

Rootpas = Admin

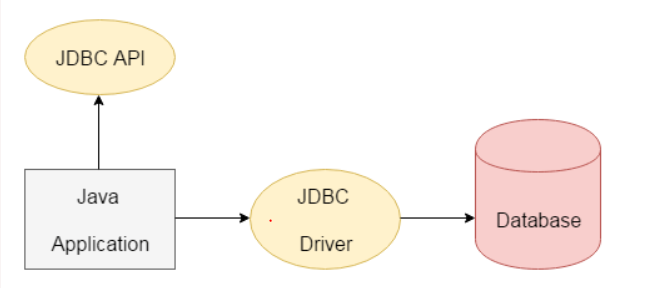
User-adithya

Pas-admin

# JDBC

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 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. A list of popular *interfaces* of JDBC API are given below:

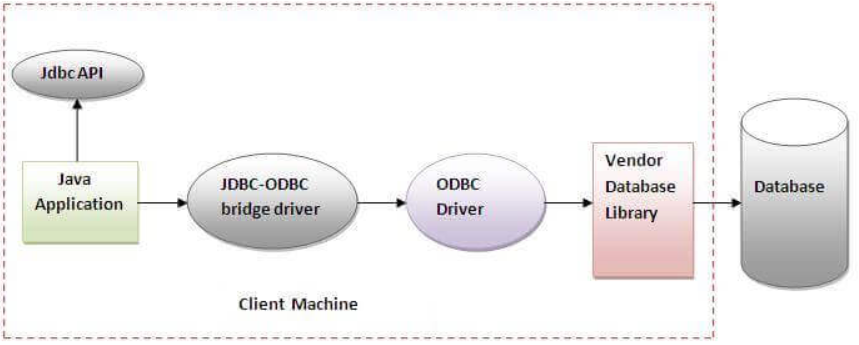
* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

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

* DriverManager class
* Blob class
* Clob class
* Types class

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:

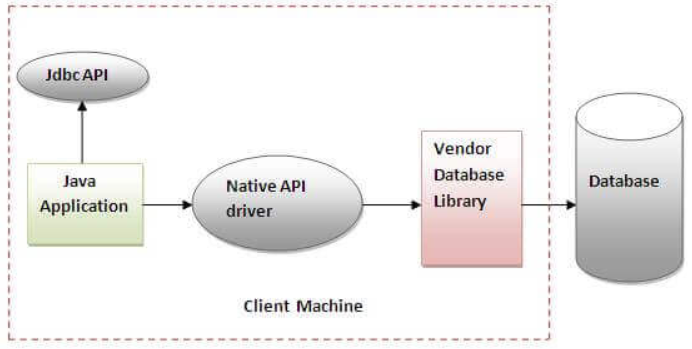
* easy to use.
* can be easily connected to any database.

### 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:

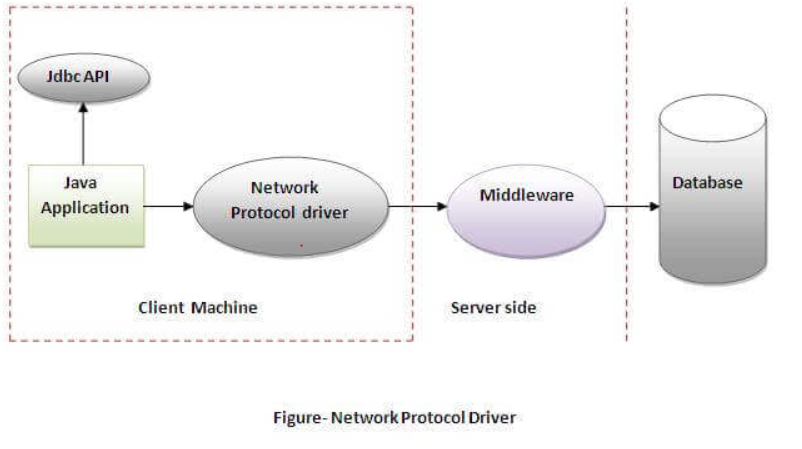
* performance upgraded than JDBC-ODBC bridge driver.

