**Java JDBC Tutorial**

* JDBC stands for Java Database Connectivity.
* JDBC is a Java API to connect and execute the query with the database.
* It is a part of JavaSE (Java Standard Edition).
* JDBC API uses JDBC drivers to connect with the database.
* There are four types of JDBC drivers:

JDBC-ODBC Bridge Driver,

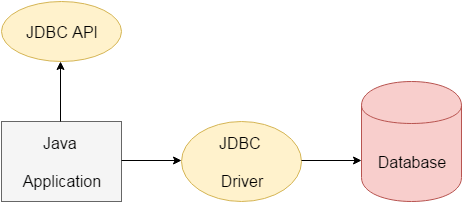
Native Driver,

Network Protocol Driver, and

Thin Driver

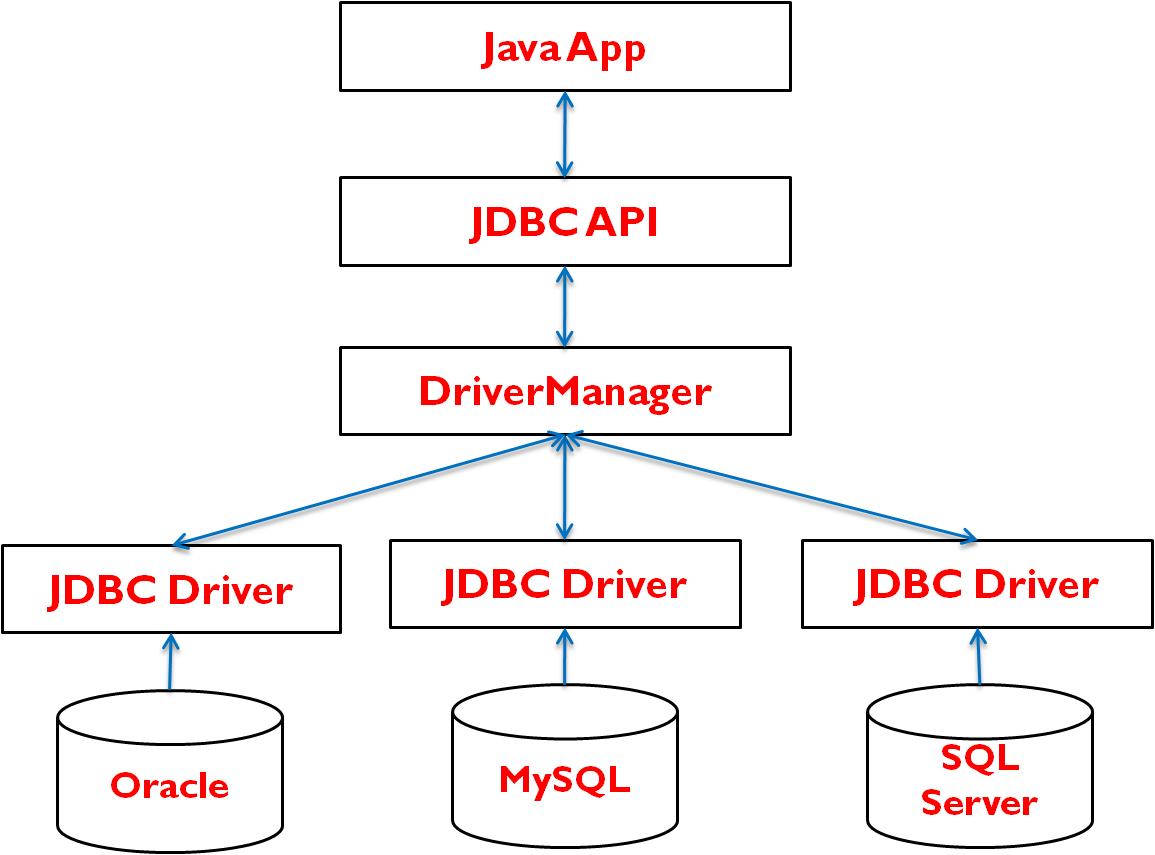
We have discussed the above four drivers in the next chapter.

* We can use JDBC API to access tabular data stored in any relational database.
* By the help of JDBC API, we can save, update, delete and fetch data from the database.
* It is like Open Database Connectivity (ODBC) provided by Microsoft.



