Unit - III

JDBC API

Introduction to JDBC API - System requirements - Types of JDBC drivers, A brief overview of Java DB, Creating a database table - Oracle database, Connecting to a database, Setting the auto-Commit mode, Committing and Rolleback transactions, Transaction isolation level, JDBC types to Java types mapping, Knowing about the Database, Executing SQL statements, Processing Result sets making changes to a Result set, Handling multiple results from a statement. (Application programming interface)

Introduction to JDBC API:- (Java Database Connectivity)

The JDBC API provides data base independent interface to interact with any tabular data source most of the time it is used to interact with RDBMS (Relational data Base management System). Now we can also interact with any tabular data sources such as Excel spread sheets, oracle, mySQL etc.

In JDBC API to process the data Java application may sent a query to the data base, processing the data and update the data. it will use a standard sql syntaxes.

The main purpose of data base is to manage the business data. To manage this the data bases are providing to the following things.

1) A standard SQL syntax.
2) An Extension of standard SQL Syntax which is called Proprietary SQL syntax.
3) A proprietary programming languages.

For Example, Oracle Corporation is using PL/SQL as a programming language to Execute Procedures, functions and triggers. And microsoft SQL Servers using TSQL (Transaction SQL) as a programming language to perform stored procedures functions and triggers.

Here Every data base as its own Syntaxes in order to process the data. So that these Syntaxes and logics are the database dependent, but the JDBC API provides a Common Syntaxes and logics for different

data bases. it simply use a standard SQL syntax.

using JDBC API we can hides the implementation differences for different data bases. it has more classes and interfaces to perform data base activities a collection of implementation classes and interfaces which are supplied by data base vendors to interact with data base is called "JDBC drivers."

Some JDBC drivers requires software installation at client machine but some JDBC drivers doesn't requires any installation each and every data base has its own JDBC driver the Java application use these drivers to connect with a particular data base.

System requirements:-

The JDBC API has different drivers to connect with a database. some JDBC drivers doesn't requires any software installations, so that we have to provide the class CLASSPATH for supported library JAR (or) JIP files. Some JDBC drivers require Software installation. so that we have to instal software at client machine. If you want to connect with any data bases such as mysql, oracle, JavaDB, DB_2 and sybase database etc. Then we have to use the respective drivers or we have to install the respective drivers without driver we cannot connect to a particular data base

Types of JDBC drivers:-

we have the following types of JDBC drivers those are

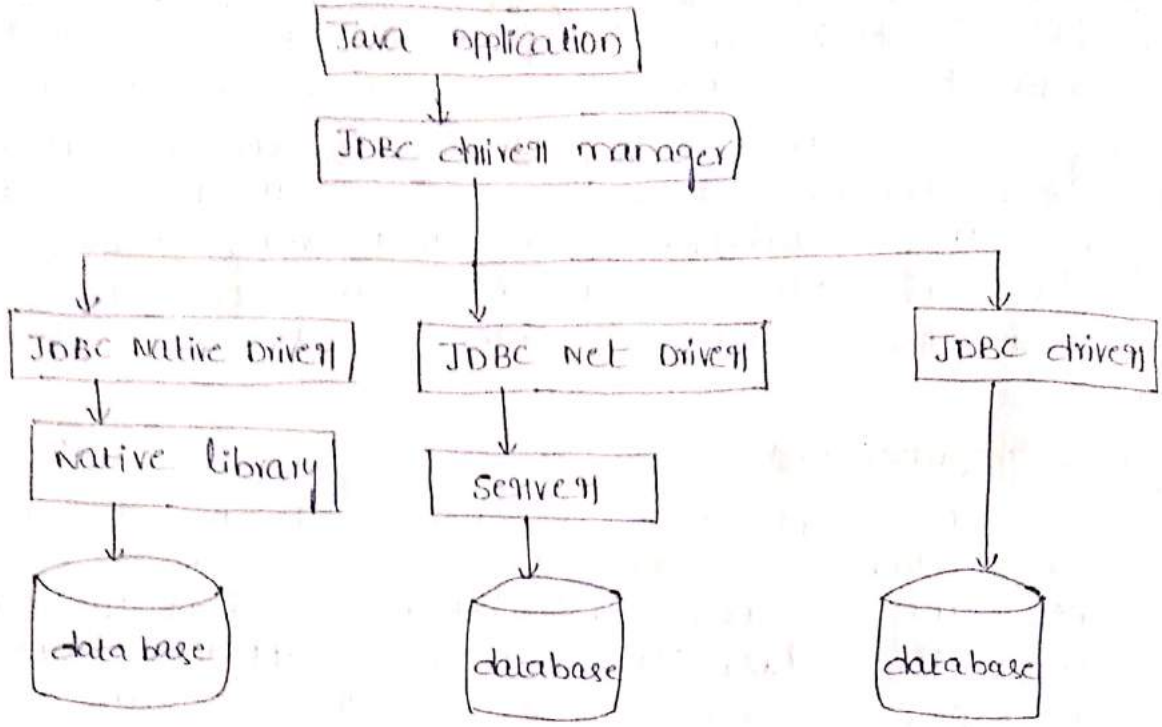
1) JDBC - ODBC Bridge driver (Type 1)
2) JDBC Native API driver (Type 2) (or) Native protocol driver
3) JDBC Net driver (Type 3) (or) Pure Java drivers)
4) JDBC driver (Type 4) (or) (100% pure Java Driver)

Generally, we have the above four types of drivers before Java 8. But after Java 8 we have only three types of drivers and they have removed JDBC - ODBC bridge driver. we have the following three types of drivers

1) JDBC Native API driver

2) JDBC Net drivers
3) JDBC drivers

we have the following architecture to demonstrate the JDBC drivers.



1) JDBC Native API driver

The JDBC Native API driver is a database dependent driver and platform dependent driver. It uses the DBMS specific library calls to perform various data base activities. it translates JDBC calls into DBMS specific Native library calls and these libraries are communicate to the data base. we must install the DBMS specific client software to use this type of driver that's why it is platform dependent.

2) JDBC Net drivers:-

It was written in pure java it requires a server to interact with the data base it translate JDBC calls into network protocols and these network protocols are translated into DBMS specific calls before translate the network protocols are passes to the server. A server will take the whole responsibility to perform various activities. The client machine cannot know about the technologies which server is using. so that it is a database independent.