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:

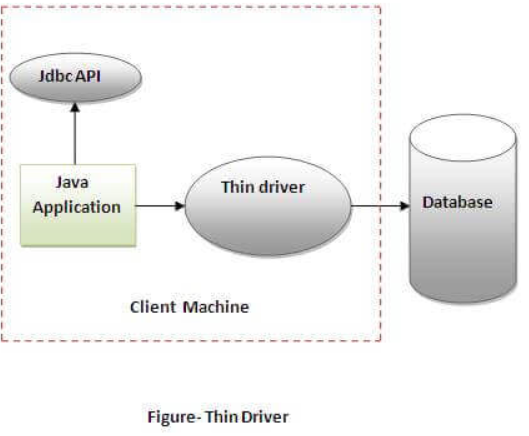
* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

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 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. |



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**

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

* Register the Driver class
* Create connection
* Create statement
* Execute queries
* Close 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

#### **Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.**

### Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

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

//Registering the Driver

      DriverManager.registerDriver(new com.mysql.jdbc.Driver());

2) Create the connection object

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

Syntax of getConnection() method

**public** **static** Connection getConnection(String url)**throws** SQLException

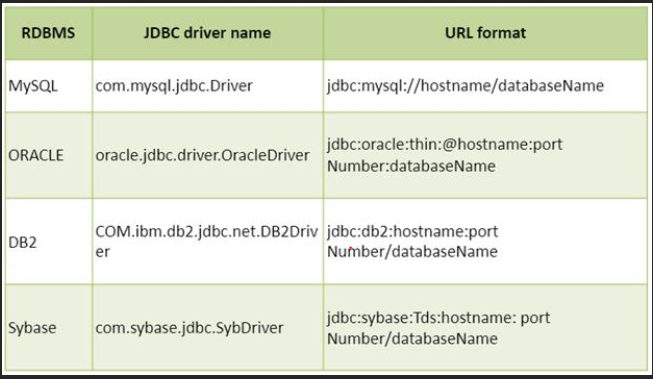
**public** **static** Connection getConnection(String url,String name,String password)

**throws** SQLException

Example to establish connection with the Oracle database

Connection con=DriverManager.getConnection(

"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

**public** Statement createStatement()**throws** SQLException

Example to create the statement object

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

**public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

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

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

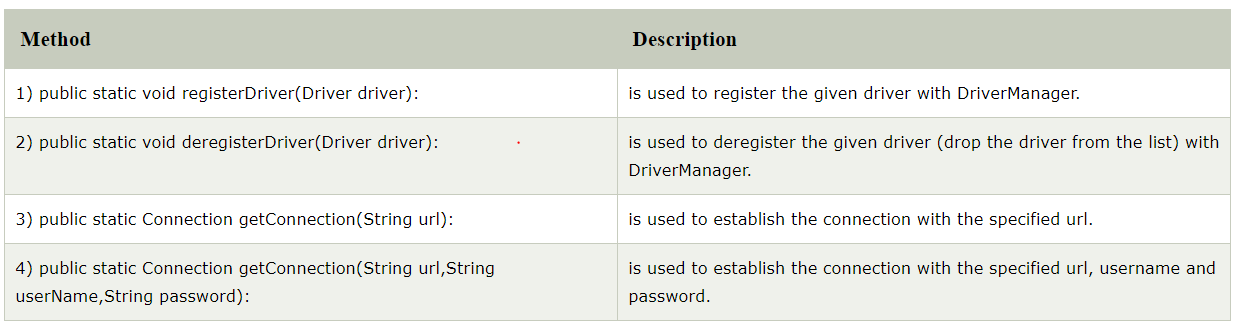
Example to close connection

con.close();