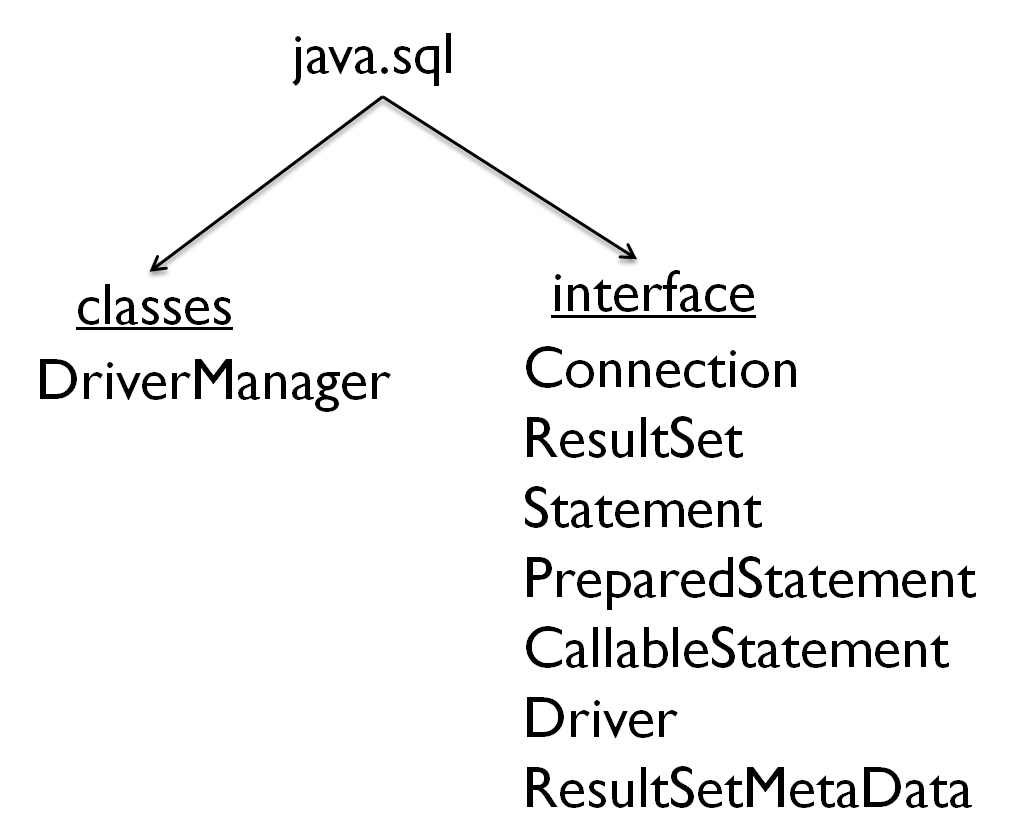
* The current version of JDBC is 4.3.
* It is the stable release since 21st September, 2017.
* It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API.

**JDBC Architecture**



**Types of Driver**

* They are 4 types of JDBC driver :
  + Type 1 (JDBC – ODBC bridge)
  + Type 2 (Native API / partly java driver)
  + Type 3 (Net protocol / All java driver )
  + Type 4 (Native protocol / All java drivert of popular *interfaces* of JDBC API are given below:



* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

A list of popular *classes* of JDBC API are given below:

* DriverManager class
* Blob class Clob class Types class

**Why Should We Use JDBC**

* Before JDBC, ODBC API was the database API to connect and execute the 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).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

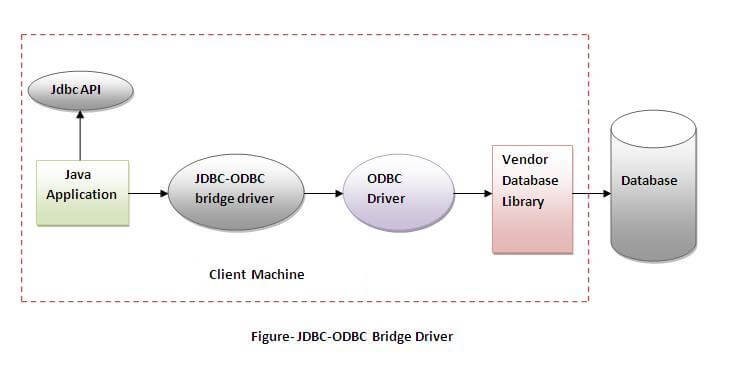
**JDBC Driver**

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.



In Java 8, the JDBC-ODBC Bridge has been removed.

* Oracle does not support the JDBC-ODBC Bridge from Java 8.
* Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

**Advantages:**

easy to use.

can be easily connected to any database.

