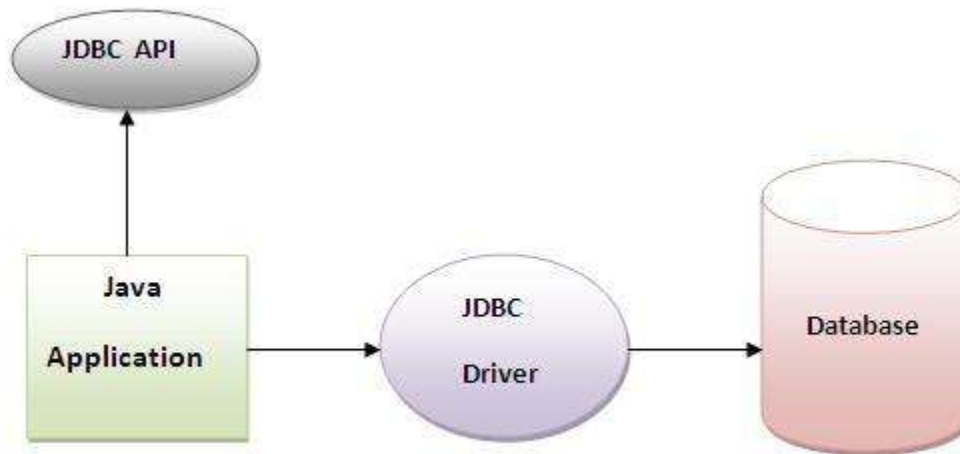


Java JDBC Tutorial

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

What is API

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

Topics in Java JDBC Tutorial

[2\) JDBC Drivers](#)

In this JDBC tutorial, we will learn 4 types of JDBC drivers, their advantages and disadvantages.

[3\) 5 Steps to connect to the database](#)

In this JDBC tutorial, we will see the 5 steps to connect to the database in java using JDBC.

[4\) Connectivity with Oracle using JDBC](#)

In this JDBC tutorial, we will connect a simple java program with the oracle database.

[5\) Connectivity with MySQL using JDBC](#)

In this JDBC tutorial, we will connect a simple java program with the mysql database.

[6\) Connectivity with Access without DSN](#)

Let's connect java application with access database with and without DSN.

[7\) DriverManager class](#)

In this JDBC tutorial, we will learn what does the DriverManager class and what are its methods.

[8\) Connection interface](#)

In this JDBC tutorial, we will learn what is Connection interface and what are its methods.

[9\) Statement interface](#)

In this JDBC tutorial, we will learn what is Statement interface and what are its methods.

[10\) ResultSet interface](#)

In this JDBC tutorial, we will learn what is ResultSet interface and what are its methods. Moreover, we will learn how we can make the ResultSet scrollable.

[11\) PreparedStatement Interface](#)

In this JDBC tutorial, we will learn what is benefit of PreparedStatement over Statement interface. We will see examples to insert, update or delete records using the PreparedStatement interface.

[12\) ResultSetMetaData interface](#)

In this JDBC tutorial, we will learn how we can get the metadata of a table.

[13\) DatabaseMetaData interface](#)

In this JDBC tutorial, we will learn how we can get the metadata of a database.

[14\) Storing image in Oracle](#)

Let's learn how to store image in the oracle database using JDBC.

[15\) Retrieving image from Oracle](#)

Let's see the simple example to retrieve image from the oracle database using JDBC.

[16\) Storing file in Oracle](#)

Let's see the simple example to store file in the oracle database using JDBC.

[17\) Retrieving file from Oracle](#)

Let's see the simple example to retrieve file from the oracle database using JDBC.

[18\) CallableStatement](#)

Let's see the code to call stored procedures and functions using CallableStatement.

19) Transaction Management using JDBC

Let's see the simple example to use transaction management using JDBC.

20) Batch Statement using JDBC

Let's see the code to execute batch of queries.

21) JDBC RowSet

Let's see the working of new JDBC RowSet interface.