3) JDBC drivers:-

It was written in 100% pure java programming

it is also called as direct to data base pure java drivers. it doesn't required any library files and middle ware servers. because it's directly connected to the data base it is most suitable for real time applications and also it is suitable for Applets.

Note:- The type 4 drivers is used whenever we are developing a real time applications and whenever we are using single data base such as either oracle or mysql etc.

* Type 3 drivers is use whenever we have multiple data bases which are maintained by a single server.

* Type 2 drivers is use whenever the Type 3 or Type 4 are not available.

A brief overview of Java DB:-

JavaDB is a Java data base generally it is used whenever there is no any data base installation because the javaDB will be installed whenever JDK will be installed. we can find a javaDB in the following link.

C:\> Programfiles\Java\JDK1.8\db

In db subdirectory we have sub directories those are lib and bin. In bin sub directory we can find different commands to start our javaDB. for example startNetworkServer.bat (batch file) and ij.bat etc.

The lib subdirectory has supported library files for example, derby.jar, derbyNet.jar. It was developed by Apachi derby foundation. we have the following supported library files.

Library	description
derby.jar –	It is used to connect JavaDB we have to set the 'Derby.jar' as classpath of our JavaDB
derbytools.jar –	It is used to configure different JavaDB tools. for Example ij (interactive javaDB)
derbyrun.jar –	It is used to configure classpath and path of our javaDB and it is also used to run of the other derby.jar files.

derbynet.jar - It is used to connect javaDB in server mode. So, that we can create the client and server environment.

derbyclient.jar - It is used by the client application in order to connect with javaDB server.

Running JavaDB Server:-

We can run JavaDB server in the following two modes

1) Embedded mode
2) Server mode

* In Embedded mode both application and javaDB in one place and there is no any client and server interaction.

* In server mode the javaDB in one machine and the client application in another machine. So that we have client and server Interaction.

* Before Connecting to the JavaDB first of all we have to set the Classpath and the path in our system. generally the Classpath has the derby home directory And path has the derby bin directory.

Step1:- Setting Classpath and Path.

Classpath:- C:\> Programfiles\Java\JDK1.8\db\lib\derby.jar;
C:\> Programfiles\Java\JDK1.8\db\lib\derbytools.jar;

Path:- C:\> Programfiles\Java\JDK1.8\bin;
C:\> Programfiles\Java\JDK1.8\db\bin;

The above classpath and path will be set to both users and system environment variables

Step2: Now we have to create our own directory in any drive.

Ex:- F:\>cse\

Step3: Now we have to open Command prompt and set a current working directory path.

Step4:- Now we have to type StartNetworkServer Command

Ex:- F:\>cse\ startNetworkServer.

It will start the javaDB server in Embedded mode in the default port no is 1527.

number

If you want to start JavaDB in server mode. we can use the following command.

-h 192.168.1.1 -p 1527
↓ host ↓ port

step 5: Now we have to minimize the current working command prompt because we are running the server mode.

step6: Then we have to open one more command prompt and set current working directory path.

step7: Type ij in second command prompt where ij means interactive JavaDB, it is used to connect with JavaDB and as well as to run SQL commands (queries)

```
f:|>cse|ij
ij>
```

step8: we have to use the following statement to create a new derby data base in and current working directory here we have to use Connect Command in order to Connect our newly created derby data base

```
ij> connect 'jdbc:derby:CSEDB;create=true;user=CSE;
                ↓ protocol  ↓ sub protocol  ↓ new database name  ↓ property  password=cse';
                                                                  ↓ authentication (optional)
```

where JDBC is a protocol it is common for every data base and derby is a sub protocol the sub-protocol will be different from one data base to other data bases and the remaining part is data base details.

After using connect command we may find our new data base name in our current working directory. Now we can create a table using create query we can insert the data using insert query. We can select the data using select query. These queries are same as SQL queries.

If you want to disconnect from the data base, we have to use the

```
ij> disconnect;
```

If you want to exit from the interactive JavaDB we have to use ij> exit;

```
f:|>cse|
```

The above steps are used to connect JavaDB

server in embedded mode using command prompt.
If you want to connect JavaDB server in server mode then the connection url is as follows

'jdbc : derby : // 192.168.1.1 : 501 / CSEDB; create = true; user = CSE; password = CSE';

Note: If there is no any network connection then it will take hostname has local host and its default port number is 1527.

localhost : 1527

we can stop the networkserver using stopNetworkServer command.

using NetBeans IDE:-

step1:- open NetBeans IDE

step2: select windows Menu. In windows Menu we have services MenuItem and we have to open services or we can open services by using a short cut key Ctrl + 5

step3: In services Tab we have a database Node.

step4: we have to open a database Node then it will displays the databases in these databases we can find JavaDB

* Select JavaDB and use mouse right click button then it will display start option. If we click start option then the javaDB server will be start.

step5: Select javaDB and use mouse right click button then you find create "database option" by clicking on create data base option. it will open a small window which contains data base name, data base username and database password.

step6:- Then we have to press ok button immediately it will create a new JavaDB database.

step7: we can find our new database by double click on JavaDB Node.

step8:- select our newly created database and use mouse click Right button on it then it will show connect option.

step9: By selecting connect option simply it will connect to data base and it will create a connection url as a new node. The connection url as follows:

Jdbc : derby : // localhost : 1527 / CSEDB [user on password]

Now, we have to select a connection until node by double click on it. we can find out user name
* we can open an Executable SQL command, by double clicking on connection URL. using prompt
* In Executable Command prompt we can Execute in SQL queries in order to create, insert, update and selecting tables
* we can also change the data base location by choosing a property of the javaDB.

Creating a Table - Oracle data base:

we can create the table using create query every table has the following fields,

column-name datatype length value comment/constrain

for Example, if we have a student table then the attributes of student table (or) are sid, sname, semail, sphone.

the schema of student table as follows

Column-name	datatype	length	value	Comment
student-id	int		Not Null	primary key
student-name	string	50	Null/Not Null	
student-email	string	50	Null/Not Null	
student-phone	string	10	Null/Not Null	

* In Oracle data base we can create the student table as follows

```
Create table student
(
  sid int not Null,
  sname varchar(50),
  semail varchar(50),
  sphone varchar(10),
  primary-key (sid)
);
```

Connecting to a database:

* we can connect to a database by using following steps:

step1 :- Obtain JDBC drivers and add it to the class path Environment variable in our System.

step2 :- Register JDBC drivers using driver manager class

step3: Construct Connection URL

step4:- use getConnection() to establish a connection to the database

obtain JDBC driver and setting class path:-

* If we want to connect to a particular data base first of all we have to get JDBC supported jar files. The JavaDB data base we can find these supported files in db/lib directory but in oracle data base we have to either downloading it or we can copy the path of OJDBC7.jar file or OJDBC14.jar.

we have to set a classpath environment variables by setting the entire path of the supported jar files.

