JDBC

Shristi Technology Labs

Contents

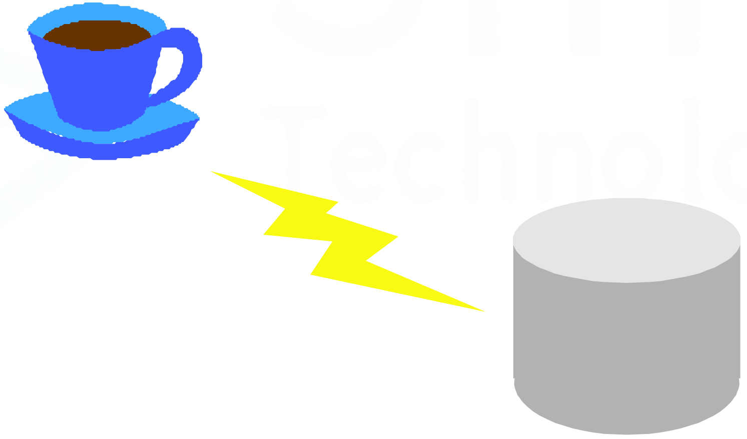
- Introduction
- JDBC API
- JDBC Architecture
- Steps to connect to a database
- Prepared Statement
- ResultsetMetadata
- DatabaseMetadata

Introduction

- JDBC is a standard interface for connecting to relational databases from Java

- The JDBC classes and interfaces are in the `java.sql` package

- The `java.sql` package contains a set of interfaces that specify the JDBC API.



Using JDBC

JDBC helps to write code that :

- Connects to one or more data servers
- Executes any SQL statement
- Obtains a result set, to navigate through query results
- Obtains metadata from the data server

JDBC API

JDBC Architecture consists of two layers:

- **JDBC API:**
  - This provides the application-to-JDBC Manager connection.
  - This is the JDBC API for applications writers.

- **JDBC Driver API:**
  - This supports the JDBC Manager-to-Driver Connection.
  - This is the lower-level JDBC driver API for driver writers.

- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases

Java.sql

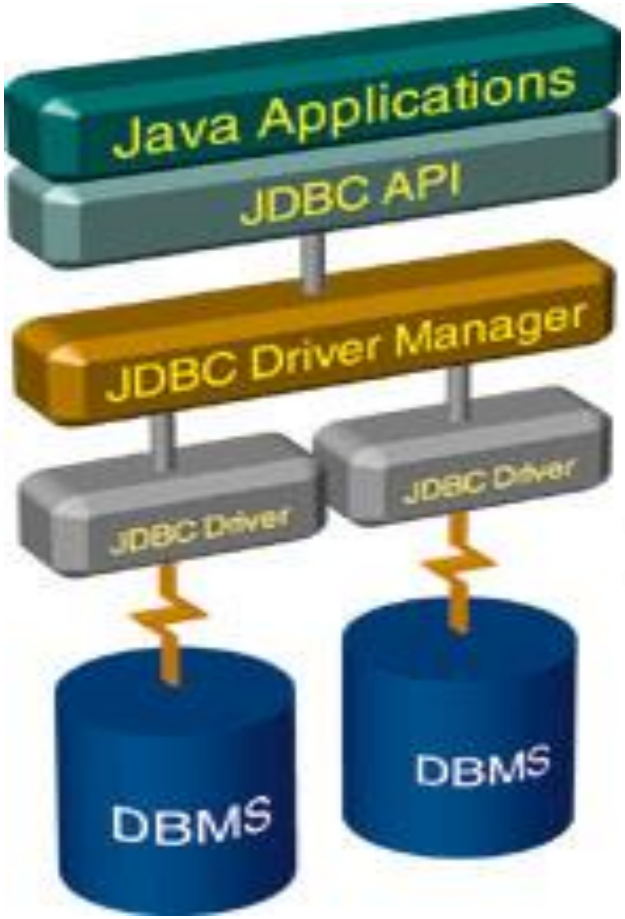
The main interfaces in the `java.sql` package

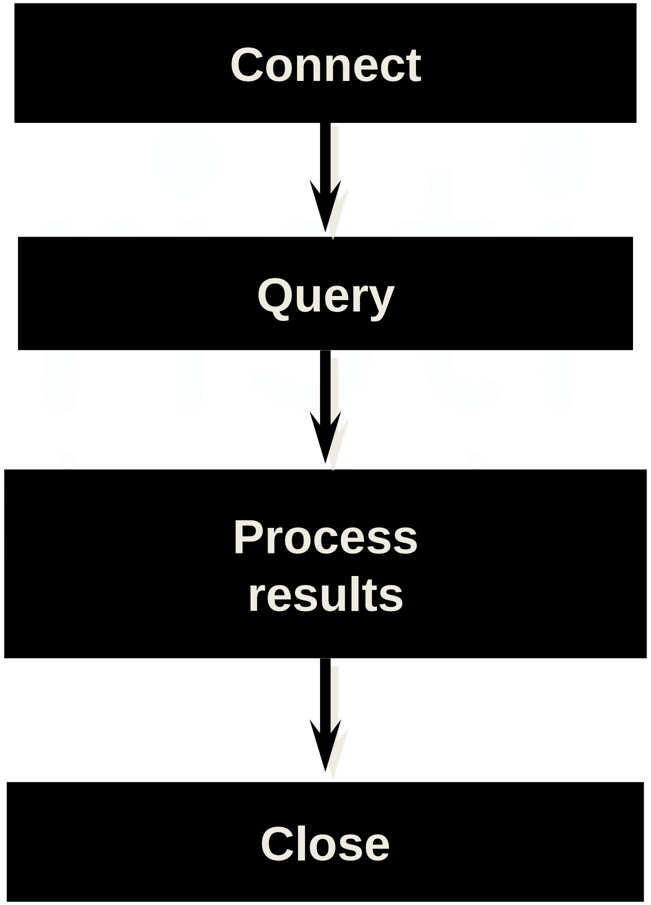
- `Connection`
- `Driver`
- `Statement`
- `ResultSet`
- `ResultSetMetadata`
- `PreparedStatement`
- `CallableStatement`
- `DatabaseMetadata`

The main class in the `java.sql` package

- `DriverManager`

JDBC ARCHITECTURE



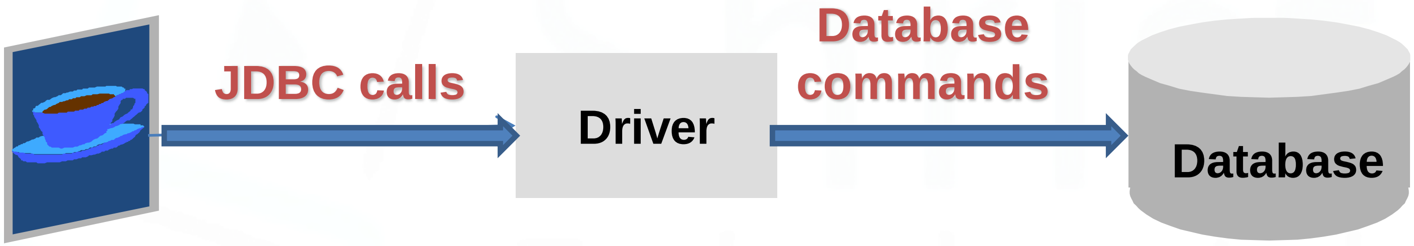


DriverManager

- keeps track of the drivers that are available
- handles establishing a connection between a database and the appropriate driver.
- attends to things like driver login time limits and the printing of log and tracing messages.