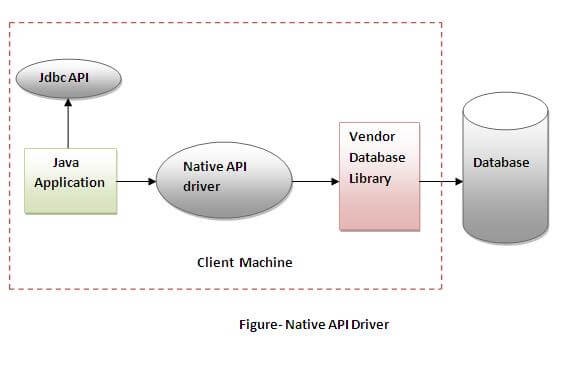
**Disadvantages:**

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.



**Advantage:**

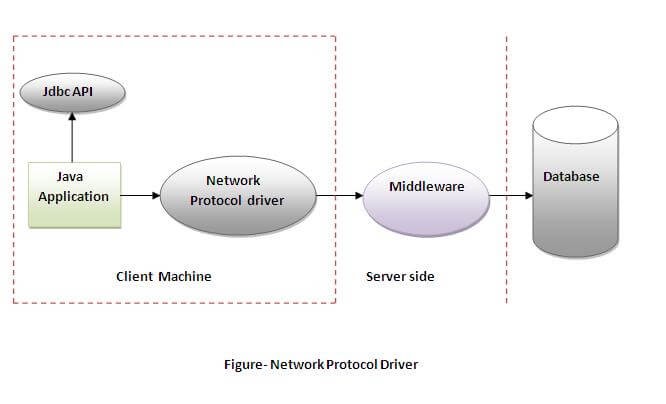
* performance upgraded than JDBC-ODBC bridge driver.

**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.



**Advantage:**

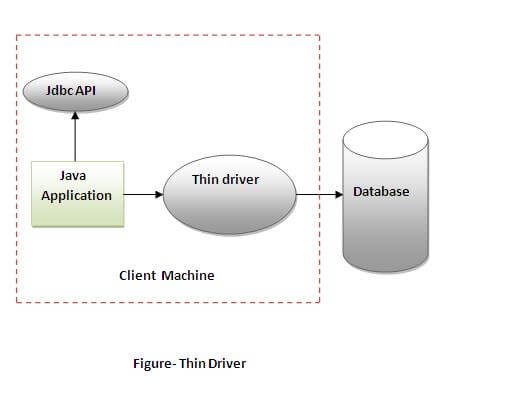
* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

**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 databasespecific coding to be done in the middle tier.

1. **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.



**Advantage:**

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

**Disadvantage:**

* Drivers depend on the Database.

Java Database Connectivity with 5 Steps

**Steps to develop JDBC Application**

1. Load JDBC driver.
2. Establish connection with database software.
3. Create Statement Object
4. Use statement object to send SQL queries to database software.
5. Use statement object to gather results from database software.
6. Process the results
7. close connection with database software

**Load JDBC driver**

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

**Establish connection with database software**

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/cdac","root","root");

**Create Statement Object**

Statement st = con.createStatement();

**Use statement object to send SQL queries to database software**

**Non- Select Query**

insert

update

delete

st.executeSQL("SQL query");

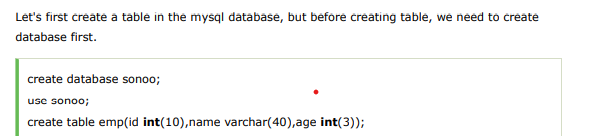
**Select Query**

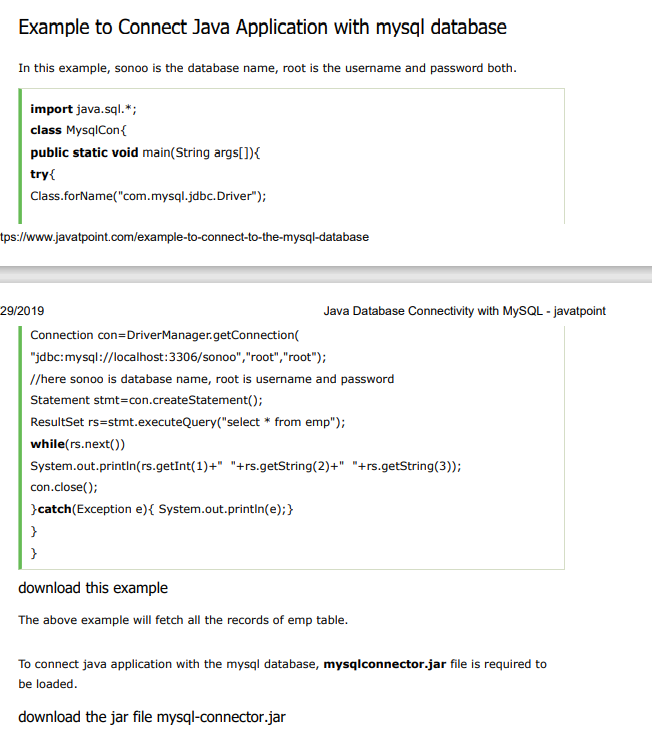
select

st.executeQuery("SQL query");

