database. ex- classes12.zip (for ORACLE)

Q. How java retrive the result returns from stored procedure ?

In the Data Base :   
create procedure getEmpId (VARCHAR) returns integer '   
declare   
e\_id NUMBER;   
begin   
select emp\_id into e\_id from employee where name = $1;   
return e\_id;   
  
end;   
  
In the Java Code :   
  
CallableStatement proc =   
connection.prepareCall("{ ? = call getEmpId (?) }");   
proc.registerOutParameter(1, Types.INTEGER);   
proc.setString(2, name);   
cs.execute();   
int empId = proc.getInt(2);

### **Q.How to retrive the results in java returned by refcursor in Stored procedure ?**

In the Stored procedure :   
create procedure emp\_details () return refcursor as '   
declare   
empref refcursor;   
begin   
open empref for   
SELECT emp.name,emp.age   
FROM emp;   
return empref;   
end;   
  
In the Java Code :   
  
Connection con = null;   
CallableStatement cstmt = null;   
try   
{   
con = ConnectionPool.getConnection();   
  
con.setAutoCommit(false);   
  
// Setup the call.   
cstmt = connection.prepareCall("{ ? = call emp\_details () }");   
// you can use   
cstmt = connection.prepareCall("{call emp\_details (?) }");   
  
cstmt.registerOutParameter(1, OracleTypes.CURSOR);   
cstmt.execute();   
  
ResultSet rs = (ResultSet)cstmt.getObject(1);   
while (rs.next())   
{   
String name = rs.getString(1);   
int age = rs.getInt(2);   
}   
rs.close();   
}   
catch (SQLException e)   
{   
cstmt.close();   
con.close();   
}

Q.Reading an Oracle ARRAY from a stored procedure as an out ?

ORCALE code :   
Step 1. Create a Object in Oracle   
CREATE OR REPLACE   
TYPE test.EMP\_TYPE IS OBJECT (   
join\_date DATE,   
emp\_name VARCHAR2(200)   
)   
Step 2. Create ORACLE ARRAY   
CREATE OR REPLACE   
TYPE test.EMP\_ARRAY AS VARYING ARRAY (5000) OF EMP\_TYPE;   
Step 3.   
Create Stored procedure   
CREATE OR REPLACE PACKAGE test.EMP\_TEST AS   
PROCEDURE EMP\_RET (p\_val            IN OUT    NOCOPY EMP\_ARRAY   
}   
// So what ever you want and store the data into the Array.   
Java Code :   
OracleCallableStatement ocs = null;   
ResultSet rs = null;   
//Prepare statement and array   
ocs = (OracleCallableStatement) conn.prepareCall({?CALL EMP\_TEST. EMP\_RET(?)?);   
ArrayDescriptor ad = null;   
ad = new ArrayDescriptor(?EMP\_ARRAY?, conn);   
ARRAY emp\_aa = new ARRAY(ad, conn, null);   
ocs.setArray(1, emp\_aaa);   
ocs.registerOutParameter(1,OracleTypes.ARRAY,?EMP\_ARRAY?);   
ocs.execute();   
ARRAY array = (ARRAY) ocs.getArray(1);   
if (array != null && array.length() > 0) {   
rs = array.getResultSet();   
}   
while (rs.next()) {   
STRUCT rowStruct = (STRUCT) rs.getObject(1);   
Object[] cols = rowStruct.getAttributes();   
System.out.println(cols[1]);//for join\_date   
System.out.println(cols[2]); // for emp\_name   
}

What is Metadata and why should I use it?

Metadata (?data about data?) is information about one of two things: Database information (java.sql.DatabaseMetaData), or Information about a specific ResultSet (java.sql.ResultSetMetaData). Use DatabaseMetaData to find information about your database, such as its capabilities and structure. Use ResultSetMetaData to find information about the results of an SQL query, such as size and types of columns

How Class.forName() load the Driver and DriverManager.getConnection() return connection?