JDBC Driver

- Is an interpreter that translates JDBC method calls to vendor-specific database commands
- A database vendor or third-party developer writes a JDBC driver

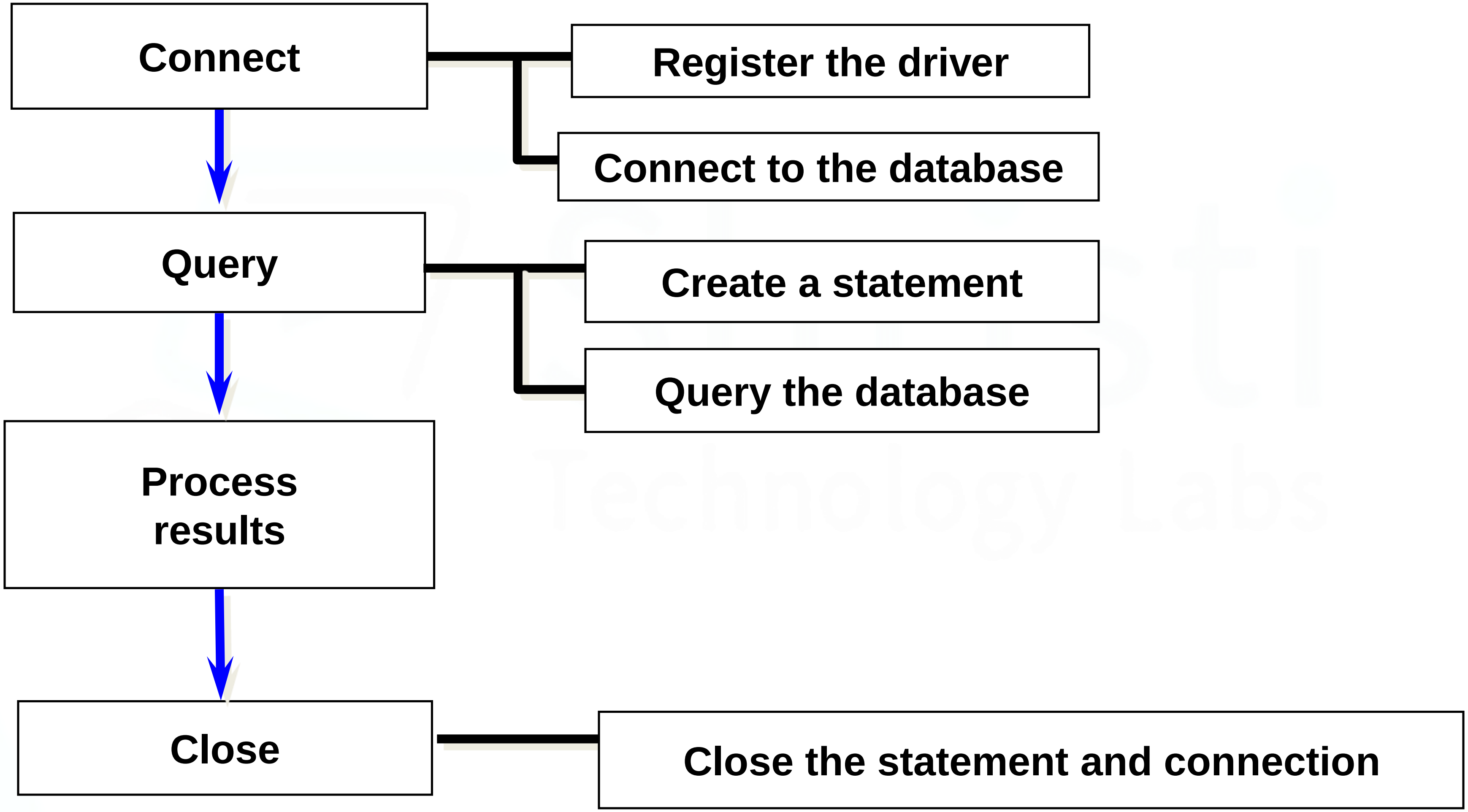


- The JDBC Driver which is a set of classes that implements these interfaces for a particular database system.
- An application can use a number of drivers interchangeably.
- JDBC drivers are available for most database platforms, from a number of vendors and in a number of different flavors.

Step to connect database

- Link and load the driver
- Establish the connection with the specified database
- Create a statement object
- Query the database