Step2: Register for JDBC driver using driver manager class

we can register JDBC drivers by using Java.SQL.drivermanager class. we have the following steps in order to register a JDBC driver. A JDBC driver is a class which provides the supported functionalities to get connected with data base.

step1: By using class.forName() derby in embedded mode

Ex:- Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
Class.forName("org.apache.derby.jdbc.clientDriver");
derby in server mode

we have two jdbc drivers for JavaDB data base, embedded driver is for derby data base in embedded mode, the client driver is for derby database in server mode.

for oracle data base we have the following driver
oracle.jdbc.OracleDriver;

Ex:- Class.forName("oracle.jdbc.OracleDriver");

Step2: By creating an (or) instance for JDBC driver to load JDBC Driver in Jvm machine

```
Driver driver1 = new org.apache.derby.jdbc.EmbeddedDriver();
Driver driver2 = new org.apache.derby.jdbc.clientDriver();
Driver driver3 = new oracle.jdbc.OracleDriver();
```

step3: Now we have to pass these jdbc driver instances to the register driver method which is in DriverManager class

DriverManager.registerDriver(driver1);
DriverManager.registerDriver(driver2); } JavaDB
DriverManager.registerDriver(driver3); → oracle (Db)

step4: By setting "jdbc.drivers" system property we can set this property by using setProperty() which is in system class

Ex: String driver1 = "org.apache.derby.jdbc.EmbeddedDriver";
System.setProperty("jdbc.driver", driver1); JavaDB ←
String driver2 = "oracle.jdbc.oracledriver";
System.setProperty("jdbc.driver", driver2); } oracle

Construct Connection URL:-

* we can construct Connection URL based on following syntax

<protocol>:<sub-protocol>:<data-sourcedetails>.

where protocol is jdbc it is common for every database. where sub-protocol is the database names If you are using javaDB the subprotocol is derby If we are using oracle database the subprotocol is oracle:thin. The data source details are different from one data base to other data base. generally it has host name port number data base name or service Id username and password.

In JavaDB the Connection URL as follows:

"jdbc:derby://localhost:1527/csedb;user=cse;password=cse"

In oracle database the Connection URL has follows:

"jdbc:oracle:thin:cse/cse@localhost:1521/XE
user password portno SID(serviceID)

Establish a Connection using getConnection():-

we have the following syntaxes in order to use getConnection(). it will return a Connection type it is used to establish a connection with data base.

Syn:- static Connection getConnection(String url);
static Connection getConnection(String url, property info);
static Connection getConnection(String url, String user, String password);

where String url's refers the database Connection

URL where property refers a property of database for Example, Create = true.

where user refers the data base user where password refers the data base password.

* we can call getConnection() using DriverManager class and it will return a Connection type object. where Connection is a class.

This method always throws SQL Exception

Ex:- Connection Conn = DriverManager.getConnection("jdbc:derby://localhost:1527/cseDB;user=cse;password=cse");

Connection Conn = DriverManager.getConnection("jdbc:oracle:thin:cse/cse@localhost:1521/xe");

(or)

Connection Conn = DriverManager.getConnection("jdbc:derby://localhost:1527/cseDB","cse","cse");

Connection Conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/xe","cse","cse");

The following sample code is used to connect with oracle database.

```
try
{
 Class.forName("oracle.jdbc.OracleDriver");
Connection Con = DriverManager.getConnection("jdbc:oracle:thin:cse/cse@localhost:1521/xe");
System.out.println("The Connection is established");
/* perform data base activities */
}
Catch(SQLException e)
{
System.out.println(e);
}
```

Setting the Auto-Commit mode:

By default every Connection has Auto-Commit mode this mode is always true for every Connection object. we can also disable Auto-Commit mode. for a connection in java we have setAutoCommit(). It has only one parameter that is boolean. If the

parameter value is true then the Auto-Commit mode is enable. If the parameter value is false then the Auto-Commit mode is disable.

whenever this mode is enable then all modifications or deletions can be effected on our database automatically. This method will be called after establishment of jdbc connection.

```
Connection Con = DriverManager.getConnection("jdbc:
                 Oracle:thin@localhost:1521|xe","cse","cse");
Con.setAutoCommit(false);
```

Commiting and Rollback transactions:-

We can disable Auto-Commit mode using setAutoCommit by passing parameter as false. If it was disable then there is no effect on our database, even if there is any modification (or) deletion. If we want to commit your changes we have to call Commit(). If we want to Roll back your transaction we have to call Rollback() so, that these two methods are called inorder to Commit and Rollback our transaction. when ever the AutoCommit mode is disable.

we have the following sample code to show Commit() and Rollback()

Ex:-
```
// Execute Queries
 - - - -
if Transaction is Success
then
   Con.Commit();
else
   Con.Rollback();
elif
   Con.Clos();
 - - - -
```

Transaction Isolation level:- In multiuser environment we have to consider the following two factors.

1. Data Concurrency
2. Data Consistency

* The data Concurrency means ability to maintain a database by the multiusers concurrently.
* The data Consistency refers accuracy of the

data that is maintain whenever multiusers are accessing concurrently.

* In database we have to maintain data consistency by using logs and by Isolating one transaction from another transaction. A transactior is isolated from other based on desired level of isolation. Let us consider the following three situations where data consistency may be compromize in multiuser environment.

a) Dirty Read

b) Non-Repeatable Read

c) phantom Read

a) Dirty Read:- In Dirty Read a transaction reads uncommitted data from the another transaction consider the following steps based on this situation.

Step1:- Transaction A inserted a row in a table but it has not yet committed.

Step2: The transaction B reads uncommitted data which was inserted by transaction A

step3: The transaction A roll back its changes.

Step4: At this point the transaction B is left with the data of a row which does not exist.

b) Non-Repeatable Read:- In Non-Repeatable Read a transaction re-reads the data of another transaction. Consider the following steps in this situation.

step1:- Transaction A reads the data

step2: Transaction B modify or delete the same row and committed the changes.