These are the steps happing inside   
Step 1.   
Class.forName("com.mysql.jdbc.Driver") load the Driver.   
Step 2.   
In the com.mysql.jdbc.Driver class there is static bock .   
That will execute because of static bock.   
static   
{   
try   
{   
DriverManager.registerDriver(new Driver());   
}   
catch(SQLException E)   
{   
throw new RuntimeException("Can't register driver!");   
}   
}   
This static block call DriverManager.registerDriver(new Driver());   
  
  
Step 3.   
Inside DriverManager.registerDriver() method.   
public static synchronized void registerDriver(java.sql.Driver driver)   
    throws SQLException {   
       
    DriverInfo di = new DriverInfo();   
    di.driver = driver;   
    di.driverClass = driver.getClass();   
    di.driverClassName = di.driverClass.getName();   
    drivers.addElement(di);   
    println("registerDriver: " + di);   
}   
  
DriverManager class create a Vector name "drivers" and add the Driver class to the vector.   
Now In the Vector we have com.mysql.jdbc.Driver object.   
  
Step 4.   
To get connection we can   
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/testDB","username","password");   
In this method , it search for the Driver in the vector , if available then connect and return connection.   
for (int i = 0; i < drivers.size(); i++) {   
     DriverInfo di = (DriverInfo)drivers.elementAt(i);   
// if "jdbc:mysql" keywork which is input , match with the driver in the vector then {   
Connection result = di.driver.connect(url, info);   
  
}   
}   
Return connection;   
This class getting jdbc:mysql://localhost:3306/testDB as input . This class check for the driver based on "jdbc:mysql" in this case and connect the driver.

What will Class.forName do while loading drivers?

These are the steps happing inside   
Step 1.   
Class.forName("com.mysql.jdbc.Driver") load the Driver.   
Step 2.   
In the com.mysql.jdbc.Driver class there is static bock . That will execute because of static bock.   
static   
{   
try   
{   
DriverManager.registerDriver(new Driver());   
}   
catch(SQLException E)   
{   
throw new RuntimeException("Can't register driver!");   
}   
}   
This static block call DriverManager.registerDriver(new Driver());   
  
Step 3.   
Inside DriverManager.registerDriver() method.   
public static synchronized void registerDriver(java.sql.Driver driver)   
    throws SQLException {   
       
    DriverInfo di = new DriverInfo();   
    di.driver = driver;   
    di.driverClass = driver.getClass();   
    di.driverClassName = di.driverClass.getName();   
    drivers.addElement(di);   
    println("registerDriver: " + di);   
}   
  
DriverManager class create a Vector name drivers and add the Driver class to the vector.   
Now In the Vector we have com.mysql.jdbc.Driver object.   
  
Step 4.   
To get connection we can   
Connection con = DriverManager.getConnection(?jdbc:mysql://localhost:3306/testDB?,?username?,?password?);   
In this method , it search the Driver in the vector , if available then connect and return connection.   
for (int i = 0; i < drivers.size(); i++) {   
     DriverInfo di = (DriverInfo)drivers.elementAt(i);   
// if ?jdbc:mysql? keywork withic in input , match with the driver in the vector {   
Connection result = di.driver.connect(url, info);   
  
}   
}   
Return connection;   
This class getting jdbc:mysql://localhost:3306/testDB as input . This class check for the driver based on ?jdbc:mysql? in this case and connect the driver

### **How to Determining If a Result Set Is Scrollable?**

resultSet.getType() will return you the type.   
try {   
// Get type of the result set   
int type = resultSet.getType();   
  
if (type == ResultSet.TYPE\_SCROLL\_INSENSITIVE   
|| type == ResultSet.TYPE\_SCROLL\_SENSITIVE) {   
// Result set is scrollable   
} else {   
// Result set is not scrollable   
}   
} catch (SQLException e) {   
}

How Getting the Number of Rows in a Table Using a Scrollable Result Set?