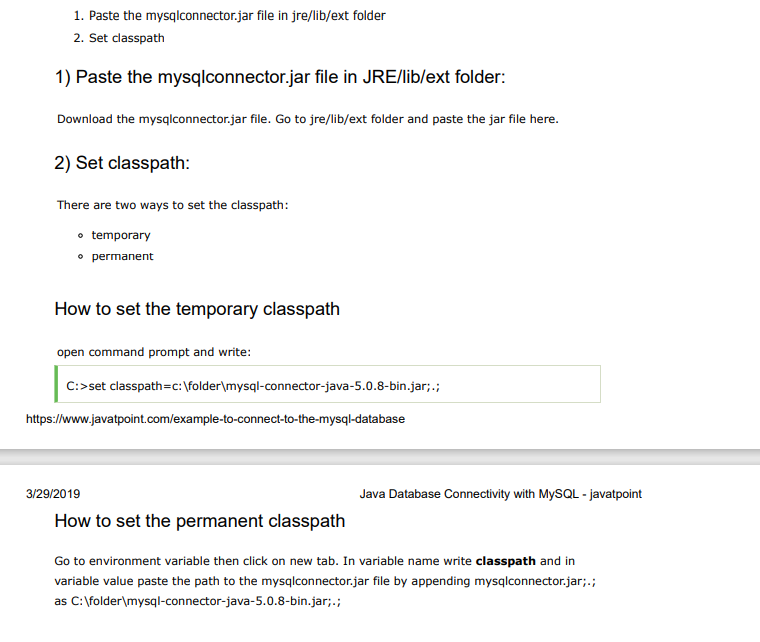
**close connection with database software**

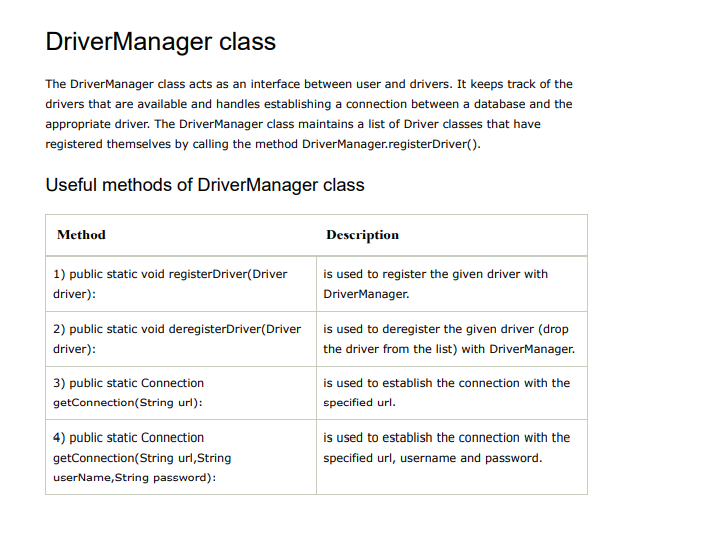
con.close()