Connect



Connect

- Register the driver
- Connect to the database

Query

- Create a statement
- Query the database

Process results

Close

- Close the statement and connection

Connect

Register and load the driver

```
        Class.forName(String driver name);
```

eg:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Establish the Connection

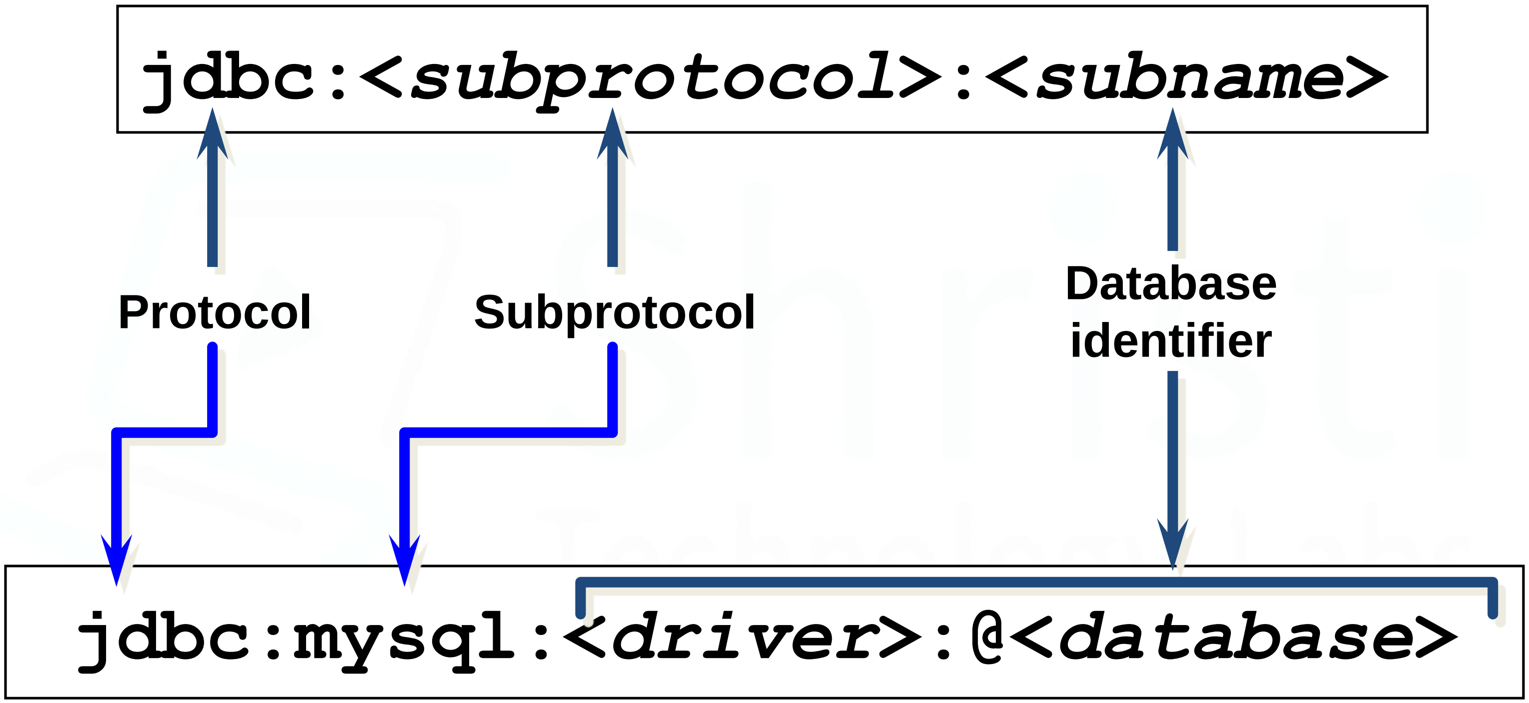
- Call the static method getConnection from **DriverManager** class

```
Connection con = DriverManager.getConnection(String url,
                      String username,String password);
```

eg:

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

JDBC url



`jdbc:<subprotocol>:<subname>`

Protocol

Subprotocol

Database identifier

`jdbc:mysql:<driver>:@<database>`

JDBC uses a URL to identify the database connection.

eg: `jdbc:mysql://localhost:3306/oradb`

Query

Create a Statement Object

- A Statement object sends the SQL statement to the database
- Create a statement object,

```
Statement st  = con.createStatement();
```

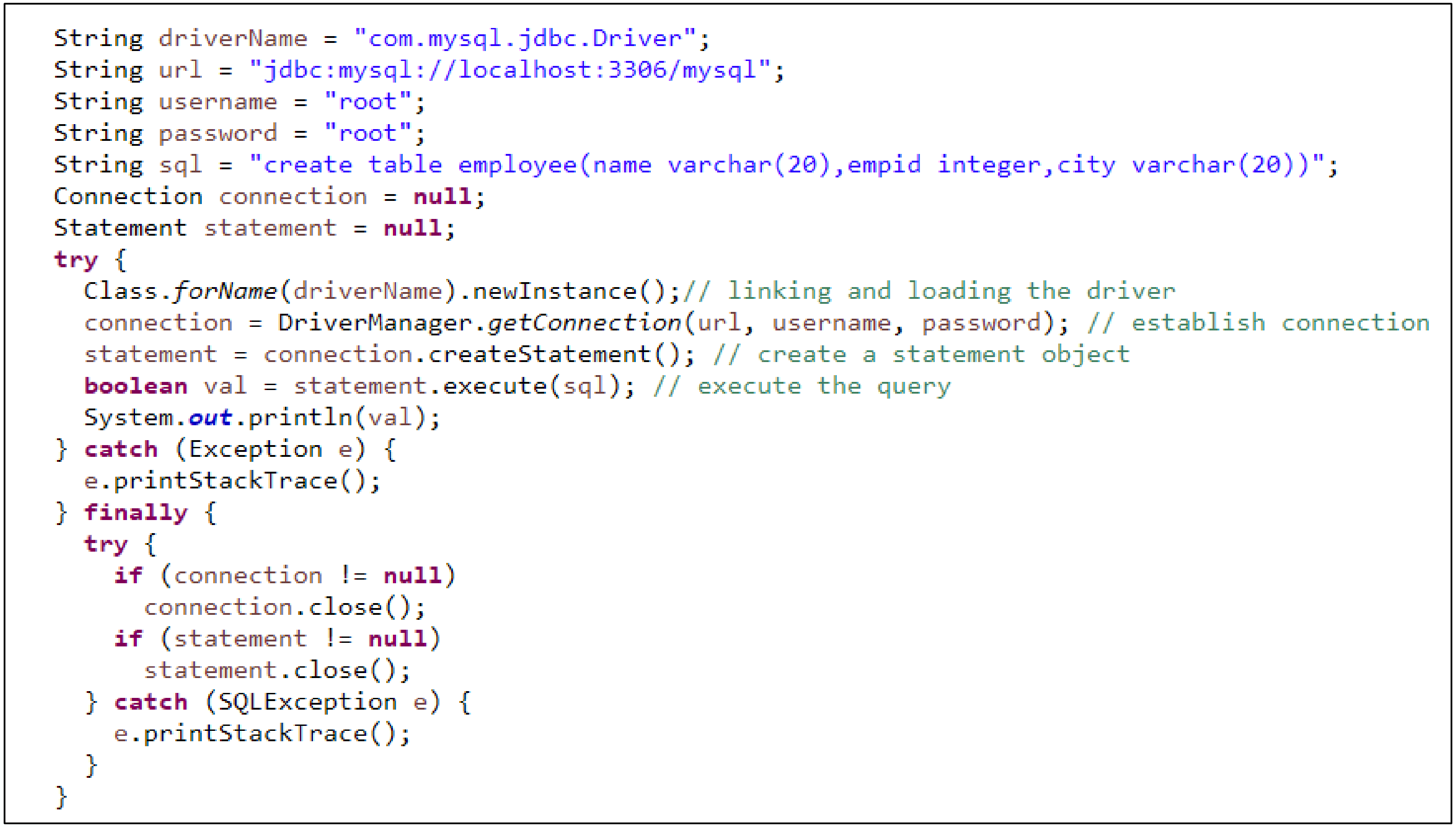
- Has as three methods to execute a SQL statement:
  - `executeUpdate()` for INSERT, UPDATE, DELETE, or DDL statements
  - `executeQuery()` for QUERY statements
  - `execute()` for either type of statement