JDBC Driver

1. JDBC Drivers
1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

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

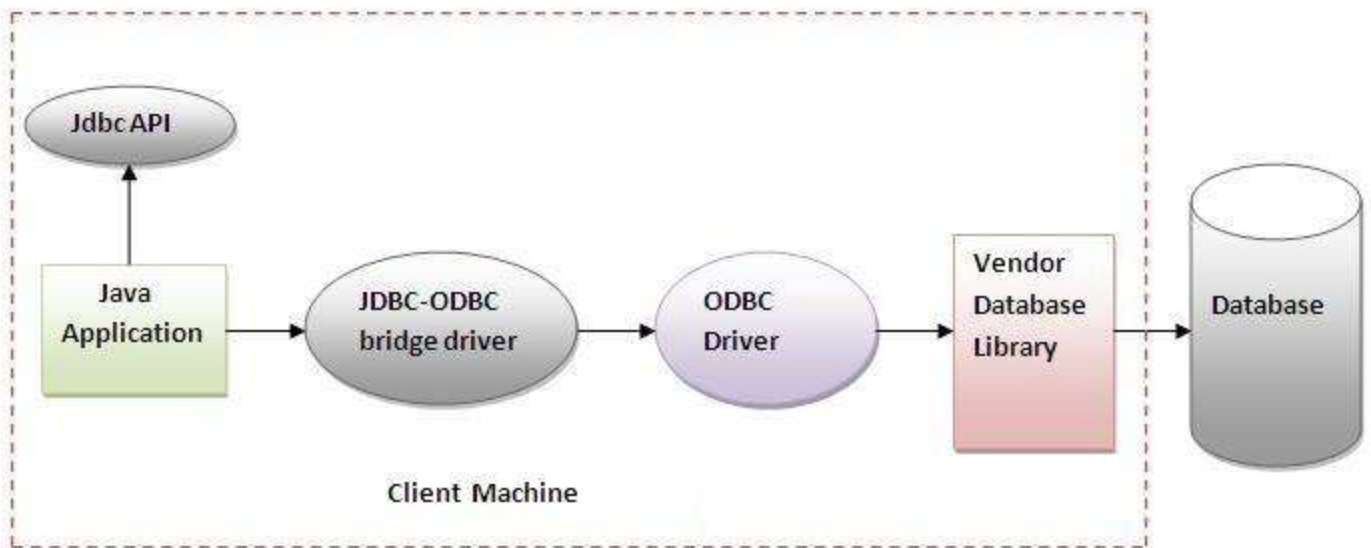


Figure- JDBC-ODBC Bridge Driver

Advantages:

- Easy to use.
- Can be easily connected to any database.

Disadvantages:

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

2) Native-API driver

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

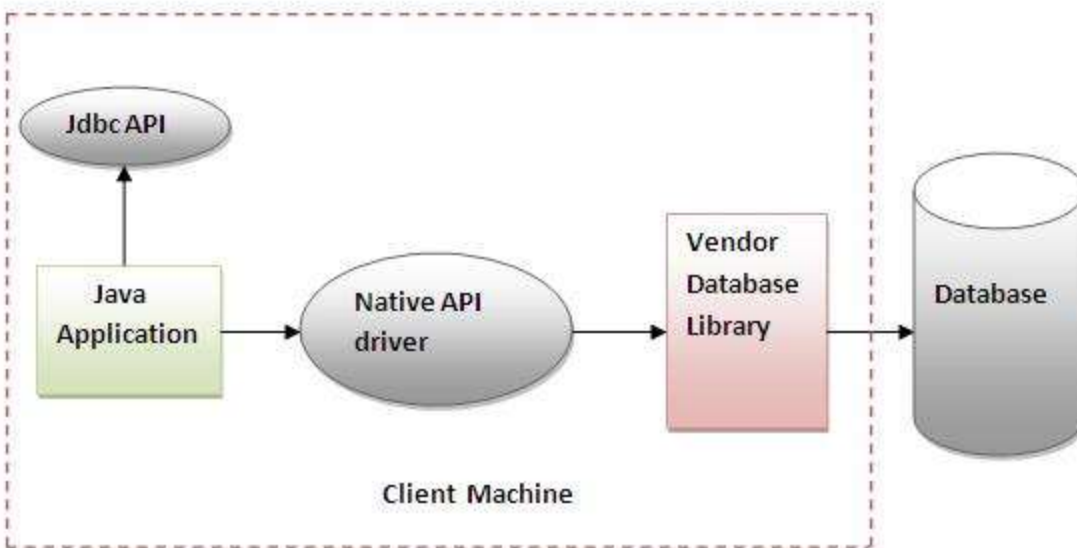


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

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

3) Network Protocol driver

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

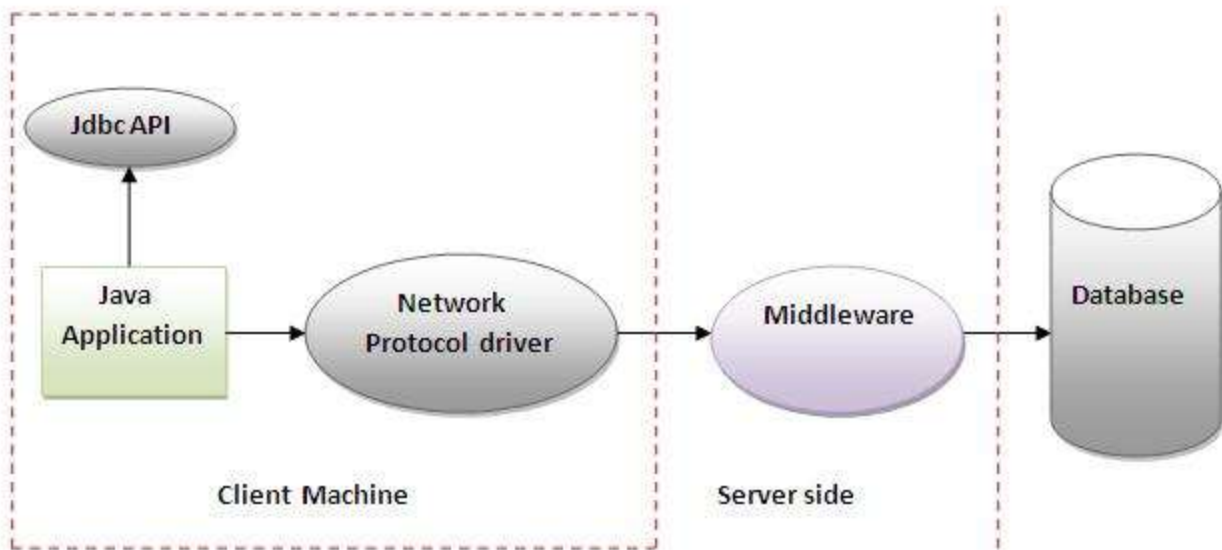


Figure- Network Protocol Driver

Advantage:

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

Disadvantages:

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

4) Thin driver

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

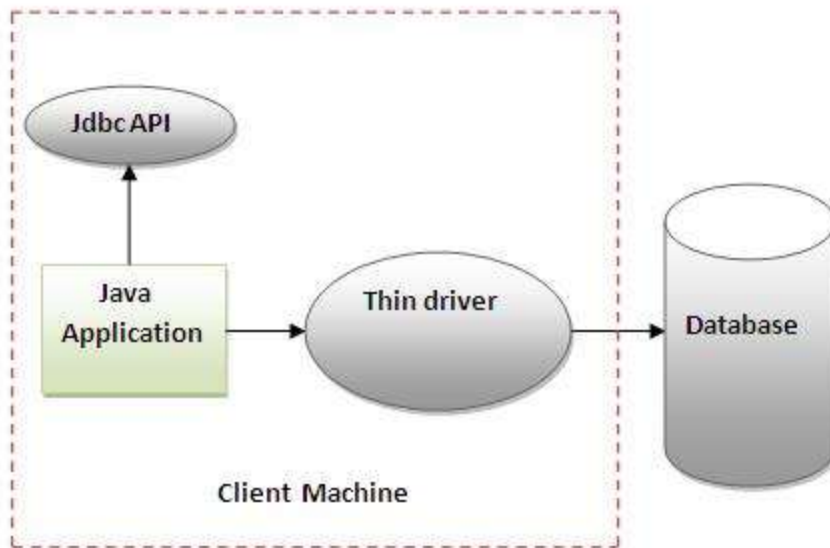


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depends on the Database.

5 Steps to connect to the database in java

1. 5 Steps to connect to the database in java

1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

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

- Register the driver class
 - Creating connection
 - Creating statement
 - Executing queries
 - Closing connection
-

1) Register the driver class

The `forName()` method of `Class` class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of `forName()` method

1. **public static void** `forName(String className)`**throws** `ClassNotFoundException`

Example to register the `OracleDriver` class

1. `Class.forName("oracle.jdbc.driver.OracleDriver");`

2) Create the connection object

The `getConnection()` method of `DriverManager` class is used to establish connection with the database.

Syntax of `getConnection()` method

1. **1) public static** `Connection getConnection(String url)`**throws** `SQLException`
2. **2) public static** `Connection getConnection(String url,String name,String password)`
3. **throws** `SQLException`

Example to establish connection with the Oracle database

1. `Connection con=DriverManager.getConnection(`
2. `"jdbc:oracle:thin:@localhost:1521:xe","system","password");`

3) Create the Statement object

The `createStatement()` method of `Connection` interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of `createStatement()` method

1. **public** Statement `createStatement()`**throws** SQLException

Example to create the statement object

1. Statement `stmt=con.createStatement();`
-

4) Execute the query

The `executeQuery()` method of `Statement` interface is used to execute queries to the database. This method returns the object of `ResultSet` that can be used to get all the records of a table.

Syntax of `executeQuery()` method

1. **public** ResultSet `executeQuery(String sql)`**throws** SQLException

Example to execute query

1. ResultSet `rs=stmt.executeQuery("select * from emp");`
 - 2.
 3. **while**(`rs.next()`){
 4. `System.out.println(rs.getInt(1)+" "+rs.getString(2));`
 5. }
-

5) Close the connection object

By closing connection object statement and `ResultSet` will be closed automatically. The

close() method of Connection interface is used to close the connection.

Syntax of close() method

1. **public void** close()**throws** SQLException

Example to close connection

1. con.close();

Example to connect to the Oracle database in java

For connecting java application with the oracle database, you need to follow 5 steps to perform database connectivity. In this example we are using Oracle10g as the database. So we need to know following information for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
3. **Username:** The default username for the oracle database is **system**.
4. **Password:** Password is given by the user at the time of installing the oracle database.

Let's first create a table in oracle database.

1. create table emp(id number(10),name varchar2(40),age number(3));

Example to Connect Java Application with Oracle database

In this example, system is the username and oracle is the password of the Oracle database.

1. **import** java.sql.*;

```
2. class OracleCon{
3. public static void main(String args[]){
4. try{
5. //step1 load the driver class
6. Class.forName("oracle.jdbc.driver.OracleDriver");
7.
8. //step2 create the connection object
9. Connection con=DriverManager.getConnection(
10. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
11.
12. //step3 create the statement object
13. Statement stmt=con.createStatement();
14.
15. //step4 execute query
16. ResultSet rs=stmt.executeQuery("select * from emp");
17. while(rs.next())
18. System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
19.
20. //step5 close the connection object
21. con.close();
22.
23. }catch(Exception e){ System.out.println(e);}
24.
25. }
26. }
```

[download this example](#)

The above example will fetch all the records of emp table.