Step3: Transaction A re-reads the same row and find the same row was modify or deleted

c) pantom Read:- In pantom read when a transaction a re-execute on same query and it finds more no. of rows that are satisfy the same query.

step1:- Transaction A Executes a query (Q) and

find 'x' no. of rows matching the same query.

step2: Transaction B inserts rows that are satisfy the same query (Q) and committed.

step3: Transaction A re-execute the same query and it finds '4' no. of rows (4>x) that are matching the same query.

Some SQL Command Line →

To providing the following transaction isolation →

For the above three situations for data consistency each isolation level will describe whether it is permitted or not.

permitted for data inconsistance situations

1) Read uncommitted
2) Read Committed
3) Repeatable read
4) Serialization

	Dirty Read	Non-Repeatable Read	phantom Read
1.	permitted	Permitted	permitted
2.	Not permitted	permitted	permitted
3.	Not permitted	Not permitted	permitted
4.	Not permitted	Not permitted	Not permitted

A javas has following four constant in order to represent the four isolation level.

These Constants are provided in Connection class. Those Constants as follows.

1) TRANSACTION_READ_UNCOMMITTED
2) TRANSACTION_READ_COMMITTED
3) TRANSACTION_REPEATABLE_READ
4) TRANSACTION_SERIALIZATION

we can set transaction isolation level for a particular data base connection using the following method.

Syntax: void set-TransactionIsolation-level (int level);

Ex:-

Con. setTransactionIsolationLevel(Connection.TRANSACTION_READ_COMMITTED);

where Con represents the object of Connection class. we have the following other methods.

Syntax: int getTransactionLevel();

It is used to get the current transaction isolation level.

Syntax: boolean supports_Transaction_Isolation(int level);

It will return true if a Connection is supporting Transaction Isolation level.

Syntax: boolean supportsTransaction();

If will return true whenever the Connection is supporting Transactions.

JDBC Types to Java Mapping:-

The JDBC API has several dataType. These data types are Mapping with Java API as follows.

JDBC Type	Java Type
1) ARRAY	java.sql.Array;
2) BIGINT	long
3) BIT	byte
4) BOOLEAN	byte
5) BINARY	byte[]
6) BLOB	java.sql.Blob
7) CHAR	String
8) CLOB	java.sql.clob
9) DATE	java.sql.Date
10) DATE	java.sql.localDate
11) DECIMAL	java.math.Bigdecimal
12) DOUBLE	double
13) FLOAT	double
14) INTEGER	int
15) JAVA_OBJECT	underline class object

16) LONGVARCHAR	String
17) LONGVARBINARY	~~Stri~~ byte array/byte[]
18) LONGNVARCHAR	String
19) NCHAR	String
20) NCLOCK	java.sql.NClock
21) NUMERIC	java.math.Bigdecimal
22) NVARCHAR	String
23) REAL	Float
24) REF	java.sql.Ref
25) REF_CURSOR	java.sql.Resultset
26) ROW-ID	java.sql.RowId
27) SMALLINT	Short
28) SQLXML	java.sql.SQLXML
29) ABSTRACT	java.sql.Struct
30) TIME	java.sql.Time
31) TIMESTAMP	java.sql.Timestamp
32) TIME-WITH-TIMEZONE	java.sql.OffsetTime
33) TIMESTAMP-WITH-TIMEZONE	java.sql.OffsetDataTime
34) TIME	java.sql.LocalTime
35) TINYINT	byte
36) VARCHAR	String
37) VARBINARY	byte[]

no pending

* Java 8 has a different class in order to mention a JDBC typed constants. we have Type class in order to represents the JDBC type constants.

for Example, if you want to set a null value for the parameter index to so that we have to set Null(). we have following syntax.

void setNull(int index, int Type)

The parameter two has index type so that we can set null has follows.

Stmt.setNull(2, java.sql.Type.Integer)

where stmt is a statement object.

* Java 8 has the following methods in order to set, get and update values from the DBMS or data base

getXxx()
SetXxx()
updataXxx()

where Xxx represents the respective value mapped data type.

* A getXxx() is used to get the value from the JDBC to java program.
* A SetXxx() is used to set the value to the JDBC
* The updataXxx() is used to retrieve the value from JDBC and forward the updated value to the JDBC.

for Example, in a table if we have an integer attribute then we can get the integer value using getInt(), we can set using setInt(). and also we can update using updateInt().

The varchar datatype attribute is mapped with String data type in java program so, that we have getString(), setString(), updateString(). If we have a Real data type attribute then we have getfloat(), Setfloat(), updatefloat(), because the Real data type is mapped with Mapped float data type in java program.

Knowing about the database:-

we can know about the data base using DatabaseMetaData interface in these interface we have getMetaData() to get all details of particular

data base we have the following methods in order to know the details of a product and driver and supported features of a data base

A getmetadata() is used to get a connected data base details so, that we can access this method using a connection class object.

DatabaseMetaData dbmd = Con.getMetaData();

1) getDatabaseProductName() - it is used to get the data base product name.
2) getDatabaseProductVersion() - it is used to get the data base product version.
3) getURL() - it is used to get Connection url.
4) getDriverName() - it is used to get data base driver name
5) getDriverVersion() - it is used to get database driver version.
6) SupportsANSI92EntryLevelSQL() - it will return a boolean value if it is supporting for Entry level SQL
7) SupportsANSI92IntermediateSQL() - it will return a boolean value when it supports for Intermediate SQL.
8) SupportsANSI92fullSQL() - it will also return a boolean value when it supporting for Full SQL
9) SupportsBatchupdate() - it will also return a boolean value if it is supporting for Batchupdate().

Ex:- Write a java program to know the details of a data base.

```
import java.sql.*;
Class MetaData
{
 public static void main(String[] args)
 {
 try
 {
 Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
 Connection Con = DriverManager.getConnection("jdbc:derby:CSEDB", "cse", "cse");
 DatabaseMetaData dbmd = Con.getMetaData();
 System.out.Println("Details of product");
 System.out.Println("productName:"+dbmb.getDatabaseproductName());
```

```
System.out.println("ProductVersion:"+dbmd.getDatabaseProductVersion());
System.out.println("URL:"+dbmd.getURL());
System.out.println("Details of Driver");
System.out.println("Driver Name:"+dbmd.getDriverName());
System.out.println("DriverVersion:"+dbmd.getDriverVersion());
System.out.println("Supported Features");
System.out.println("SupportsEntryLevelSQL:"+dbmd.supportsANSI92EntryLevelSQL());
System.out.println("supportsIntermediateSQL:"+dbmd.supportsANSI92IntermediateSQL());
System.out.println("supports Full SQL:"+dbmd.supportsANSI92FullSQL());
System.out.println("supports Batch updates:"+dbmd.supportsBatchUpdates());
catch(SQLExecutionE)
 {
 System.out.println(e);
}
finally
 {
 Connection con.Close();
  }
}
}
```

Executing SQL Statements:-

We can Execute SQL statements using JDBC drivers. we have the following categories of SQL statements

1) DDL (Data definition language) :- It has Create and Alter etc.