Execute the Query

```
boolean b = st.execute(String query);
int v = st.executeUpdate(String query);
ResultSet rs = st.executeQuery(String query);
```

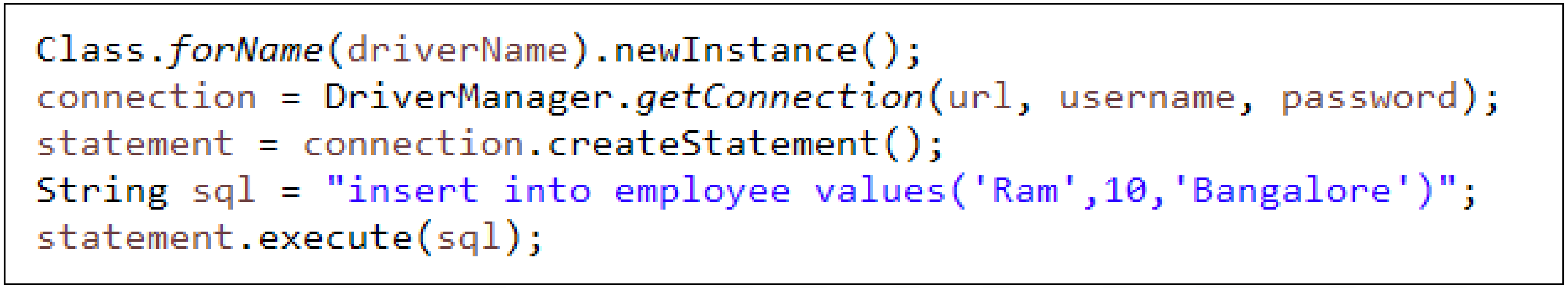
returns a resultset

Example – create table



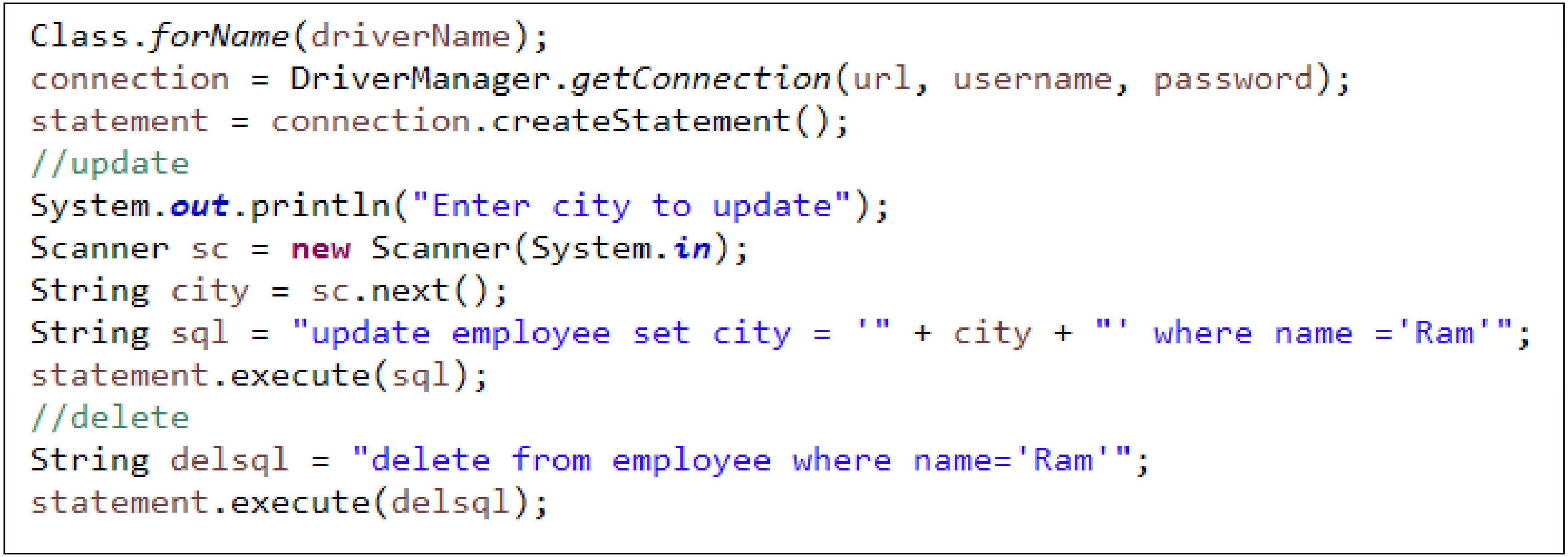
```
String driverName = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql://localhost:3306/mysql";
String username = "root";
String password = "root";
String sql = "create table employee(name varchar(20),empid integer,city varchar(20))";
Connection connection = null;
Statement statement = null;
try {
  Class.forName(driverName).newInstance();// linking and loading the driver
  connection = DriverManager.getConnection(url, username, password); // establish connection
  statement = connection.createStatement(); // create a statement object
  boolean val = statement.execute(sql); // execute the query
  System.out.println(val);
} catch (Exception e) {
  e.printStackTrace();
} finally {
  try {
    if (connection != null)
      connection.close();
    if (statement != null)
      statement.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

Example – Insert



```
Class.forName(driverName).newInstance();
connection = DriverManager.getConnection(url, username, password);
statement = connection.createStatement();
String sql = "insert into employee values('Ram',10,'Bangalore')";
statement.execute(sql);
```

Example – Update, Delete