This example gets the number of rows in a scrollable result set by moving the cursor to the last row of the result set and then calling ResultSet.getRow().   
try {   
// Create a scrollable result set   
Statement stmt = connection.createStatement(   
ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);   
ResultSet resultSet = stmt.executeQuery("SELECT \* FROM my\_table");   
  
// Move to the end of the result set   
resultSet.last();   
  
// Get the row number of the last row which is also the row count   
int rowCount = resultSet.getRow();   
} catch (SQLException e) {   
}

What is Scrollable Result Set ?

scrollable result set allows the cursor to be moved to any row in the result set. Back and forward.   
try {   
// Create an insensitive scrollable result set   
Statement stmt = connection.createStatement(   
ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);   
  
// Create a sensitive scrollable result set   
stmt = connection.createStatement(   
ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY);   
} catch (SQLException e) {   
}

What does setAutoCommit do?

When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed. The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode:   
  
con.setAutoCommit(false);   
  
Once auto-commit mode is disabled, no SQL statements will be committed until you call the method commit explicitly.   
  
con.setAutoCommit(false);   
PreparedStatement updateSales =   
    con.prepareStatement( "UPDATE COFFEES SET SALES = ? WHERE COF\_NAME LIKE ?");   
updateSales.setInt(1, 50); updateSales.setString(2, "Colombian");   
updateSales.executeUpdate();   
PreparedStatement updateTotal =   
    con.prepareStatement("UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF\_NAME LIKE ?");   
updateTotal.setInt(1, 50);   
updateTotal.setString(2, "Colombian");   
updateTotal.executeUpdate();   
con.commit();   
con.setAutoCommit(true);

### **What is the difference between TYPE\_SCROLL\_INSENSITIVE , and TYPE\_SCROLL\_SENSITIVE?**

You will get a scrollable ResultSet object if you specify one of these ResultSet constants.The difference between the two has to do with whether a result set reflects changes that are made to it while it is open and whether certain methods can be called to detect these changes. Generally speaking, a result set that is TYPE\_SCROLL\_INSENSITIVE does not reflect changes made while it is still open and one that is TYPE\_SCROLL\_SENSITIVE does. All three types of result sets will make changes visible if they are closed and then reopened:   
  
Statement stmt =   
    con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);   
ResultSet srs =   
    stmt.executeQuery("SELECT COF\_NAME, PRICE FROM COFFEES");   
srs.afterLast();   
while (srs.previous())   
{   
    String name = srs.getString("COF\_NAME");   
    float price = srs.getFloat("PRICE");   
    System.out.println(name + " " + price);   
}

How to Make Updates to Updatable Result Sets?

Another new feature in the JDBC 2.0 API is the ability to update rows in a result set using methods in the Java programming language rather than having to send an SQL command. But before you can take advantage of this capability, you need to create a ResultSet object that is updatable. In order to do this, you supply the ResultSet constant CONCUR\_UPDATABLE to the createStatement method.   
  
Connection con =   
    DriverManager.getConnection("jdbc:mySubprotocol:mySubName");   
Statement stmt =   
    con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);   
ResultSet uprs =   
    stmt.executeQuery("SELECT NAME, PRICE FROM COFFEES");

How can you move the cursor in scrollable result sets?

One of the new features in the JDBC 2.0 API is the ability to move a result set?s cursor backward as well as forward. There are also methods that let you move the cursor to a particular row and check the position of the cursor.   
  
Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY);   
ResultSet srs = stmt.executeQuery(?SELECT COF\_NAME, PRICE FROM COFFEES?);   
  
The first argument is one of three constants added to the ResultSet API to indicate the type of a ResultSet object: TYPE\_FORWARD\_ONLY, TYPE\_SCROLL\_INSENSITIVE , and TYPE\_SCROLL\_SENSITIVE. The second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable: CONCUR\_READ\_ONLY and CONCUR\_UPDATABLE. The point to remember here is that if you specify a type, you must also specify whether it is read-only or updatable. Also, you must specify the type first, and because both parameters are of type int , the compiler will not complain if you switch the order. Specifying the constant TYPE\_FORWARD\_ONLY creates a nonscrollable result set, that is, one in which the cursor moves only forward. If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE\_FORWARD\_ONLY and CONCUR\_READ\_ONLY.