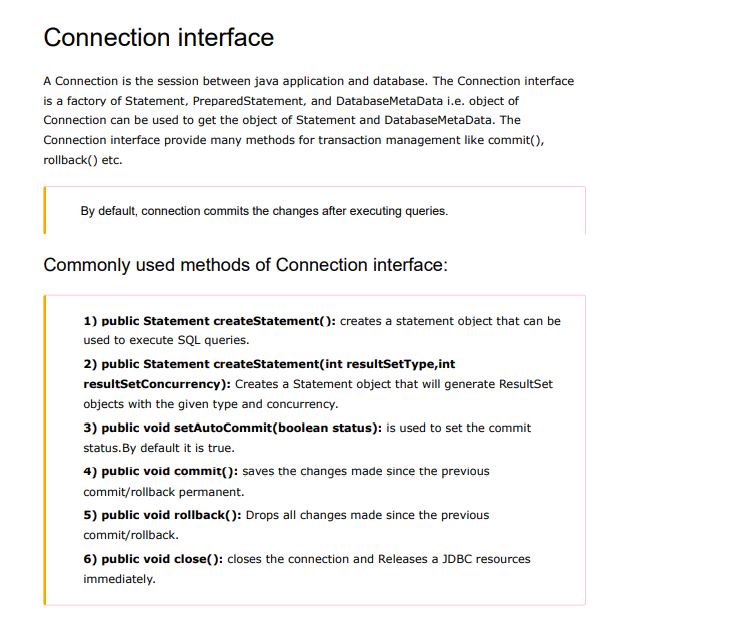


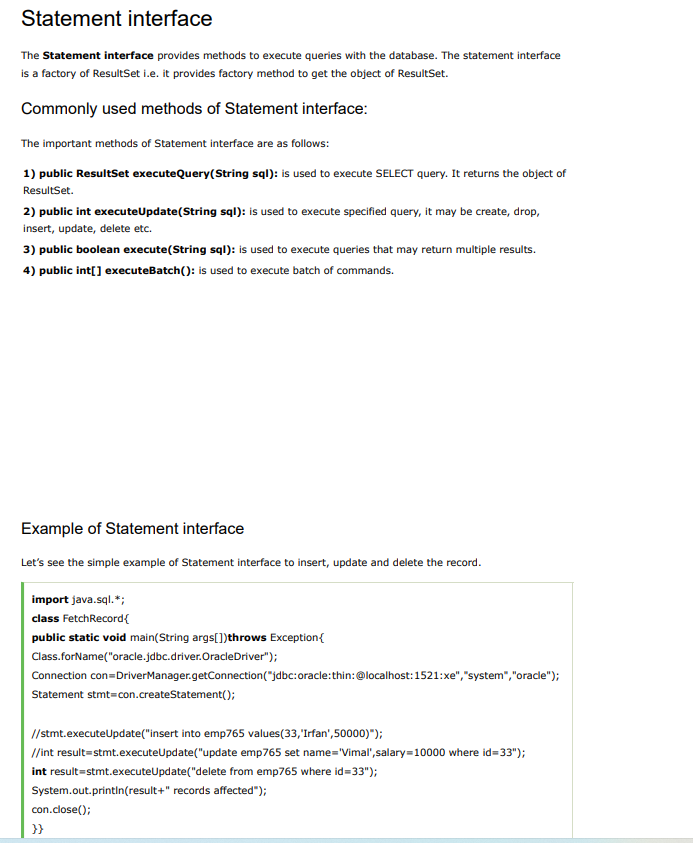


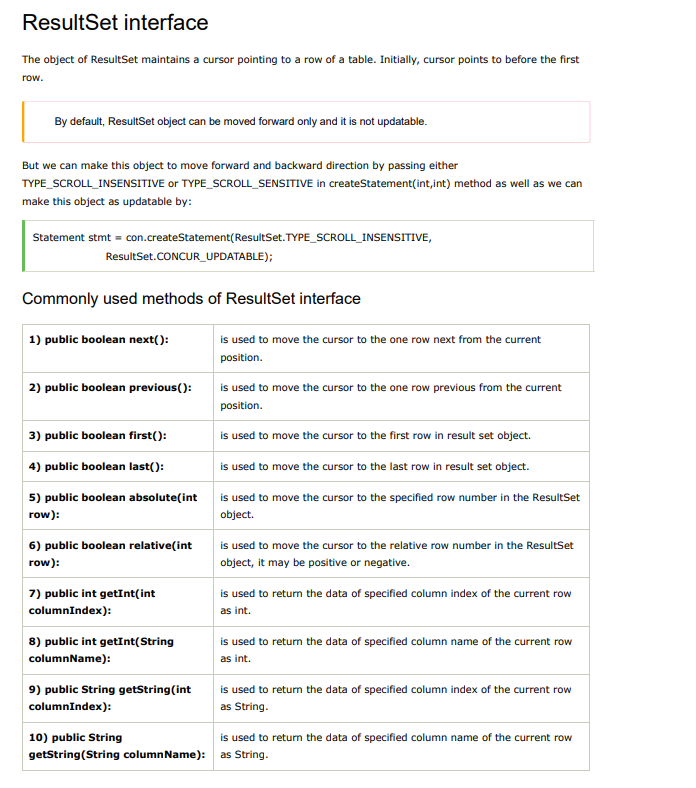
**Two ways to load the jar file**





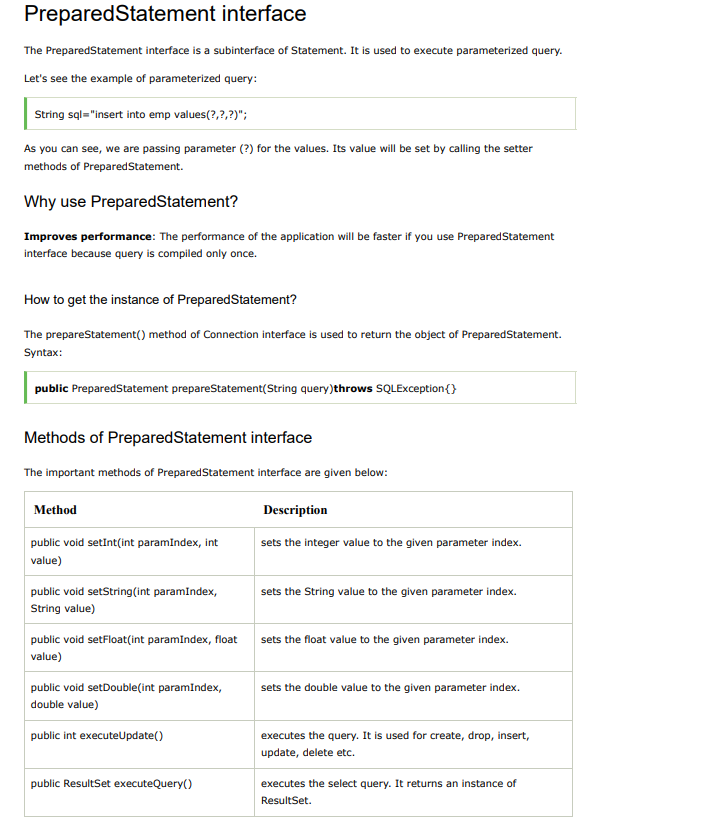