```
Class.forName(driverName);
connection = DriverManager.getConnection(url, username, password);
statement = connection.createStatement();
//update
System.out.println("Enter city to update");
Scanner sc = new Scanner(System.in);
String city = sc.next();
String sql = "update employee set city = '" + city + "' where name ='Ram'";
statement.execute(sql);
//delete
String delsql = "delete from employee where name='Ram'";
statement.execute(delsql);
```

Process Results

- JDBC returns the results of a query in a **ResultSet** object

- A ResultSet maintains a cursor pointing to its current row of data

- The data stored in a ResultSet object is retrieved through use of get methods that allows access to the various columns of the current row.

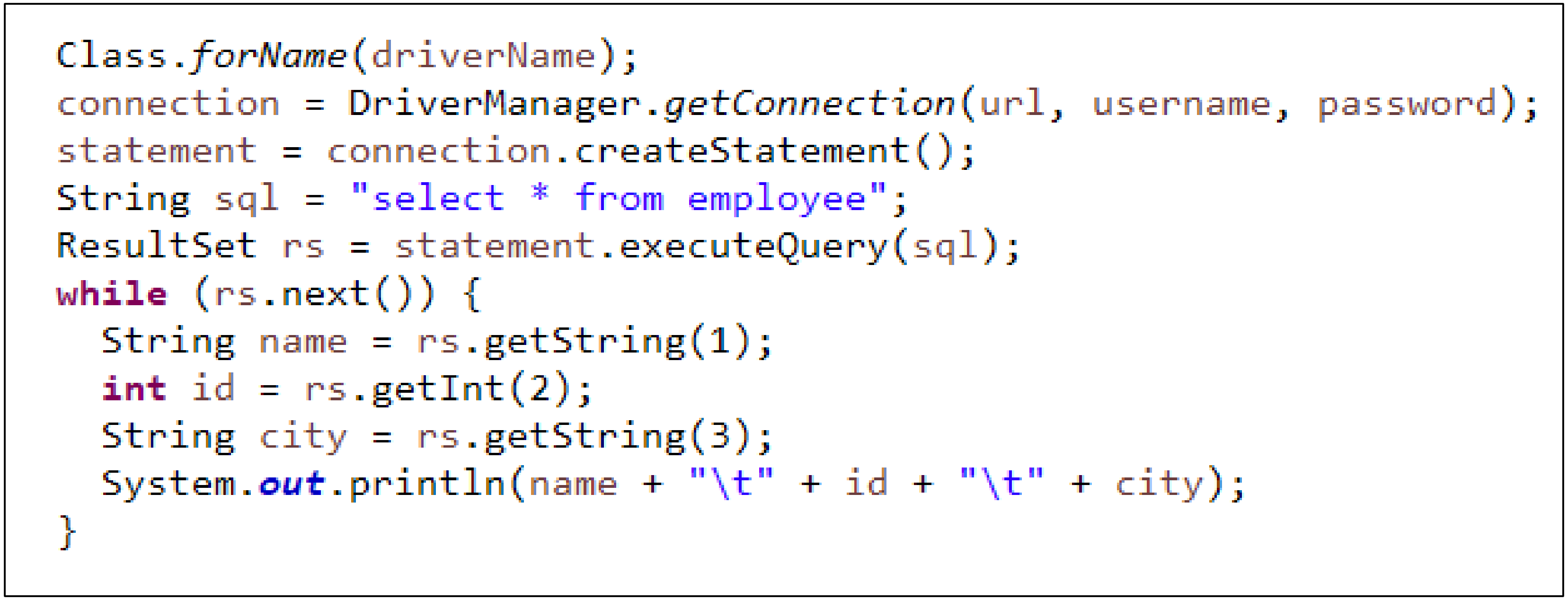
(ie) getString(), getInt()

- ResultSet.next method is used to move to the next row of the ResultSet making it the current row.

eg.

```
ResultSet rs = st.executeQuery(“select * from emp”);
While(rs.next()){
String name = rs.getString(1); //points to column name or no
}
```

Example – Retrieve



```
Class.forName(driverName);
connection = DriverManager.getConnection(url, username, password);
statement = connection.createStatement();
String sql = "select * from employee";
ResultSet rs = statement.executeQuery(sql);
while (rs.next()) {
  String name = rs.getString(1);
  int id = rs.getInt(2);
  String city = rs.getString(3);
  System.out.println(name + "\t" + id + "\t" + city);
}
```

Scrollable ResultSet

Non-Scrollable ResultSet	Scrollable ResultSet
Cursor move only in forward direction	Cursor can move both forward and backward direction
Slow performance, If we want to move nth record then we need to n+1 iteration	Fast performance, directly move on any record.
Non-Scrollable ResultSet cursor can not move randomly	Scrollable ResultSet cursor can move randomly

Methods in Scrollable ResultSet

To move the cursor in Scrollable ResultSet

- **afterLast**
  - Used to move the cursor after last row.
- **beforeFirst**
  - Used to move the cursor before first row.
- **previous**
  - Used to move the cursor backward.
- **first**
  - Used to move the cursor first at row.
- **last**
  - Used to move the cursor at last row.

Scrollable Resulttypes

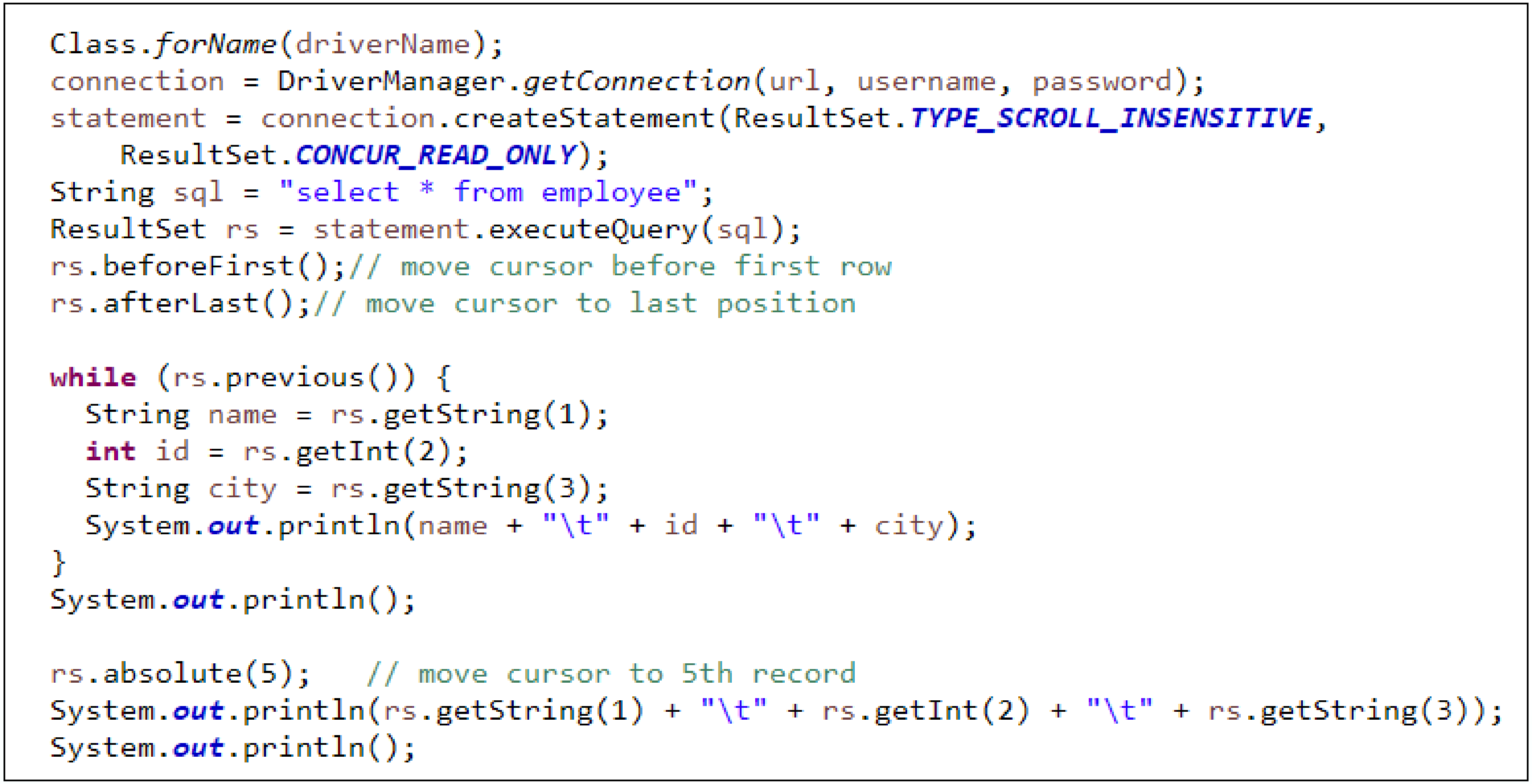
resultSetType - a result set type

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

resultSetConcurrency - a concurrency type

- ResultSet.CONCUR_READ_ONLY
- ResultSet.CONCUR_UPDATABLE

Example - ScrollableResultSet