What are the different types of Statements with Example?

### **Regular statement (use createStatement method), prepared statement (use prepareStatement method) and callable statement (use prepareCall) statement : Statement stmt = con.createStatement(); ResultSet rs = stmt.executeQuery("SELECT NAME, PRICE FROM COFFEES"); String s = rs.getString("NAME"); prepareStatement : precompiled statement String sql = " select FIRST\_NAME, LAST\_NAME, USER\_DATA where user\_id = ? "; prepareStatement pStmt = conn.prepareStatement(sql); pStmt.setLong(1, user.getUserId()); rs = pStmt.executeQuery(); String s = rs.getString("FIRST\_NAME"); callable statement : A CallableStatement object contains a call to a stored procedure.     CallableStatement cs = con.prepareCall("{call SHOW\_SUPPLIERS}");     ResultSet rs = cs.executeQuery(); How can you retrieve data from the ResultSet?**

conn = getConnection();   
            
         String sql = " select FIRST\_NAME from USER\_DATA where user\_id = ? ";   
         pStmt = conn.prepareStatement(sql);   
         pStmt.setLong(1, user.getUserId());   
while(rs.next()){   
String fname=rs.getString("FIRST\_NAME");   
}

How can you create JDBC statements and what are they?

A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate. It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object   
  
  
  
Example:   
Statement stmt = con.createStatement("select \* from emp");   
rs = stmt.executeQuery();

### **How can you make the connection?**

While making a JDBC connection we go through the following steps :   
  
  
  
Step 1 : Register the database driver by using :   
  
Class.forName(\" driver classs for that specific database\" );   
  
Class.forName("com.mysql.jdbc.Driver");   
  
  
  
Step 2 : Now create a database connection using :   
  
Connection con = DriverManager.getConnection(url,username,password);   
  
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/art1",art1,art1);   
  
  
  
Step 3: Now Create a query using :   
  
Statement stmt = Connection.Statement(\"select EMP\_NAME from EMP\");   
  
  
  
Step 4 : Exceute the query :   
  
ResultSet rs = stmt.exceuteQuery();   
  
while(rs.next()){   
  
  
  
System.out.println(rs.getString(EMP\_NAME));   
  
  
  
}

What will Class.forName do while loading drivers?

It is used to create an instance of a driver and register it with the   
DriverManager. When you have loaded a driver, it is available for making a connection with a DBMS

How can you load the drivers?

Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:   
  
Class.forName(?sun.jdbc.odbc.JdbcOdbcDriver?);   
  
Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ, you would load the driver with the following line of code:   
  
Class.forName(?jdbc.DriverXYZ?);

What is PreparedStatement ?

PreparedStatement is precompiled statement. Fist time it compile the SQL query and next call it only pass the parameter value and execute, in the same connection.   
  
PreparedStatement is not compiling query every time.   
PreparedStatement is better for CLOB and BLOB object.   
Example:   
PreparedStatement pstmt = con.prepareStatement(   
    "UPDATE table4 SET name = ? WHERE eid = ?");   
  
pstmt.setString(1,"das");   
pstmt.setInt(2,5);   
pstmt.executeUpdate();   
  
The object pstmt now contains the statement "UPDATE table4 SET m = ? WHERE x = ?", which has already been sent to the DBMS and been prepared for execution.

what is rowset? and how does it differ from resultset ?

Both rowset and resultset is contains result data from executed query.   
But after connection close resultset data will be lost. But in case of rowset you can get the data after connection close also.

How to find total column from a resultset?

int columnCnt = java.sql.ResultSetMetaData.getColumnCount();

what is rowset? and how does it differ from resultset ?

Both rowset and resultset is contains result data from executed query.   
But after connection close resultset data will be lost. But in case of rowset you can get the data after connection close also.

What is the fastest type of JDBC driver?

Type 4 (JDBC Net pure Java Driver) is the fastest JDBC driver. Type 1 and Type 3 drivers will be slower than Type 2 drivers (the database calls are make at least three translations versus two), and Type 4 drivers are the fastest (only one translation).