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

[download the jar file ojdbc14.jar](#)

Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder

2. set classpath

1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

1. C:>set classpath=c:\folder\ojdbc14.jar;.;

How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;. as C:\oracle\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

Example to connect to the mysql database in java

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

In this example we are using MySQL as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use

IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.

3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password.

1. **import** java.sql.*;
2. **class** MysqlCon{
3. **public static void** main(String args[]){
4. **try**{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. //here sonoo is database name, root is username and password
9. Statement stmt=con.createStatement();
10. ResultSet rs=stmt.executeQuery("select * from emp");
11. **while**(rs.next())
12. System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
13. con.close();
14. }**catch**(Exception e){ System.out.println(e);}
15. }
16. }

[download this example](#)

The above example will fetch all the records of emp table.

To connect java application with the mysql database mysqlconnector.jar file is required to be loaded.

download the jar file [mysql-connector.jar](#)

Two ways to load the jar file:

1. paste the mysqlconnector.jar file in jre/lib/ext folder
2. set classpath

1) paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;. as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

Connectivity with Access without DSN

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)
2. With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

```
1. import java.sql.*;
2. class Test{
3. public static void main(String ar[]){
4. try{
5.     String database="student.mdb";//Here database exists in the current directory
6.
7.     String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
8.         DBQ=" + database + ";DriverID=22;READONLY=true";
9.
10.    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
11.    Connection c=DriverManager.getConnection(url);
12.    Statement st=c.createStatement();
13.    ResultSet rs=st.executeQuery("select * from login");
14.
15.    while(rs.next()){
16.        System.out.println(rs.getString(1));
17.    }
18.
19. }catch(Exception ee){System.out.println(ee);}
20.
21. }}
```

Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

```
1. import java.sql.*;
2. class Test{
```



```

3. public static void main(String ar[]){
4.   try{
5.     String url="jdbc:odbc:mydsn";
6.     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
7.     Connection c=DriverManager.getConnection(url);
8.     Statement st=c.createStatement();
9.     ResultSet rs=st.executeQuery("select * from login");
10.
11.    while(rs.next()){
12.      System.out.println(rs.getString(1));
13.    }
14.
15. }catch(Exception ee){System.out.println(ee);}
16.
17. }}

```

DriverManager class

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Commonly used methods of DriverManager class:

- | | |
|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| 1) <code>public static void registerDriver(Driver driver):</code> | is used to register the given driver with DriverManager. |
| 2) <code>public static void deregisterDriver(Driver driver):</code> | is used to deregister the given driver (drop the driver from the list) with DriverManager. |
| 3) <code>public static Connection getConnection(String url):</code> | is used to establish the connection with the specified url. |
| 4) <code>public static Connection getConnection(String url,String userName,String password):</code> | is used to establish the connection with the specified url, username and password. |

Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType, int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status. By default it is true.

4) public void commit(): saves the changes made since the previous commit/rollback permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

- 1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- 2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- 3) public boolean execute(String sql):** is used to execute queries that may return multiple results.
- 4) public int[] executeBatch():** is used to execute batch of commands.

Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
1. import java.sql.*;
2. class FetchRecord{
3.     public static void main(String args[])throws Exception{
4.         Class.forName("oracle.jdbc.driver.OracleDriver");
5.         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
6.         Statement stmt=con.createStatement();
7.
8.         //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
9.         //int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
10.        int result=stmt.executeUpdate("delete from emp765 where id=33");
11.        System.out.println(result+" records affected");
12.        con.close();
13.    }}
```

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

1. Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
2. ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int	is used to return the data of specified column index of the current

columnIndex):	row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```

1. import java.sql.*;
2. class FetchRecord{
3.     public static void main(String args[]){throws Exception{
4.
5.         Class.forName("oracle.jdbc.driver.OracleDriver");
6.         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
7.         Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
8.         ResultSet rs=stmt.executeQuery("select * from emp765");
9.
10.        //getting the record of 3rd row
11.        rs.absolute(3);
12.        System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
13.
14.        con.close();
15.    }}