```java
Class.forName(driverName);
connection = DriverManager.getConnection(url, username, password);
statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
String sql = "select * from employee";
ResultSet rs = statement.executeQuery(sql);
rs.beforeFirst();// move cursor before first row
rs.afterLast();// move cursor to last position

while (rs.previous()) {
  String name = rs.getString(1);
  int id = rs.getInt(2);
  String city = rs.getString(3);
  System.out.println(name + "\t" + id + "\t" + city);
}
System.out.println();

rs.absolute(5);   // move cursor to 5th record
System.out.println(rs.getString(1) + "\t" + rs.getInt(2) + "\t" + rs.getString(3));
System.out.println();
```

Close

- Close the ResultSet object
  - `rs.close();`

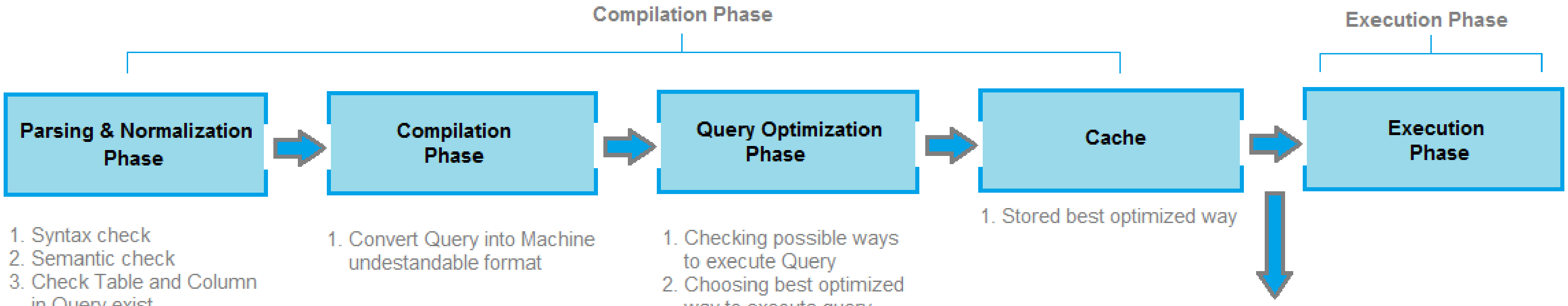
- Close the statement object
  - `st.close();`

- Close the connection object(not required for server side driver)
  - `con.close();`

PreparedStatement

- A `PreparedStatement` object holds precompiled SQL statements
- Its SQL statement is compiled only once, when you first prepare the `PreparedStatement`.
- Can supply the actual values when the st is executed
- Use this object for statements you want to execute more than once

Query Execution Phases



Compilation Phase

Execution Phase

Parsing & Normalization Phase

1. Syntax check
2. Semantic check
3. Check Table and Column in Query exist.

Compilation Phase

1. Convert Query into Machine undestandable format

Query Optimization Phase

1. Checking possible ways to execute Query
2. Choosing best optimized way to execute query.

Cache

1. Stored best optimized way

Execution Phase

After compile & Before exceution phase.
In this state, Query is called as,
Pre-Compiled Query or already Compiled Query.

Only task remaining is replacing the **?** with **data** supplied and execute query.

Data supplied for replacing with ? will be treated as pure data and Query is not goinng to compile again.

Beauty of PrepareStatement

Using PreparedStatement

Syntax

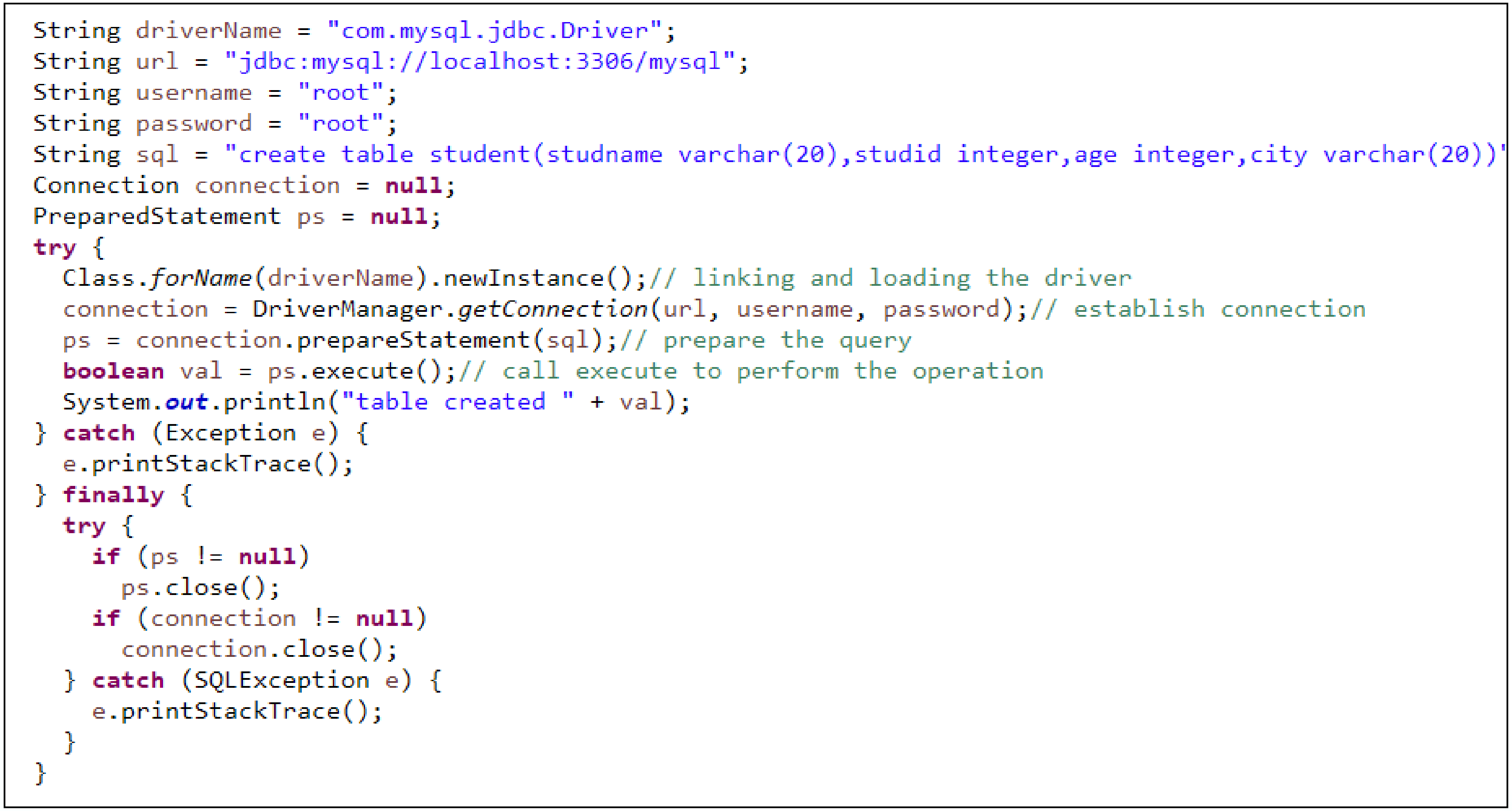
```
PreparedStatement ps = con.prepareStatement(String query);
```

where query can be