Q.What are collection pools? What are the advantages?

A connection pool create database connections on server start up and maintained in memory (vector). so that the connections may be reused.   
If you need a connection the connection pool give you the connection from connection pool. So it perform better because no need to create connection to database. Create connection to Database is time consuming.   
Example:   
Genererally most of the application server has their cown connection pool..like weblogic, websphare , ATG, JBOSS.   
  
just we need to set in properties file or xml file   
min=10;   
max=20;

### **Q. What is Dirty read? Q.What are different types of Transaction Isolation Levels?**

The isolation level describes the degree to which the data being updated is visible to other transactions. This is important when two transactions are trying to read the same row of a table.   
Imagine two transactions: A and B. Here three types of inconsistencies can occur:   
  
Dirty-read: A has changed a row, but has not committed the changes. B reads the uncommitted data but his view of the data may be wrong if A rolls back his changes and updates his own changes to the database.   
  
Non-repeatable read: B performs a read, but A modifies or deletes that data later. If B reads the same row again, he will get different data.   
  
Phantoms: A does a query on a set of rows to perform an operation. B modifies the table such that a query of A would have given a different result. The table may be inconsistent.   
  
TRANSACTION\_READ\_UNCOMMITTED : DIRTY READS, NON-REPEATABLE READ AND PHANTOMS CAN OCCUR.   
  
TRANSACTION\_READ\_COMMITTED : DIRTY READS ARE PREVENTED, NON-REPEATABLE READ AND PHANTOMS CAN OCCUR.   
TRANSACTION\_REPEATABLE\_READ : DIRTY READS , NON-REPEATABLE READ ARE PREVENTED AND PHANTOMS CAN OCCUR.   
TRANSACTION\_SERIALIZABLE : DIRTY READS, NON-REPEATABLE READ AND PHANTOMS ARE PREVENTED.

Q.Difference between Statement , PreparedStatement and CallableStatement ?

Statement : Statement every time compile the SQL and Execute.   
Example: String sql="select \* from emp where emp\_id = 1";   
Statement stmt = conn.createStatement(sql);   
  
PreparedStatement :   
If we are using PreparedStatement the execution time will be less. First time RDBMS comiple the SQL and PreparedStatement is executed then other calls doesn't compile the SQL only execute the SQL within the connection live.   
  
you must use a PreparedStatement object if you want to use large objects like BLOBs or CLOBs.   
  
PreparedStatement is its support for batching   
  
Example :   
String sql="select \* from emp where emp\_id = ?";   
PreparedStatement pStmt = conn.prepareStatement(sql);   
pStmt.setLong(1, profile.getUserId());   
  
CallableStatement :CallableStatement is for call to a stored procedure.   
Example :   
CallableStatement cs =   
con.prepareCall("{call SHOW\_CUSTOMER}");   
ResultSet rs = cs.executeQuery();

Q.What is the advantage of using PreparedStatement?

If we are using PreparedStatement the execution time will be less. First time RDBMS comiple the SQL and PreparedStatement is executed then other calls doesn't compile the SQL only execute the SQL within the connection live.   
  
you must use a PreparedStatement object if you want to use large objects like BLOBs or CLOBs.   
  
PreparedStatement is its support for batching.

Q.What is Single-Phase Commit ?

If only a single resource (database) is enlisted in the transaction, you can use single-phase commit.   
Example:   
conn.setAutoCommit(false);   
//do whatever   
conn.commit();

Q.What is two-phase commit?

When a transaction involves multiple distributed resources, for example, a database server on each of two different network hosts, the commit process is somewhat complex because the transaction includes operations that span two distinct software systems, each with its own resource manager, log records, and so on. (In this case, the distributed resources are the database servers.)   
  
Two-phase commit is a transaction protocol designed for the complications that arise with distributed resource managers. With a two-phase commit protocol, the distributed transaction manager employs a coordinator to manage the individual resource managers.   
  