2) DML (Data Modification language):- It has Select, insert, update and delete etc.

3) DCL (Data Control language):- It has Grant and Revoke

4) TCL (Transaction Control language):- It has Commit and Rollback etc.

In Java we can Represent a query statements by using following three interfaces

a) Statement
b) PreparedStatement
c) Callable Statement

Results of Executing SQL statements:-

For Every Execution of sql statements, it will return a group of records, or an updated count (The no. of inserted rows, no. of deleted rows, no. of updated rows).

using statement interface:- The statement interface will create a query statement using create statement Method by calling on Connection class object. Here we can represent a query in the form of string. So, that the statement interface always accepts the string format but using statement interface there is a chance of SQL injection (Hack the table information) and it will also degrades the performance of database.

* It has the following 3 methods in order to execute a query.

boolean execute(String sql) throws SQLException)
int executeupdate(String sql) throws SQLException)
Resultset executeQuery(String sql) throws SQLException)

The Execute method will return a boolean value on successful execution of a query which does not return any integer value.

Ex:- Creation of a table.

* The Execute update method will return an integer value on successful Execution of a query. Here the integer value must be an updated count.

Ex:- insertion, updation and deletion.

* The Execute query method will return the Resultset type which will be equals to a group of records or rows.

Ex:- Selection of a table

Note: we cannot use Execute update and Execute method for selection query because these are not return any Resultset values.

we cannot use Execute query to perform insertion, updation, deletion because it will return a Resultset type.

In Resultset class we have a next() in order to move from one row to another row. And we can get the row values using getXxx().

Ex:- If we are reading studentid value then we can use getxxx() as follows

int id = rs.getInt("sid");

where 'rs' is a resultset object and sid is a attribute name of student id.

Ex:- Statement stmt = Con.CreateStatement() (for using stmt interface)
String sql = "insert into person values (501, 'cse')";
stmt.executeUpdate(sql)

2. preparedStatement interface:- The preparedStatement interface is a subinterface of statement interface so, that it will inherits properties from statement interface. we can create a preparedStatement using prepareStatement() by calling on connection class object. In preparedStatement the query is executed first then it call multiple times, will using statement interface we are representing the row values, but in preparedStatement instead of row values we have place holders (?)

The prepared statements also executed using following 3 methods

Execute()
ExecuteUpdate()
ExecuteQuery()

we can create a preparedstatement as follows

Syn: preparedStatement ps = Con.prepareStatement()

Every place holder is replaced by a value which is set by using setXxx(). Each place holder has it's own indexes for example if we have three place holders (?, ?, ?) then the first placeholder index is 1. second place holder index 2 and third place holder index is 3.

Example:

```
preparedStatement ps = Con.prepareStatement("inset
                       into person values (?, ?)");
ps.setInt(1, 501);
ps.setString(2, "cse");
ps.executeUpdate();
```

using Callable statement interface:

The CallableStatement interface inherits the properties from preparedStatement interface. we can create a callableStatement by using prepareCall() on connection class object, generally we have the stored procedure in our data base, the procedures are used to perform certain task on our table.

Before performing we have to call the stored procedures using our java program. we have the following syntax in order to call stored procedures in callable statement interface.

syn: ? ={call <procedure-name>(param1, param2, ----)};

where ? represents the return value of called procedure where param1, param2, param3 --- are the parameters of procedure.

we have the following different syntaxes in order to call a procedure.

Syntax	Description
{call <procedure_name> ()}	There is no input parameters and no return value
{call <procedure_name (?,?)}	Accept two input parameters and no return value.
{?=call<procedure_name>(?,?)}	Accept two input parameters and a return value
{call<procedure_name>(?,?,?)}	Accept two input parameters and one output parameter
{?=call procedure_name>(?,??)}	Accept two input and one output and a return value
{?=call<procedure_name>()}	Accept no parameters and return value

we can create a procedure in oracle database as follows

```
Create or replace procedure
 <procedure_name>
 (
  parameter list ----
  )
  [is/AS]
    Begin
      statements;
    End;
```

In callable statements interface will call the stored procedure. Every procedure has three types of parameters.

1. IN type parameter
2. OUT type parameter
3. INOUT type parameter

1) IN parameter:- IN parameter means that the caller has to pass the value to the stored procedure we can pass the value using any one the setXxx( ) we have to pass these values before the execution of Callable statement.

Ex:-
```
//Callable statement
cstmt.setInt(1, 501);
cstmt.setString(2,"cse");
//Execute statements
```

2) OUT type parameter:- In OUT type parameter the caller pass only the place holder to the stored procedure while executing the stored procedure it will set the value to the OUT type parameters after execution of stored procedure we will get the value using any one of the getXxx( ) but always OUT type parameter are registered with Callable stmt interface we have the following method to register OUT type parameter

```
Register OUTparameter(int index, int type);
```

After registration only we have to get the OUT type parameter

Ex:-
```
CallableStatement cstmt = con.prepareCall("{?= Call sum(?,?)}");
cstmt.setInt(2,10);
cstmt.setInt(3,10);
cstmt.registerOUTparameter(1, Types.INTEGER);
cstmt.execute();
int result = cstmt.getInt(1);
System.out.println("sum is:"+result);
```

we have the following procedure in order to execute above sample code.

```
Create or replace procedure
"sum"
(
x IN number,
y IN number,
z OUT number)
is
Begin
z = x+y
return z;
end;
```

3) INOUT type Parameter:- INOUT type parameter means is a combination of both IN and OUT type parameter first a caller send a value to the stored procedure. The stored procedure itself modify the value. After execution of stored procedure. The caller will receive the modified value

Ex:-
```
callablestatement cstmt = Con.prepareCall("{Call modify(?)}");
cstmt.SetDouble(1, 10.5);
cstmt.registerOUTparameter(1, Types.DOUBLE);
cstmt.execute();
double result = cstmt.getDouble(1);
system.out.println("value is:" + result);
```

```
Create or replace procedure
 "modify"
 (
  x  INOUT double
 )
 is
 begin
 x = 205;
  end;
```