```
query = “insert into employee values (?,?,?)”;
ps.setString(1,name);
ps.setInt(2,id);
ps.setString(3,dep);
ps.execute();
```

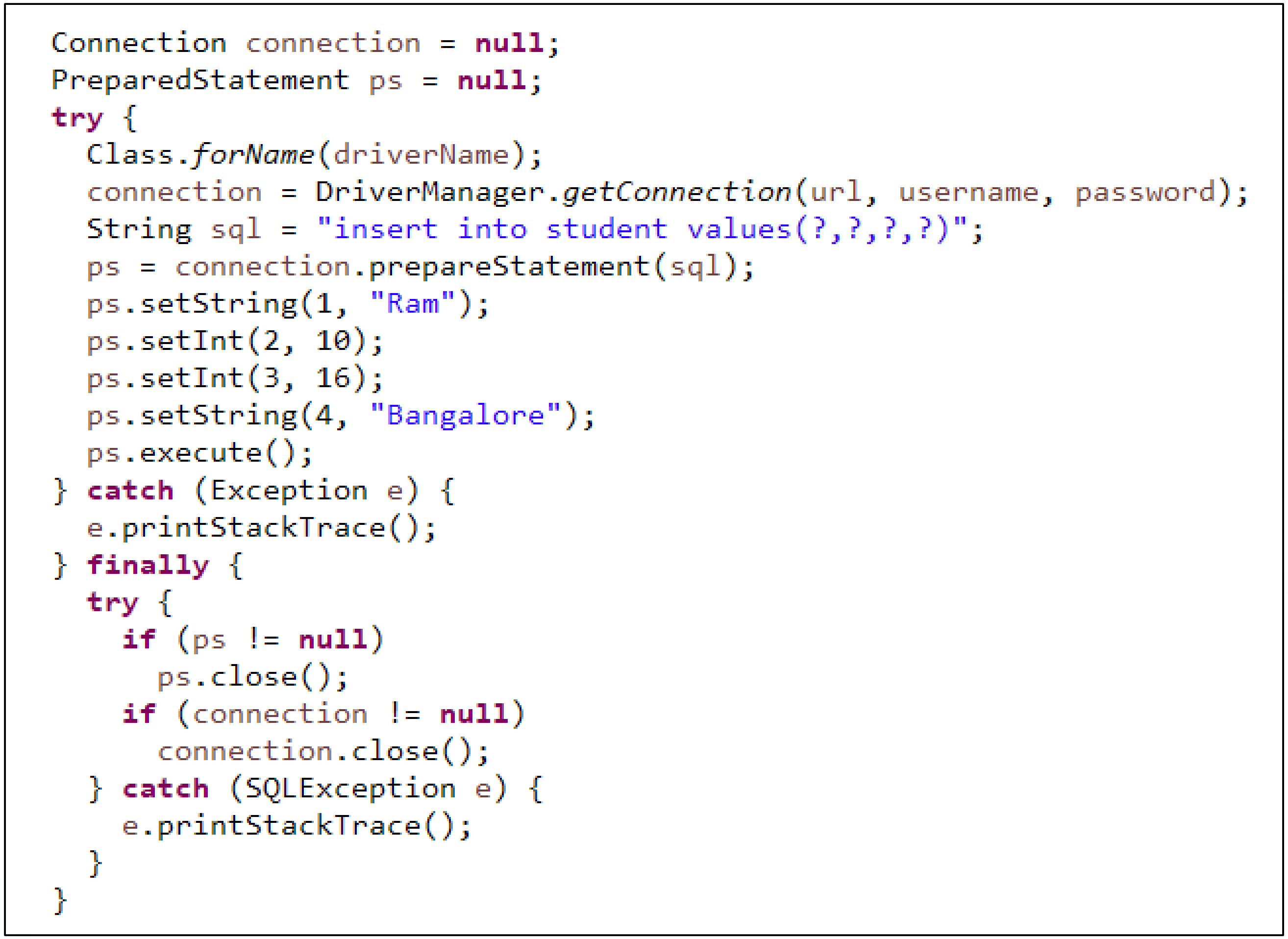
1,2,3 are placeholders for ?.
execute if not called,
st will be prepared
but not executed.

Example - Create



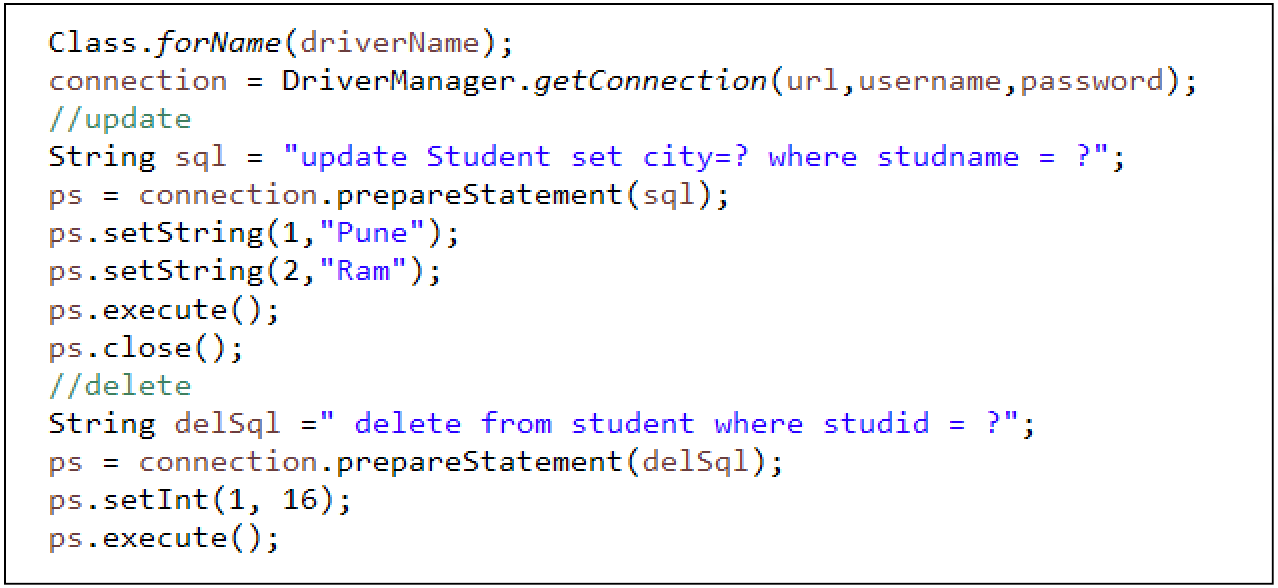
```
String driverName = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql://localhost:3306/mysql";
String username = "root";
String password = "root";
String sql = "create table student(studname varchar(20),studid integer,age integer,city varchar(20))'
Connection connection = null;
PreparedStatement ps = null;
try {
  Class.forName(driverName).newInstance();// linking and loading the driver
  connection = DriverManager.getConnection(url, username, password);// establish connection
  ps = connection.prepareStatement(sql);// prepare the query
  boolean val = ps.execute();// call execute to perform the operation
  System.out.println("table created " + val);
} catch (Exception e) {
  e.printStackTrace();
} finally {
  try {
    if (ps != null)
      ps.close();
    if (connection != null)
      connection.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