Configuring Two-Phase Commit Engine:   
When a global transaction involves multiple databases, the changes to these resources must all be committed or rolled back at the same time. That is, when the transaction ends, the transaction manager contacts a coordinator--also known as a two-phase commit engine--to either commit or roll back all changes to all included databases. The two-phase commit engine is an Oracle9i database that is configured with the following:   
  
Fully-qualified database links from itself to each of the databases involved in the transaction. When the transaction ends, the two-phase commit engine communicates with the included databases over their fully-qualified database links.   
  
A user that is designated to create sessions to each database involved and is given the responsibility of performing the commit or rollback. The user that performs the communication must be created on all involved databases and be given the appropriate privileges.

### **Q. How JDBC work with REF CURSOR returned by stored procedure and retrive results?**

REF CUSROR is retrieved by the JDBC program   
into a ResultSet.   
Example code : This will help you.   
Using regular CallableStatement :   
CallableStatement cstmt = null;   
ResultSet rset = null;   
cstmt = con.prepareCall(   
"{? = call ref\_cursor\_package.get\_ref\_cursor(?)}"   
);   
cstmt.registerOutParameter(1, OracleTypes.CURSOR);   
cstmt.setInt(2, 104);   
cstmt.execute();   
  
rset = (ResultSet) cstmt.getObject(1);   
  
while (rset.next()) {   
System.out.println(   
" - " +   
rset.getString(2) + " (" + rset.getInt(1) + "), " +   
rset.getString(3)   
);   
}   
  
  
Using OracleCallableStatement and OracleResultSet classes:   
  
OracleCallableStatement oraCallStmt = null;   
OracleResultSet deptResultSet = null;   
oraCallStmt = (OracleCallableStatement) con.prepareCall(   
"{? = call ref\_cursor\_package.get\_dept\_ref\_cursor(?)}"   
);   
oraCallStmt.registerOutParameter(1, OracleTypes.CURSOR);   
oraCallStmt.setInt(2, 104);   
oraCallStmt.execute();   
  
deptResultSet = (OracleResultSet) oraCallStmt.getCursor(1);   
  
while (deptResultSet.next()) {   
System.out.println(   
" - " +   
deptResultSet.getString(2) + " (" + deptResultSet.getInt(1) + "), " +   
deptResultSet.getString(3)   
);   
}

### **Q. How do i get result return from stored procedure in JDBC?**

CallableStatement cstmt = con.prepareCall("{ ? = call GetSeqNexVal(?) }");   
                 cstmt.registerOutParameter(1, java.sql.Types.INTEGER);   
cstmt.setString(2,seq);   
cstmt.execute();   
id = cstmt.getInt(1);   
System.out.println("Result Return from stored procedure"+id);

Q.How do you call a Stored Procedure from JDBC?

CallableStatement object is used to call stored procedure .   
    CallableStatement cs =   
        con.prepareCall("{call SHOW\_CUSTOMER}");   
    ResultSet rs = cs.executeQuery();

Q.How do you handle your own transaction ?

Connection Object has a method called setAutocommit(Boolean istrue)   
- Default is true.   
Set the Parameter to false , and begin your transaction   
Example:   
  
Connection conn = getConnection();   
conn.setAutocommit(false);   
//other code   
conn.commit();   
conn.close();

Q.What are the steps in the JDBC connection?

While making a JDBC connection we go through the following steps :   
  
Step 1 : Register the database driver by using :   
Class.forName(\" driver classs for that specific database\" );   
Class.forName("com.mysql.jdbc.Driver");   
  
Step 2 : Now create a database connection using :   
Connection con = DriverManager.getConnection(url,username,password);   
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/art1",art1,art1);   
  
Step 3: Now Create a query using :   
Statement stmt = Connection.Statement(\"select EMP\_NAME from EMP\");   
  
Step 4 : Exceute the query :   
ResultSet rs = stmt.exceuteQuery();   
while(rs.next()){   
  
System.out.println(rs.getString(EMP\_NAME));   
  
}