#### **Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.**

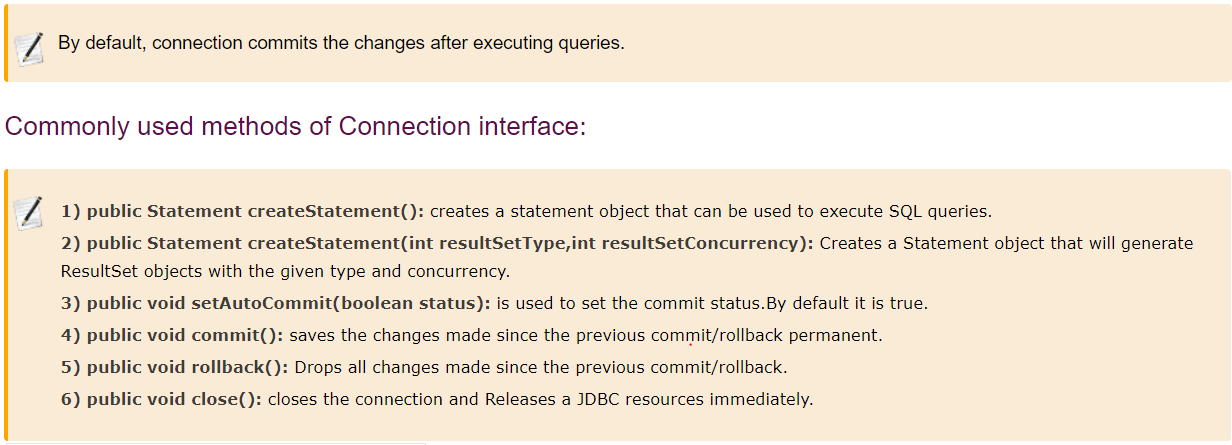
# **DriverManager class**

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()



# **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



# **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.

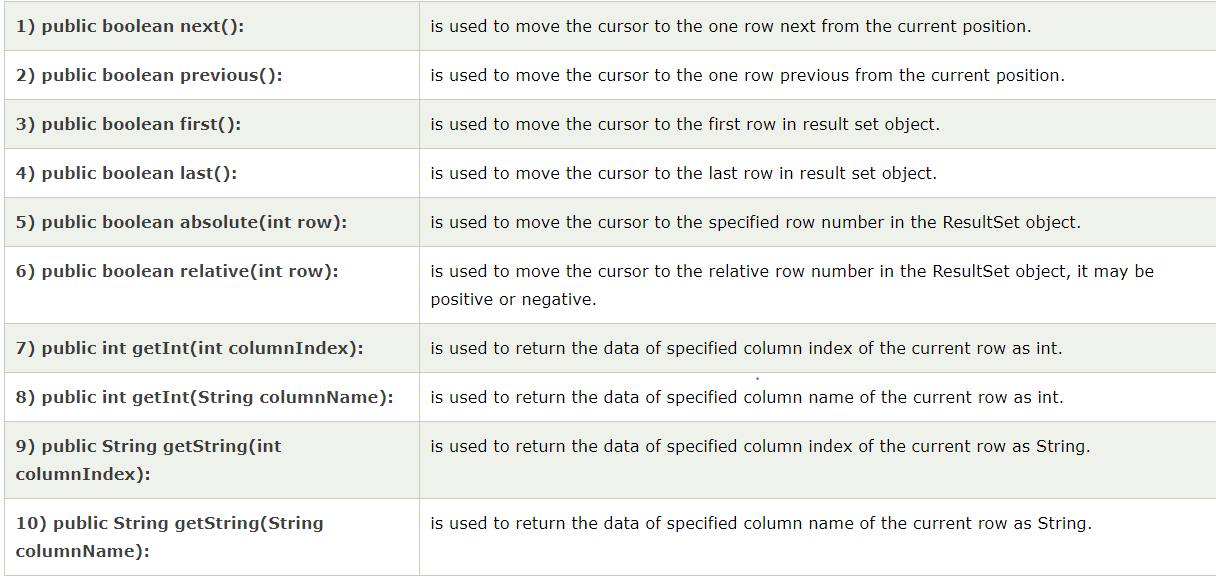
### Commonly used methods of Statement interface:

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. |

# **ResultSet interface**

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





# PreparedStatement interface

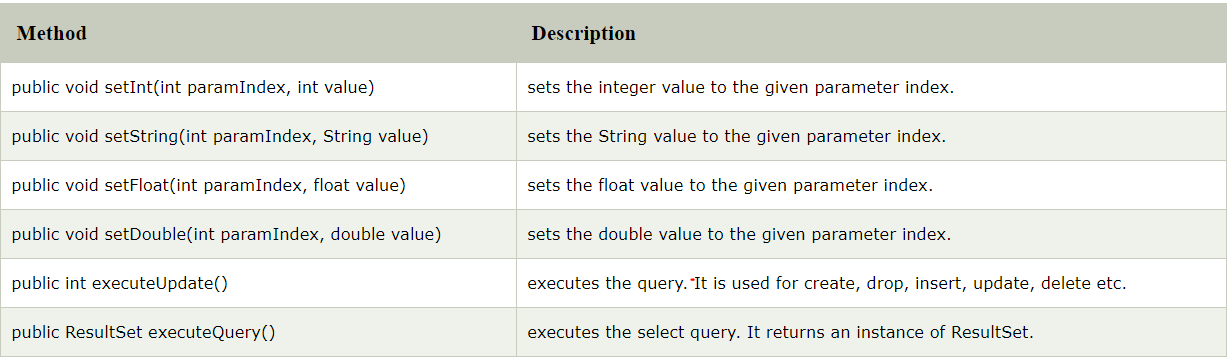
The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

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

# Why use PreparedStatement?

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



1. PreparedStatement in Java allows you to write a parameterized query which gives better performance than Statement [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html).

2. In the case of PreparedStatement, Database uses an already compiled and defined access plan, this allows prepared statement query to run faster than normal query.

3. Parametrized query written using PreparedStatement in Java prevents many common SQL Injection attacks.

4. PreparedStatement allows you to write dynamic queries in Java.

5. PreparedStatement is associated with java.sql.Connection object, once you drop a connection all PreparedStatement associated with that connection will be dropped by Database.

6. "?" is also called placeholder or IN parameter in Java.

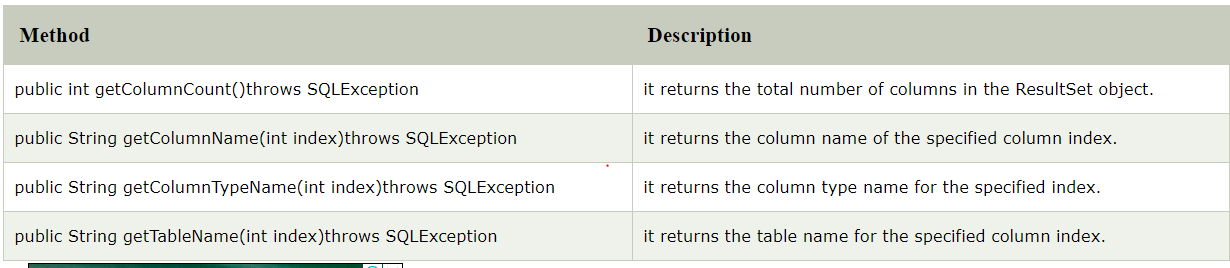
7. PreparedStatement query return FORWARD\_ONLY ResultSet, so you can only move in one direction