Example - Insert



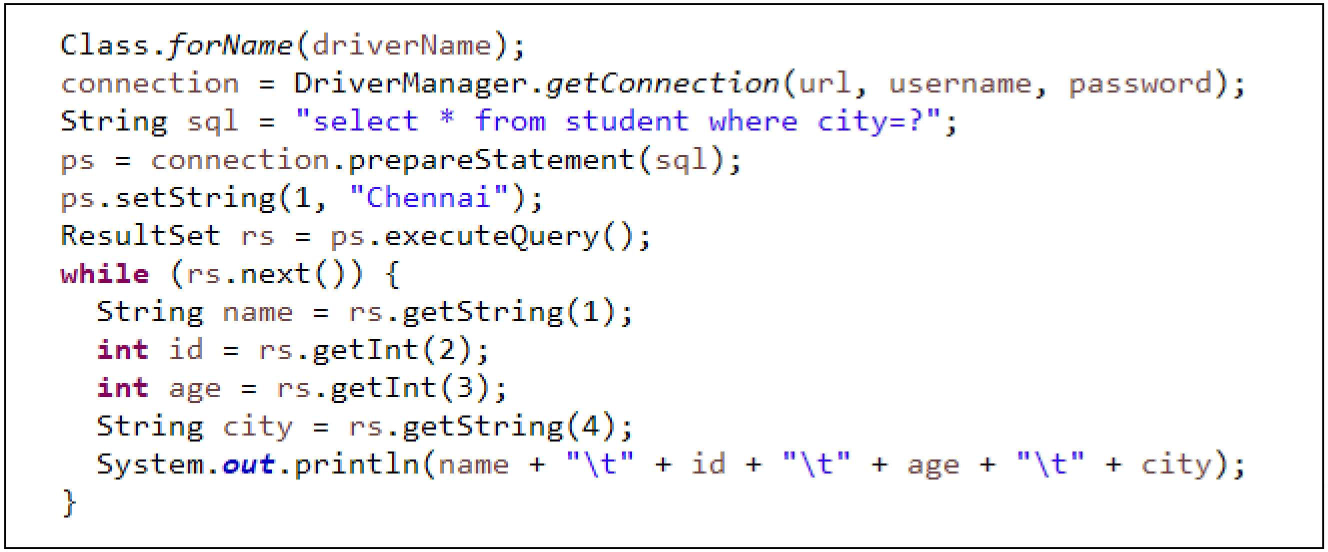
```
Connection connection = null;
PreparedStatement ps = null;
try {
  Class.forName(driverName);
  connection = DriverManager.getConnection(url, username, password);
  String sql = "insert into student values(?,?,?,?)";
  ps = connection.prepareStatement(sql);
  ps.setString(1, "Ram");
  ps.setInt(2, 10);
  ps.setInt(3, 16);
  ps.setString(4, "Bangalore");
  ps.execute();
} catch (Exception e) {
  e.printStackTrace();
} finally {
  try {
    if (ps != null)
      ps.close();
    if (connection != null)
      connection.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

Example – Update, Delete



```
Class.forName(driverName);
connection = DriverManager.getConnection(url,username,password);
//update
String sql = "update Student set city=? where studname = ?";
ps = connection.prepareStatement(sql);
ps.setString(1,"Pune");
ps.setString(2,"Ram");
ps.execute();
ps.close();
//delete
String delSql =" delete from student where studid = ?";
ps = connection.prepareStatement(delSql);
ps.setInt(1, 16);
ps.execute();
```

Example - Retrieve



```
Class.forName(driverName);
connection = DriverManager.getConnection(url, username, password);
String sql = "select * from student where city=?";
ps = connection.prepareStatement(sql);
ps.setString(1, "Chennai");
ResultSet rs = ps.executeQuery();
while (rs.next()) {
  String name = rs.getString(1);
  int id = rs.getInt(2);
  int age = rs.getInt(3);
  String city = rs.getString(4);
  System.out.println(name + "\t" + id + "\t" + age + "\t" + city);
}
```

Mapping SQL and Java Types

SQL data type	Java data type	
	Simply mappable	Object mappable
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

Thank You