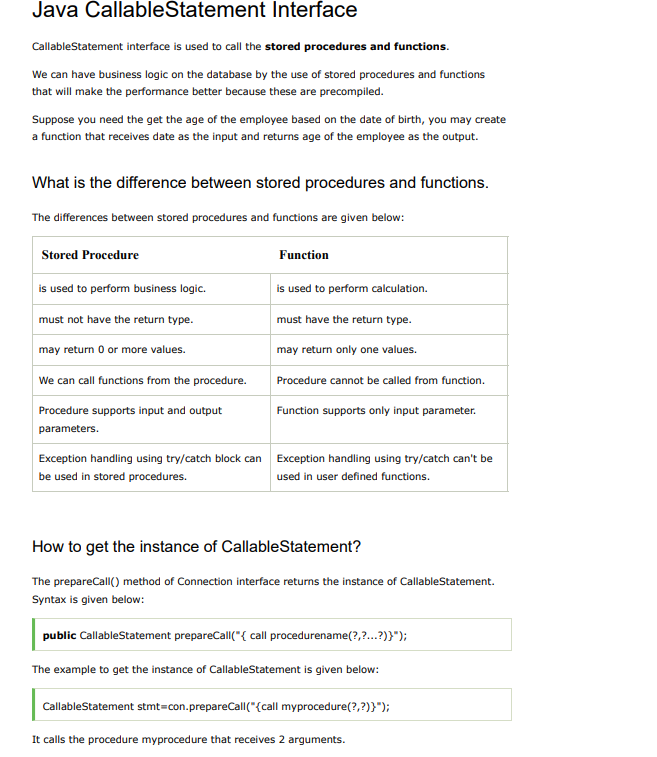




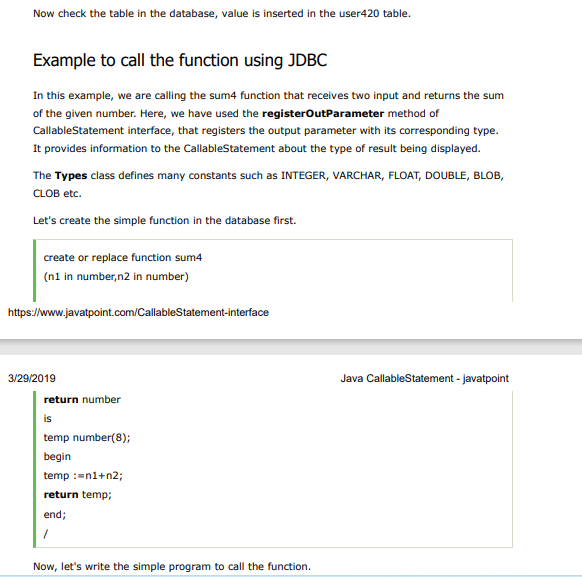
Example of Scrollable ResultSet



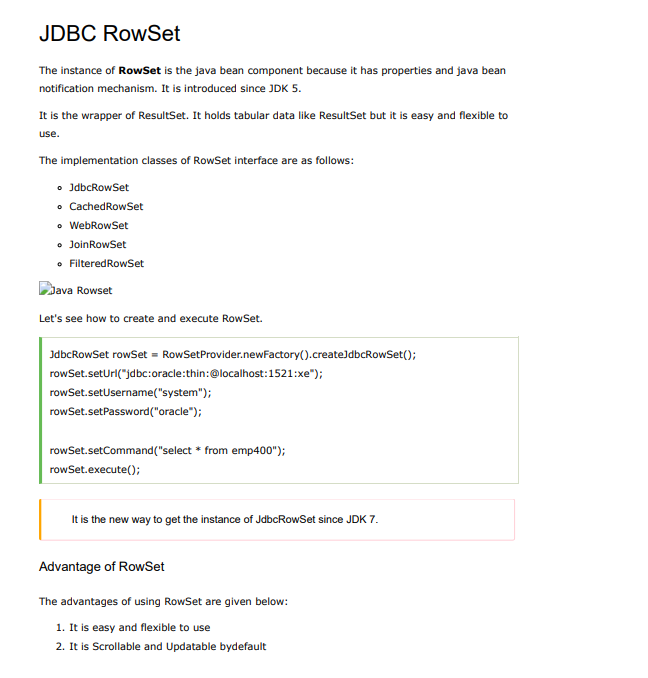




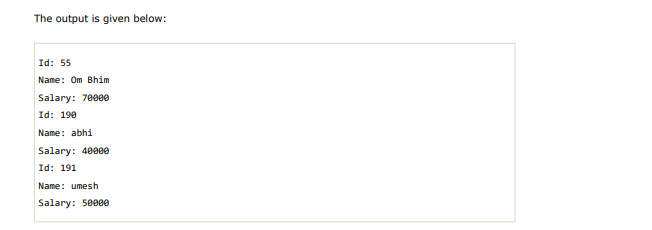


