Chapter 29

JDBC API

JDBC (Java Database Connectivity)

JDBC APIs that defines how a client accesses a database. you are not writing a program for a specific database.

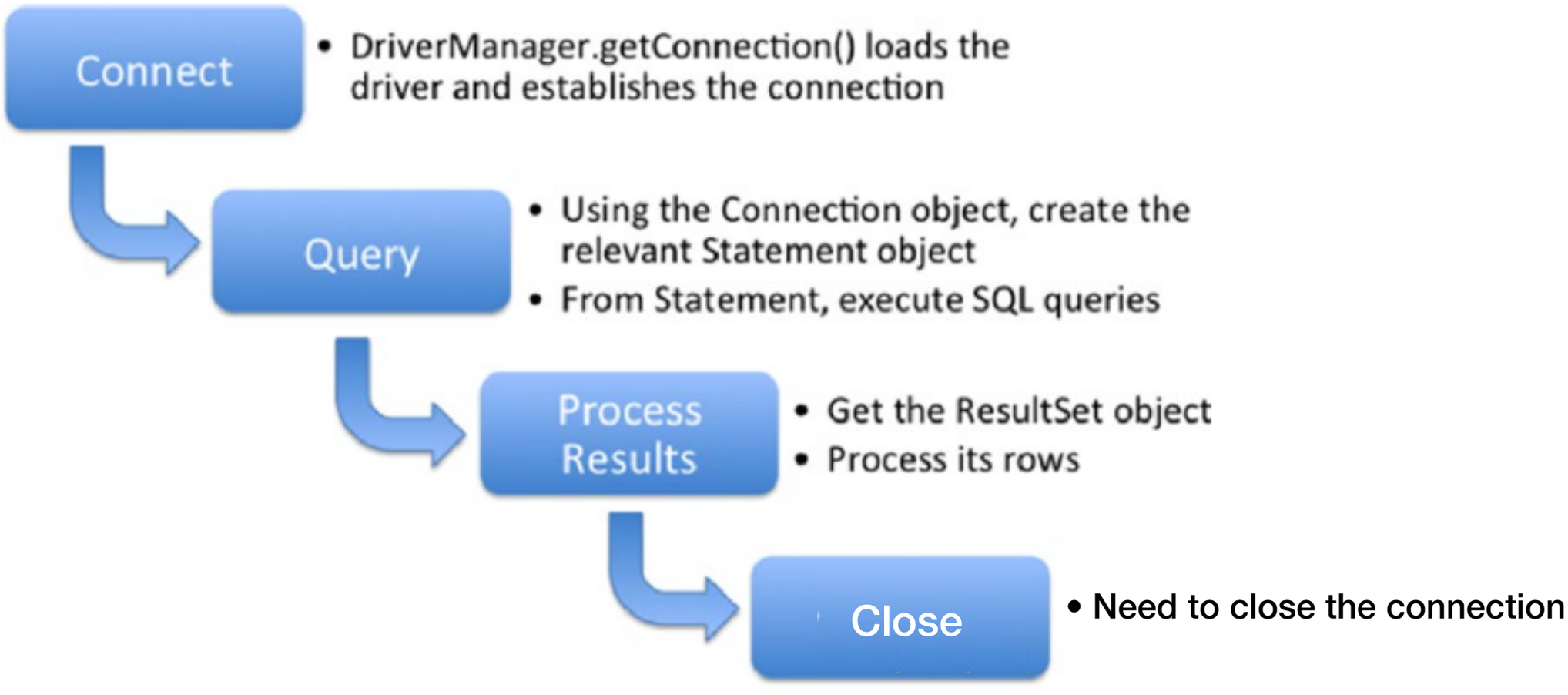
JDBC creates a loose coupling between your Java program and the type of database used.

supports only relational databases MySQL, Oracle, Microsoft SQL, and DB2.

not support NoSQL databases such as MongoDB and Neo4j.