```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```

1. String sql="insert into emp values(?,?,?);

```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. **public** PreparedStatement prepareStatement(String query)**throws** SQLException{}
-

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record

First of all create table as given below:

1. create table emp(id number(10),name varchar2(50));

Now insert records in this table by the code given below:

1. **import** java.sql.*;
2. **class** InsertPrepared{
3. **public static void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
- 6.
7. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
- 8.
9. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
10. stmt.setInt(1,101);//1 specifies the first parameter in the query
11. stmt.setString(2,"Ratan");
- 12.
13. **int** i=stmt.executeUpdate();
14. System.out.println(i+" records inserted");
- 15.
16. con.close();
- 17.
18. }**catch**(Exception e){ System.out.println(e);}
- 19.
20. }
21. }

[download this example](#)

Example of PreparedStatement interface that updates the record

1. PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
2. stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
3. stmt.setInt(2,101);
- 4.
5. **int** i=stmt.executeUpdate();

6. `System.out.println(i+" records updated");`
[download this example](#)
-

Example of PreparedStatement interface that deletes the record

1. `PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");`
 2. `stmt.setInt(1,101);`
 - 3.
 4. `int i=stmt.executeUpdate();`
 5. `System.out.println(i+" records deleted");`
[download this example](#)
-

Example of PreparedStatement interface that retrieve the records of a table

1. `PreparedStatement stmt=con.prepareStatement("select * from emp");`
 2. `ResultSet rs=stmt.executeQuery();`
 3. `while(rs.next()){`
 4. `System.out.println(rs.getInt(1)+" "+rs.getString(2));`
 5. `}`
[download this example](#)
-

Example of PreparedStatement to insert records until user press n

1. `import java.sql.*;`
2. `import java.io.*;`
3. `class RS{`
4. `public static void main(String args[])throws Exception{`
5. `Class.forName("oracle.jdbc.driver.OracleDriver");`
6. `Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");`
- 7.
8. `PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");`


```

9.
10. BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
11.
12. do{
13. System.out.println("enter id:");
14. int id=Integer.parseInt(br.readLine());
15. System.out.println("enter name:");
16. String name=br.readLine();
17. System.out.println("enter salary:");
18. float salary=Float.parseFloat(br.readLine());
19.
20. ps.setInt(1,id);
21. ps.setString(2,name);
22. ps.setFloat(3,salary);
23. int i=ps.executeUpdate();
24. System.out.println(i+" records affected");
25.
26. System.out.println("Do you want to continue: y/n");
27. String s=br.readLine();
28. if(s.startsWith("n")){
29. break;
30. }
31. }while(true);
32.
33. con.close();
34. }}

```

Java ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index

How to get the object of ResultSetMetaData:

The `getMetaData()` method of `ResultSet` interface returns the object of `ResultSetMetaData`. Syntax:

1. **public** `ResultSetMetaData` `getMetaData()`**throws** `SQLException`

Example of ResultSetMetaData interface :

1. **import** `java.sql.*;`
2. **class** `Rsmd{`
3. **public static void** `main(String args[]){`
4. **try{**
5. `Class.forName("oracle.jdbc.driver.OracleDriver");`
6. `Connection con=DriverManager.getConnection(`
7. `"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");`
- 8.
9. `PreparedStatement ps=con.prepareStatement("select * from emp");`
10. `ResultSet rs=ps.executeQuery();`
11. `ResultSetMetaData rsmd=rs.getMetaData();`
- 12.

```
13. System.out.println("Total columns: "+rsmd.getColumnCount());
14. System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
15. System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
16.
17. con.close();
18. }catch(Exception e){ System.out.println(e);}
19. }
20. }
```

```
Output:Total columns: 2
        Column Name of 1st column: ID
        Column Type Name of 1st column: NUMBER
```