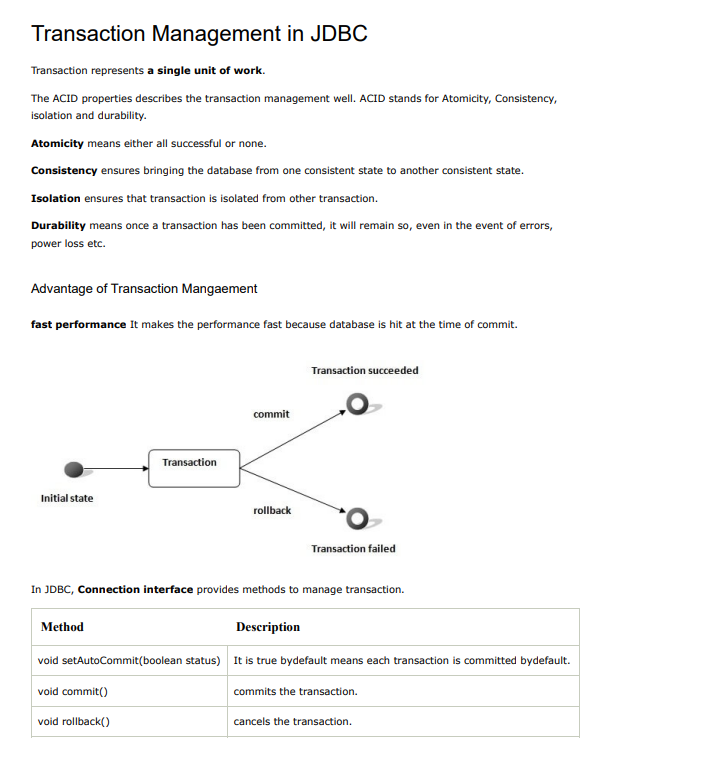


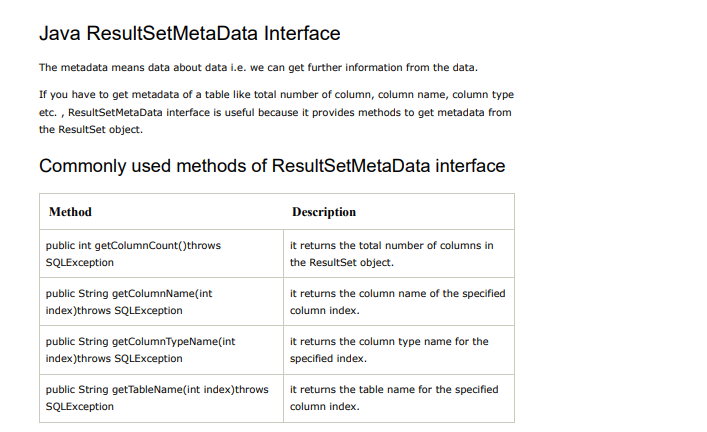














**SQL**

* SQL stands for structure query language.
* SQL used to interact with database software.
* SQL is a command based language.