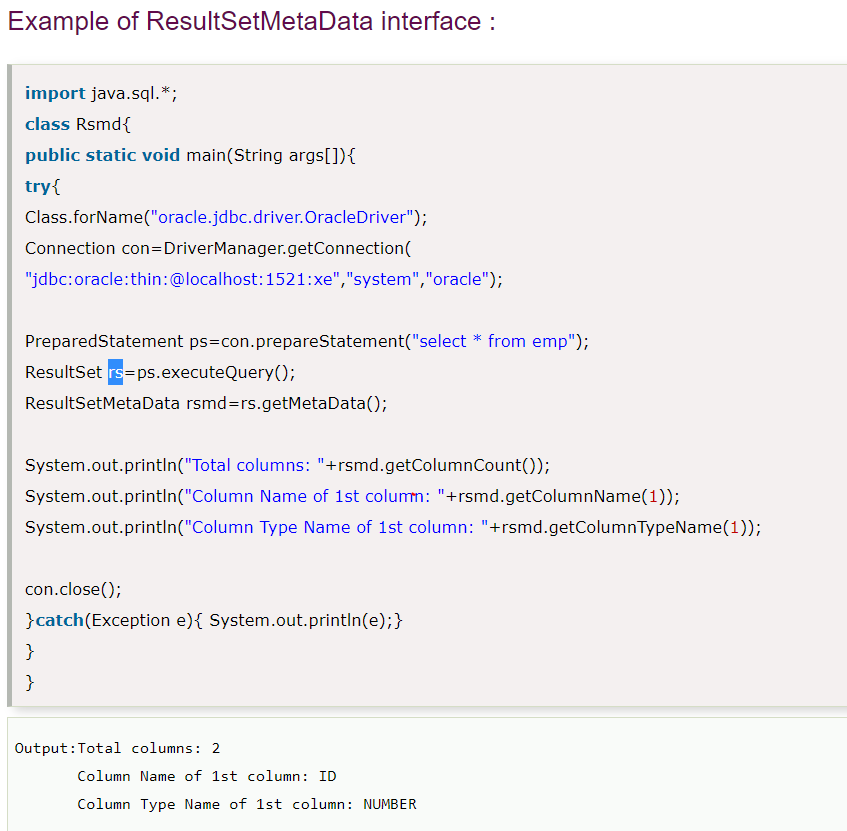


# **Java ResultSetMetaData Interface**

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



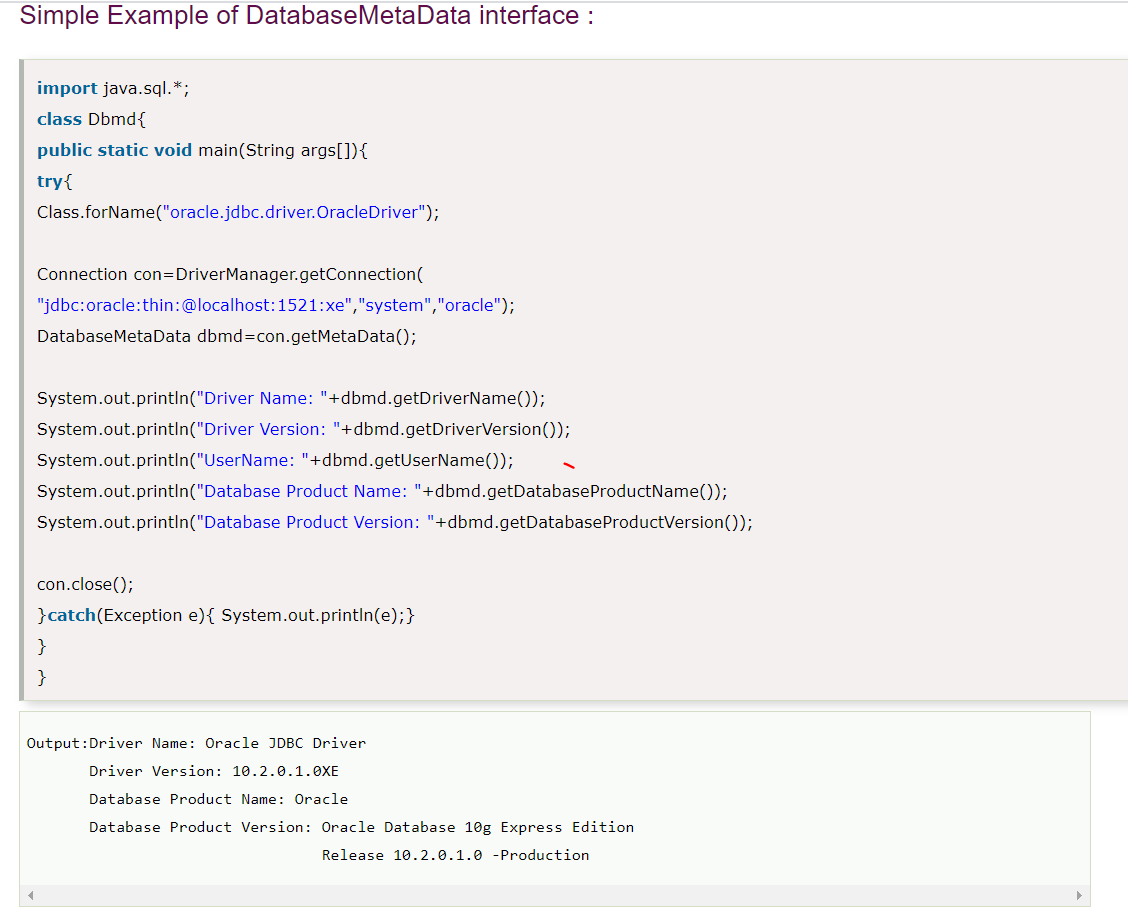


# **Java DatabaseMetaData interface**

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

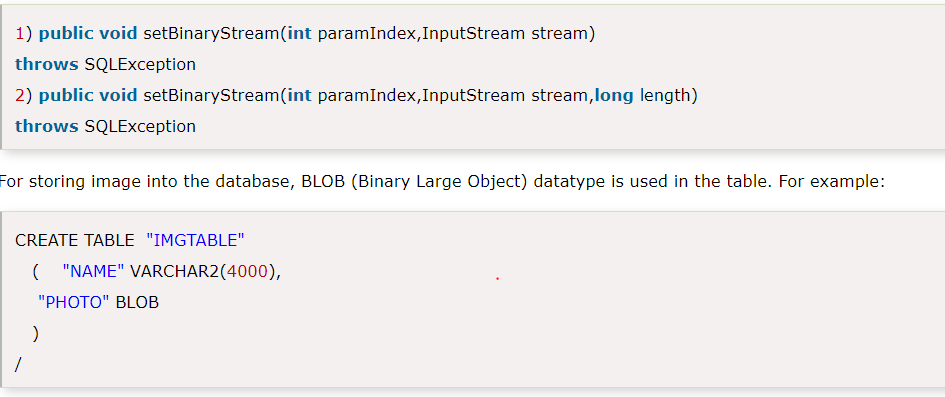
## **Commonly used methods of DatabaseMetaData interface**

* **public String getDriverName()throws SQLException:**it returns the name of the JDBC driver.
* **public String getDriverVersion()throws SQLException:**it returns the version number of the JDBC driver.
* **public String getUserName()throws SQLException:**it returns the username of the database.
* **public String getDatabaseProductName()throws SQLException:**it returns the product name of the database.
* **public String getDatabaseProductVersion()throws SQLException:**it returns the product version of the database.
* **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:**it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.



You can store images in the database in java by the help of **PreparedStatement** interface.

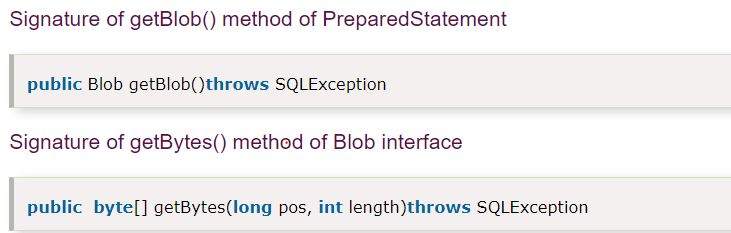
The **setBinaryStream()** method of PreparedStatement is used to set Binary information into the parameterIndex





By the help of **PreparedStatement** we can retrieve and store the image in the database.

The **getBlob()** method of PreparedStatement is used to get Binary information, it returns the instance of Blob. After calling the **getBytes()** method on the blob object, we can get the array of binary information that can be written into the image file.