* Execute insert query to insert values into person table using callable statement interface.

```
CallableStatement cstmt = Con.prepareCall("{call insertion(?,?)}");
cstmt.setInt(1, 501);
cstmt.setString(2, "CSE");
cstmt.executeupdate();
```

* The following is the insertion procedure which will be executed at data base the above sample code simply called the insertion stored procedure and it will & insert the values into the person table.

```
Create or replace procedure
 "insertion"
 (
  id IN number,
  Name IN varchar2
 )
 is
 begin
```

```
insert into person values (id, name);
end;
/
```

Write a java program to connect with the database and perform various Query operations on our database.

```
import java.sql.*;
import java.util.*;
Class JDBC
{
Static scanner sc;
Static string drive = "oracle.jdbc.OracleDriver";
Static string url = "jdbc:oracle:thin:naga/naga@
                                    localhost:1521/xe";
static Connection con = null;
Static statement stmt = null;
Static PreparedStatement ps = null;
Static CallableStatement cs = null;
public static void main (String[] args)throws Exceptior
{
Class.forName (driver);
System.out.println("Connecting to the Database...");
con = DriverManager.getConnection (url);
System.out.println("Database is Connected Successfully");
while(1) (or) while(true)
{
System.out.println("1.Create\n 2.insert\n 3.select\n 4.deletion\r
                    5.update\n 6.close\n 7.exit\n");
System.out.println("Enter your choice\n");
switch (sc.nextInt()) sc=new scanner(System.in);
{
Case 1: Create();
        break;
Case 2: insert();
        break;
Case 3: select();
        break;
Case 4: delete();
        break;
Case 5: update();
        break;
```

```
case 6: close();
        break;
case 7: return;
default : System.out.println("invalid choice");
}
}
}
public static void create() throws Exception
{ try
{
String sql1 =" create table customer1("+"id int,"
                                      +"name varchar2,"
                                      +"email varchar2,"
                                      +"phone varchar2)" ;
  Stmt = con.createStatement();
  if( !stmt.execute(sql1)
  system.out.println('customer1 table created successfully');
  else
  System.out.println(" problem in table creation");
  }
  Catch (SQLException e)
  {
  System.out.println(e);
  }
}
public static void insert() throws Exception
{
try
{
PS = con.prepareStatement("insert into customer1 values(?,?,?,?)
                                                        ");
System.out.println("Enter customer1 id");
PS.setInt(1,sc.nextInt());
System.out.println("Enter customer name");
 PS.setString(sc.next());
System.out.println("Enter1 Customer1 Email");
 ps.setString(sc.next());
System.out.println("Enter1 Customer phone ");
  PS.setString(sc.next());
  PS.executeUpdate();
System.out.println("Records inserted successfully");
}
```

```
catch( SQLException e)
{
System.out.println(e);
}
}

public static void select() throws Exception
{
try
{
String sql2 = "select * from customer1";
  Stmt = Con.CreateStatement()
System.out.println("Records are ---");
ResultSet rs = Stmt.executeQuery(sql2);
while(rs.next())
{
int id = rs.getInt(1);
Stringname = rs.getString(2);
Stringemail = rs.getString(3);
Stringphone = rs.getString(4);
system.out.println(" "+id+ "\t" +name+"\t" +email+"\t" +
                                                    phone);
}
Catch ( SQLException e)
{
System.out.println(e);
}
}

Public static void delete() throws Exception
{
try
{
PS = Cm.prepareStatement("delete from Customer where id = ?");
System.out.println(" Enter customer id");
PS.setInt(sc.nextInt());
PS.ExecuteUpdate();
system.out.println("Record deleted successfully");
}

Catch ( SQLException e)
{
System.out.println(e);
}
}
```

```
public static void update() throws Exception
{
try
{
cs = con.prepareCall("{call modify(?,?,?)}");
system.out.println("Enter Custometer id");
 cs.setInt(1, sc.nextInt());
System.out.println(" Enter Customer new email");
 cs.setString(2, sc.next());
 cs.registerOutParameter(3, Types.INTEGER);
 cs.executeupdate();
  int result = cs.getInt(3);
  system.out.println(result +" record, updated");
 }
 catch (SQLException e)
 {
system.out.println(e);
}
}
public static void close() throws Exception
{
try
{
 con.close();
 Stmt.close();
 ps.close();
 cs.close();
 System.out.println(" the Connection was closed successfully");
}
catch(SQLExecution e)
{
System.out.println(e);
}
}
}
```

Creation of new user in Oracle:

step1: we have to open Run SQL Command line

step2: sql > Connect System

Enter password : Lenovo

Connected
Sql > Create user CSE identified by CSE;
user Created
Sql > grant Connect, resource to CSE;
grant Created (or) success
Sql> Connect CSE
Enter password : CSE
Connected
Sql >

Creation of stored procedure in oracle:

```
Create or replace procedure
& " modify"
(
 Cid IN. integer;
 newemail IN varchar2;
 uCount OUT integer
 )
 is
 begin
 update customer set email = new email where id = cid;
 uCount := 1;
 end;
/
```

Processing ResultSet:-

when we Execute Select Query it will return a group of records these records are return by a resultset interface. A Resultset interface object always maintains a cursor which points to a row in a resultset we can move or scroll the cursor to a specific row in the resultset to access or manipulate the data. Resultset has the following three properties

1) scrollability
2) Concurrency
3) holdability

1) scrollability: The scrollability property always refers the moment of cursor in a resultset the moment may be either forward or backward direction by default Every resultset has forward only direction

we can also move the cursor by setting bidirectional scrollable property. It has the following three constants.

1) TYPE_FORWARD_ONLY :- This constant refers the moment of cursor in forward direction only
2) TYPE_SCROLL_SENSITIVE:- It will refers the moment of cursor either forward or backward direction and also it will make changes to our database whenever the database is modified by the other transactions or statements.
3) TYPE_SCROLL_INSENSITIVE :- It will refers the moment of cursor either forward or backward direction but it won't changes to the database even though the database is changed by other transactions.