**Types of SQL language**

* DDL (Data Definition Language)
* DML (Data Manipulation Language)
* DRL (Data Retreival Language)
* TCL (Transaction Control Language)
* DCL (Data Control Langauge)

**DDL**

* Create, Alter, Drop, Rename, Truncate

**Create Table**

CREATE TABLE table\_name(

colname1 datatype(size),

colname2 datatype(size),

colname3 datatype(size)

);

Table: Student

Column Name:

name: varchar - 30

address: varchar - 100

mobileno: varchar - 10

CREATE TABLE student(

name varchar(30),

address varchar(100),

mobileno varchar(10)

)

**Assignment**

**Employee Information**

name

doj

mobileno

emailid

address

designation

empoyee id

***CREATE TABLE employee(***

***name varchar(30),***

***dob varchar(10),***

***mobile\_no varchar(10),***

***email\_id varchar(20),***

***address varchar(100),***

***designation varchar(30),***

***employee\_id varchar(20)***

***);***

CREATE TABLE register(

name varchar(50),

emailid varchar(30),

password varchar(50),

mobile\_no varchar(10)

);

**DML**

* Insert, update, delete

**Insert Record**

INSERT INTO register(name,emailid,password,mobile\_no) values('Ram','ram@gmail.com','1234','9090123123');

**Delete Record**

delete from student where emailid='ram';

delete from student where emailid='ram' and name=’ram’;

delete from student where emailid='ram' or name=’ram’;

**Update Record**

update student set password='123456' where emailid='ram';

**DRL**

select \* from student;

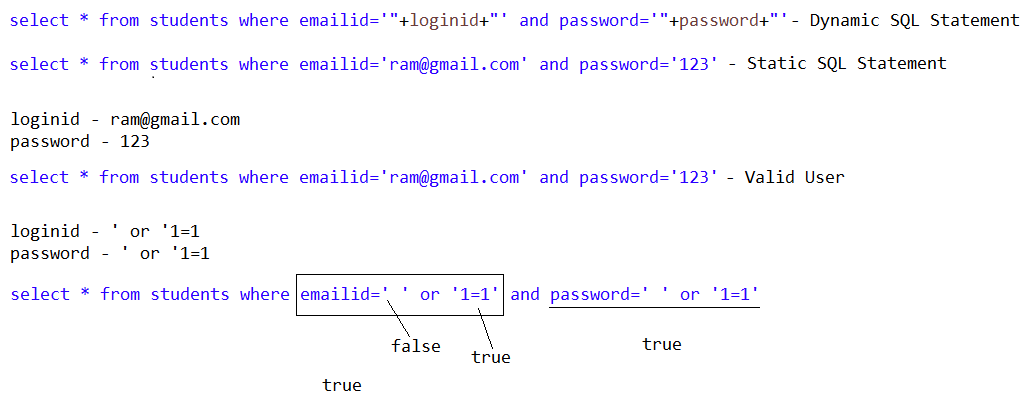
select emailid from student;

select \* from student where name='ram';

select name,mobileno from student where name='ram';

**SQL Injection**

* SQL Injection is a hacking technique.
* This technique works on dynamic SQL statements.



**Preventing SQL Injection using Prepared Statement**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** com.mysql.cj.protocol.Resultset;

**public** **class** LoginServlet **extends** HttpServlet {

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

String loginid = req.getParameter("loginid");

String password = req.getParameter("password");

**try** {

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

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb","root","root");

PreparedStatement ps = con.prepareStatement("select \* from students where emailid=? and password=?");

ps.setString(1, loginid);

ps.setString(2, password);

ResultSet rs = ps.executeQuery();

**if**(rs.next()) {

pw.println("Valid User");

}**else**

{

res.sendRedirect("login.jsp?msg=Invalid Loginid /password");

}

rs.close();

ps.close();

con.close();

}**catch**(Exception e) {

pw.println("Error: "+e);

}

}

}