|  |
| --- |
| The setCharacterStream() method of PreparedStatement is used to set character information into  the parameterIndex.        The getClob() method of PreparedStatement is used to get file information from the database Syntax of getClob method  1. **public** Clob getClob(**int** columnIndex){}     **PreparedStatement in Java** is one of several ways to execute SQL queries using JDBC API. Java provides Statement,  PreparedStatement and CallableStatement for executing queries. Out of these three, Statement is used for general-purpose queries,  PreparedStatement is used for executing a parametric query, and CallableStatement is used for executing Stored Procedures. **Java CallableStatement Interface** CallableStatement interface is used to call the **stored procedures and functions**.  We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.  Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output   How to get the instance of CallableStatement? The prepareCall() method of Connection interface returns the instance of CallableStatement.  Syntax is given below:   1. **public** CallableStatement prepareCall("{ call procedurename(?,?...?)}");   The example to get the instance of CallableStatement is given below:   1. CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");   It calls the procedure myprocedure that receives 2 arguments.       **Transaction Management in JDBC** Transaction represents **a single unit of work**.  The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.  **Atomicity** means either all successful or none.  **Consistency** ensures bringing the database from one consistent state to another consistent state.  **Isolation** ensures that transaction is isolated from other transaction.  **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc       **Batch Processing in JDBC** Instead of executing a single query, we can execute a batch (group) of queries. It makes the  performance fast.  The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing |

## **1. JDBC Rowset**

### 1.1. Connected Rowset

A rowset object may make a connection with a data source and maintain that connection throughout its life cycle, so it is called a connected rowset.

### 1.2. JavaBeans Properties

The RowSet interface provides a set of JavaBeans properties. This allows a RowSet instance to be configured to connect to a JDBC data source and read data from the data source: setUrl(), setUserName(), setDataSourceName(), setQueryTimeOut(), setReadOnly(), setTransactionIsolation(), setCommand(), … and corresponding getter methods.

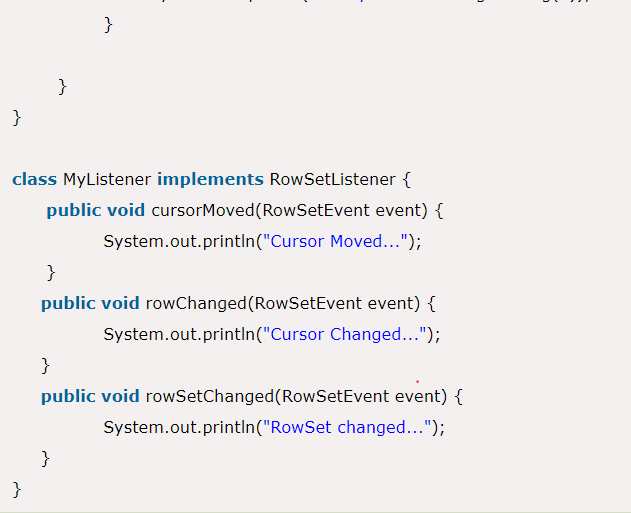
A group of setter methods (setInt(), setByte(), setString(), …) provide a way to pass input parameters to a rowset’s command property.

For example:

|  |  |
| --- | --- |
|  | JdbcRowSetImpl jrs = new JdbcRowSetImpl();  jrs.setCommand("SELECT \* FROM books WHERE author = ?");  jrs.setURL("jdbc:myDriver:myAttribute"); // set method to connect to datasource  (configure)  jrs.setUsername("myuser");  jrs.setPassword("mypwd");  jrs.setString(1, "Mark Twain"); // set method to pass input parameter  jrs.execute(); // fills this rowset object with data |







The implementation classes of RowSet interface are as follows:

* JdbcRowSet
* CachedRowSet
* WebRowSet
* JoinRowSet
* FilteredRowSet

## **CachedRowSet**

A CachedRowSet is a RowSet in which the rows are cached and the RowSet is disconnected, that is, it does not maintain an active connection to the database



# Savepoint

Savepoint interface gives you the additional transactional control. Most modern DBMS, support save points within their environments such as Oracle's PL/SQL.

When you set a save point you define a logical rollback point within a transaction. If an error occurs past a save point, you can use the rollback method to undo either all the changes or only the changes made after the save point.

The Connection object has two new methods that help you manage save points −

* **setSavepoint(String savepointName):** Defines a new save point. It also returns a Savepoint object.
* **releaseSavepoint(Savepoint savepointName):** Deletes a Savepoint. Notice that it requires a Savepoint object as a parameter. This object is usually a save point generated by the setSavepoint() method.

There is one **rollback (String savepointName)** method, which rolls back work to the specified save point.