* By default every resultset has forward direction but if you want to set bidirection scrollable property we have to use the following methods

1) while using statement interface we can call create statement method in order to create a statement object

```
Statement CreateStatement();
Statement CreateStatement(int scrollability, int Concurrency);
Statement CreateStatement(int scrollability, int Concurrency,
                          int holdability);
```

2) if we are using preparedStatement interface then we have to use the following methods to set bidirectional scrollability

```
PreparedStatement prepareStatement(String sql);
PreparedStatement prepareStatement(String sql, int
                          scrollability, int Concurrency);
PreparedStatement prepareStatement(String sql, int
                          Scrollability, int Concurrency, int holdability);
```

3) If you are using Callable Statement interface then we can use the following methods to set the bidirectional scrollability.

```
CallableStatement prepareCall(String sql);
CallableStatement prepareCall( String sql, int scrollability,
                          int Concurrency);
CallableStatement prepareCall(String sql, int scrollability,
                          int Concurrency, int holdability);
```

Concurrency:- This property refers ability of Resultset to update the data. By default every Resultset has read only Concurrency we can also make it has updatable by using concurrency property. it has the following two constants.

1) CONCUR_READ_ONLY:- Makes the Resultset has readonly
2) CONCUR_UPDATABLE:- Makes the Resultset has updatable

Holdability:- This property refers the state of Resultset which may be close or keeps open after the transaction is committed. it has the following two constants.

1) HOLDS_CURSOR_OVER_COMMIT:- It will open the Resultset after a transaction is committed
2) CLOSE_CURSOR_AT_COMMIT:- It will close the Resultset after a transaction is committed.

* In the above methods (CreateStatement(), PreparedStatement(), PreparedCall()). we have scrollability, Concurrency and holdability has integer type. These integers are referred the respective property constants. we have to use these constants by using Resultset interface. for Example

```
Statement stmt = Con.CreateStatement(Resultset.TYPE-SCROLL
                 SENSITIVE, Resultset.CONCUR_READ_ONLY,
                            Resultset.CLOSE_CURSOR_AT_COMMIT);
```

*we have to check whether above three properties are supported by data base or not. we have the following three methods to check the supportability

SupportsResultSetType() → used to check scrollability property. if the property is existed it will return true otherwise false.

SupportsResultSetConcurrency() → used to check Concurrency property. if it is existed it return true otherwise false.

SupportsResultSetHoldability() → used to check holdability property. if it is existed it returns true otherwise false.

* The following methods are used to retrieve the property values (scrollability, Concurrency, and holdability)

int getType() → used to get the scrollability property.
int getConcurrency() → used to get Concurrency property
int getHoldability() → used to get holdability property.
we can know the current position of a cursor using "getRow()" in a ResultSet.
we have two categories of cursor moment methods
1) RelativeCursorMoment():- It will move the cursor either forward or backward directions by the specified no. of rows from its current position
we have the following Relative cursor moment Methods.
1) boolean next():- move the cursor in forward direction only
2) boolean previous():- move the cursor backward direction only
3) boolean Relative(int row):- move the cursor either forward or backward direction by the specified no. of rows from its current position.
Ex:- Relative(2):- it will move the cursor in forward direction by the 2 rows from its current position.
Ex:- Relative(-2):- it will move the cursor in backward direction by the 2 rows from its current position
2) AbsoluteCursonMoment():- These are used to move either forward or backward direction by the specified row we have the following absolute cursor moment methods.
1) boolean first():- it will set the cursor in a first row of a ResultSet.
2) boolean last():- it will set the cursor at the last row of a ResultSet.
3) boolean absolute(int row):- it will set the cursor either forward or backward position by the specific row numbers.

Ex:- absolute(3) :- it will set the cursor at the 3rd row position from the top.

absolute(-3) :- it will set the cursor at the 3rd row position from the bottom.

Ex:- programme

```
import java.sql.*;
import java.util.*;
Public class PR (ProcessingResultSet)
{
 static string driver = "oracle.jdbc.OracleDriver";
 Static string url = "jdbc:oracle:thin:swetha/nani@
                     localhost:1521/xe";
 static PreparedStatement ps = null;
 Static ResultSet rs = null;
 Public static void main(string args[])
 {
 try
 {
 Class.forName(driver);
 Connection Con = DriverManager.getConnection(url);
 System.out.println("database is Connected successfully");
 PS = Con.prepareStatement("select * from Student", Resultset.
      TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATE);
 rs = Ps.executeQuery();
 rs.first();
 system.out.println(" "+rs.getInt(1)+"\t"+rs.getString(2)+
      "\t"+rs.getString(3)+"\t"+rs.getString(4));
 rs.last();
 rs.absolute(2);
 System.out.println("current position:"+rs.getRow());
 while(rs.relative(2))
 {
System.out.println(" "+rs.getInt(1)+"\t"+rs.getString(2)+
      "\t"+rs.getString(3)+"\t"+rs.getString(4));
}
rs.beforefirst();
while(rs.next())
{
System.out.println(" "+rs.getInt(1)+"\t"+rs.getString(2)+
      "\t"+rs.getString(3)+"\t"+rs.getString(4));
```

```
}
rs.afterLast();
while(rs.previous())
{
System.out.println(" "+rs.getInt(1)+"\t"+rs.getString(2)+
        "\t"+rs.getString(3)+"\t"+rs.getString(4));

}
}catch(SQLException e)
{
System.out.println(e);
}
ps.close();
con.close();
rs.close();
}
}
```

making Changes to the ResultSet:-

we can make the changes to the Resultset by using following three methods.

i) insertRow():- we can insert a row into a result set by using following procedure

ii) we have to move our cursor to a particular position to insert a row. Generally, we have two imaginary row in ResultSet we have one more imaginary row that is called insert Row. we can move the cursor to insertRow using following method.

moveToInsertRow() → This method will move the cursor to the insertRow and also it will remember the current previous position of a cursor

ii) Now we have to use one of the method of updateXXX() in order to set the values to the newly inserted row.

```
updateInt(1, 5801);
updateString(2,"MTech");
```

iii) Now, we have to use the following method to insert the updated values into the ResultSet

```
insertRow();
```

iv) Now we have to move the cursor to the current previous position. The following method is

moveToCurrentRow()

2) updateRow():- It is used to update the row in a ResultSet before updating we have to use the following procedure.

Step1:- first we have to set the cursor position at a particular row. using absolute().

Ex:- rs.absolute(2)

Step2:- Now we have to use any one of the updateXxx() to change the row values.

Ex:- rs.updateString(3, "pe@gmail.com");

Step3:- Now we have to use the following method to affect the changes into a ResultSet.

Ex:- updateRow();

3) deleteRow():- It is used to delete a row from the ResultSet. we have the following procedure to delete a row.

step1:- first we have to set a particular the cursor position at a particular row by using absolute().

Ex:- rs.absolute(3);

Step2: Now we have to use deleteRow() to delete a particular row from the Resultset.

Ex:- rs.deleteRow();

Ex:- Programme

```
import java.sql.*;
import java.util.*;
class JDBC
{
static Scanner sc;
static String drive="oracle.jdbc.OracleDriver";
static String url="jdbc:oracle:thin:swetha/nani@
                                localhost:1521:xe";
  Static Connection con=null;
  Static ResultSet rs=null;
  Static PreparedStatement ps=null;
  Public static void main(String[] args)throws Exception
  {
  Class.forName(drive);
```

```
System.out.println("Connecting to the database...");
con = DriverManager.getConnection(url);
System.out.println("Database is Connected Successfully");
ps = con.prepareStatement("select id, name, email, phone
    from student", ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.
                                        CONCUR_UPDATE);
rs = ps.executeQuery();
while(true)
{
System.out.println("1.insert\n 2.update\n 3.delete\n 4.select\n
                                    5.close\n 6.exit\n");
s.o.p("Enter your choice");
switch(sc.nextInt())
{
case 1: insert();
        break;
case 2: update();
        break;
case 3: delete();
        break;
case 4: select();
        break;
case 5: close();
        break;
case 6: return;
default: s.o.p("invalid choice");
}
}
}
public void insert() throws Exception
{   static
rs.moveToInsertRow();
s.o.pln("Enter student id");
rs.updateInt(1, sc.nextInt());
s.o.pln("Enter student Name");
rs.updateString(2, sc.next());
s.o.pln("Enter student email");
rs.updateString(3, sc.next());
s.o.pln("Enter student phone")
rs.updateString(4, sc.next());
```

```
rs.insertRow();
rs.moveToCurrentRow();
S.o.println("Row inserted successfully");
}
Public static void update() throws Exception
{ try
{
S.o.println("Enter Row number to update");
rs.absolute(sc.nextInt());
rs.updateString(3, "CSIT@gmail.com");
rs.updateRow();
S.o.println(" Row updated successfully");
}
Catch(SQLException e)
{
S.o.println(e);
}
}
public static void delete() throws Exception
{
try
{
S.o.println("Enter Row number to delete");
rs.absolute(sc.nextInt());
rs.deleteRow();
S.o.println(" Row deleted successfully");
}
Catch(SQLException e)
{
S.o.println(e);
}
}
public static void select() throws Exception
{
try
{
rs=PS.executeQuery();
rs.beforeFirst();
while(rs.next())
{
```

```
S.o.println(" "+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getString(4));
}
}
catch(SQLException e)
{
S.o.println(e);
}
}
public static void close() throws Exception
{
try
{
ps.close();
con.close();
rs.close();
}
Catch(SQLException e)
{
S.o.println(e);
}
}
```

Handling multiple Resultset from a Statement:-

Every SQL statement will return multiple statements generally will use stored procedure in order to execute multiple queries. Some results may have only updated count. But some results may have the resultset Type. whenever we are using execute() on stored procedure.

If the stored procedure will return an updated count then the execute method will returns false.

If the stored procedure will return ResultSet Type then the execute() will return true. we have the following three methods. To handle multiple results from the statement.

1) getResultSet() - it is used to get a ResultSet Type from the Statement.

2) getMoreResultSet() - It is used to get the more

no. of Resultset from the statement.
4. If the statement has more no. of Resultset then this method will return true. otherwise (if there are no Resultset Type) false.

3) getUpdateCount(): it is used to get the updated count type Results from the statement.

The above three methods are smoothly works for mysql database. ofcourse these are also supported for all databases except oracle. To handle multiple results in oracle we have to use cursors in oracle stored procedures.

Ex:- my sql database:-

Create the following stored procedure in my sql Database.

```
Create or Replace procedure
"getDetails"
is
begin
Select * from First;
Select * from second;
end;
/
```

write a java programme to handle multiple results from the statement using my sql databases.

```
import java.sql.*;
import java.util.*;
Public class MRS
{
Public static void main(String[] args)
{
try
{
Class.forName("my sql driver");
Connection con = DriverManager.getConnection("my sql
                                    Connection");
CallableStatement cs = con.prepareCall("{call getDetails()}");
boolean hasResult = cs.execute();
int count = cs.getUpdateCount();
while(hasResult)
{
```

```
ResultSet rs = cs.getResultSet();
while(rs.next())
{
//return the data values using getXxx() method;
}
hasResult = cs.getMoreResultSet();
}
cs.close();
}
catch(SQLException e)
{
s.o.println(e);
}
}
}
```

Oracle data base:-

Create a stored procedure in oracle data base to handle the multiple ResultSets.

```
create or Replace procedure
"getDetails"
x1(is (
 begin my cursor1 OUT SYS_REFCURSOR,
 select ~~*from first~~ mycursor2 OUT SYS_REFCURSOR
 ~~select * from second~~ )
 ~~open //~~                    is
 ~~//~~                         begin
                                open mycursor1 for
                                Select id as i, name as an from first;
                                open mycursor2 for
                                Select id as i, name as an from second;
                                end;
                                /
```

write a java program to handle multiple Results from the oracle database.

```
import java.sql.*;
import java.util.*;
Public Class MRS
{
 Public static void main(String[] args)
 {
```

```
try
{
Class.forName("oracle.jdbc.oracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:
                    thin:swetha/nani@localhost:1521/xe");
CallableStatement cs = con.prepareCall("{call getDetails(?,?)}");
cs.registerOutParameter(1, OracleTypes.CURSOR);
cs.registerOutParameter(2, OracleTypes.CURSOR);
cs.execute();
ResultSet rs1 = (ResultSet)cs.getObject(1);
S.o.println("Retrieving first table record...");
s.o.println("ID\t Name");
while(rs1.next())
{
S.o.println(" "+rs1.getInt(1)+"\t"+rs.getString(2));
}
ResultSet rs2 = (ResultSet)cs.getObject(2);
S.o.println("Retrieving second table records..");
S.o.println("ID\t Name");
while(rs2.next())
{
S.o.p("  "+rs2.getInt(1)+"\t"+rs.getString(2));
}
cs.close();
}
Catch(SQLException e)
{
S.o.println(e);
}
}
}
```