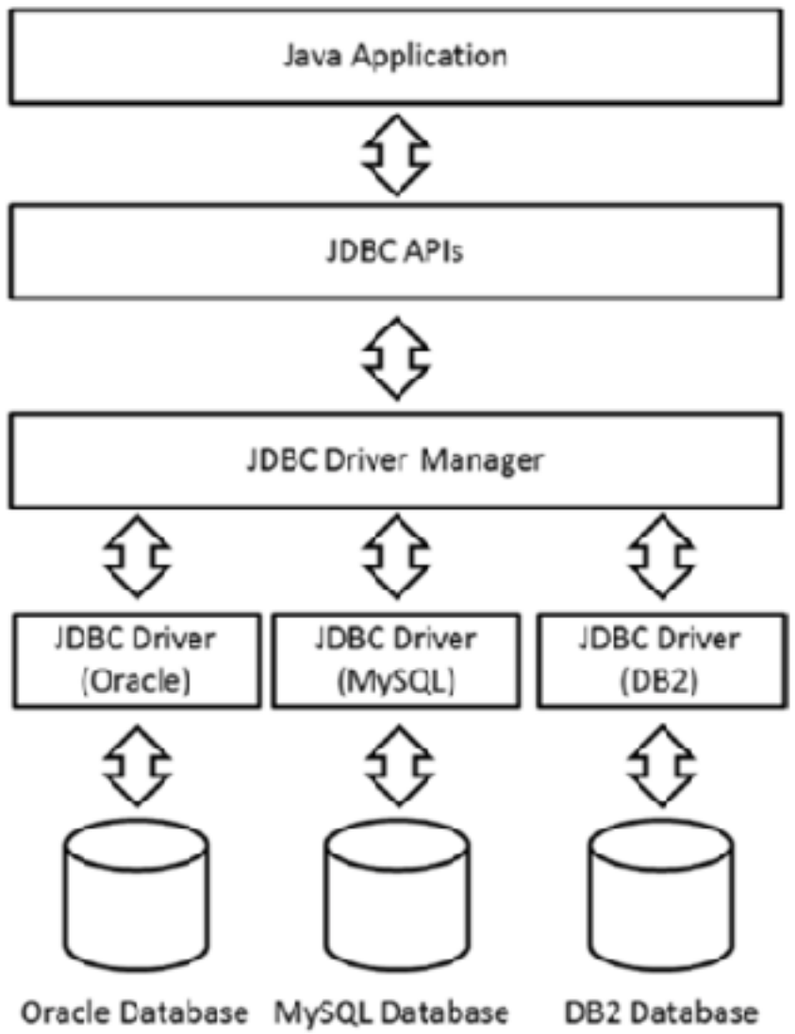
following steps:

1. Establish a connection to a database.
2. Execute SQL queries to retrieve, create, or modify tables in the database.
3. Reading Results from SQL query
4. Close the connection to the database.



Architecture

JDBC APIs interact with the JDBC driver manager to transparently connect and perform various database activities with different types of databases.



4 main interfaces have to implement

- **java.sql.Driver**
- **java.sql.Connection**
- **java.sql.Statement**
- **java.sql.ResultSet**

+ 1 class java.sql.DriverManager

1.Establish a connection to a database.

since JDBC 4.0, DriverManager automatically loads any JDBC driver in the classpath.

you can connect to a database with the `static` method `DriverManager.getConnection()`

```
Connection getConnection(String url)
Connection getConnection(String url,
                         Properties info)
Connection getConnection(String url,
                         String user,
                         String passw)
```

JDBC URL Format

jdbc:<subprotocol>:<subname>

1. Protocol (always the same)
2. Subprotocol (most of the time the name of the database/type of the driver)
3. Database specific connection properties (most of the time the location of the database with format: //SERVER:HOST/DATABASE_NAME)

more examples:

jdbc:postgresql://localhost/test

jdbc:oracle://127.0.0.1:44000/test

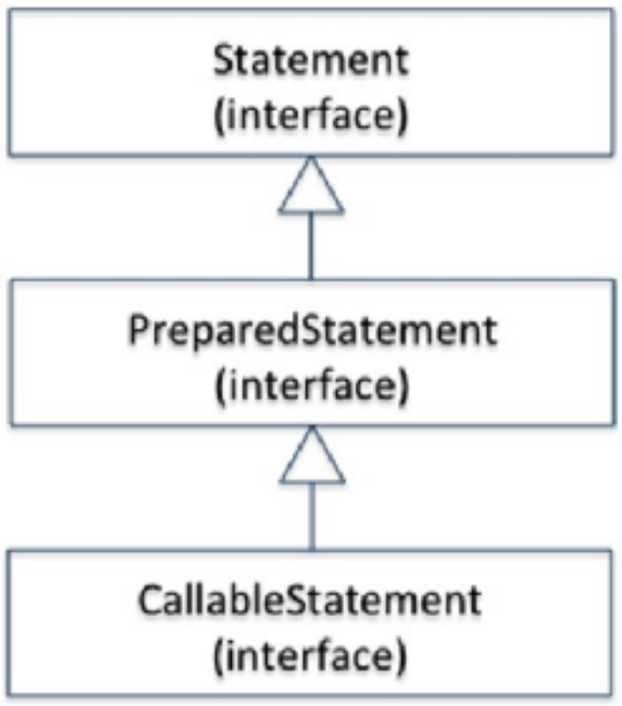
jdbc:microsoft:sqlserver://himalaya:1433 (mysql in this case. Sometimes it includes the vendor name)

```java
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/test");

Connection conn2 = DriverManager.getConnection("jdbc:mysql://localhost/test", "db_admin", "db_pwd");

Properties props = new Properties();
props.put("user", "db_admin");
props.put("password", "db_pwd");
Connection conn3 = DriverManager.getConnection("jdbc:mysql://localhost/db", props);

String url = "jdbc:mysql://localhost:3306/";
Driver driver = DriverManager.getDriver(url);
System.out.println(driver.getClass().getName()); // prints com.mysql.jdbc.Driver

Connection conn4 = driver.connect(url, props);
```

2.Executing Queries and Updating the Database

Statement

You a need a Statement object to execute queries and perform database operations.



You can get a Statement from a Connection object using the createStatement()

```
Statement createStatement()
Statement createStatement(int resultSetType,
                          int resultSetConcurrency)
```

types of result sets:

- **ResultSet.TYPE_FORWARD_ONLY**
  This is the default type. You can only go once through the results and in the order they were retrieved.
- **ResultSet.TYPE_SCROLL_INSENSITIVE**
  you can go both forward and backward through the results and to a particular position in the result set.
- **ResultSet.TYPE_SCROLL_SENSITIVE**
  you can also go forward, backward and to a particular position in the result set, but you will always see the latest changes to the data while using it

ResultSet Type	Can Go Backward	See Latest Data from Database Table	Supported by Most Drivers
ResultSet.TYPE_FORWARD_ONLY	No	No	Yes
ResultSet.TYPE_SCROLL_INSENSITIVE	Yes	No	Yes
ResultSet.TYPE_SCROLL_SENSITIVE	Yes	Yes	No

concurrency modes:

- **ResultSet.CONCUR_READ_ONLY**
  This is the default mode. you can't update (using an INSERT, UPDATE, or DELETE statement) a result set.
- **ResultSet.CONCUR_UPDATABLE**
  It indicates that the result set can be updated.

* If you ask for a CONCUR_UPDATABLE mode and your driver doesn't support it, you can get a CONCUR_READ_ONLY mode

`ResultSet` Type	Can Read Data	Can Update Data	Supported by All Drivers
`ResultSet.CONCUR_READ_ONLY`	Yes	No	Yes
`ResultSet.CONCUR_UPDATABLE`	Yes	Yes	No

The Statement interface provides three execute methods:

Method	Supported SQL statements	Return type

`execute()`	`SELECT`
	`INSERT`
	`UPDATE`
	`DELETE`
	`CREATE`
		`boolean` (`true` for `SELECT`, `false` for the rest)

`executeQuery()`	`SELECT`	`ResultSet`

`executeUpdate()`	`INSERT`
	`UPDATE`
	`DELETE`
	`CREATE`
		Number of affected rows (zero for `CREATE`)

3.Reading Results from SQL query

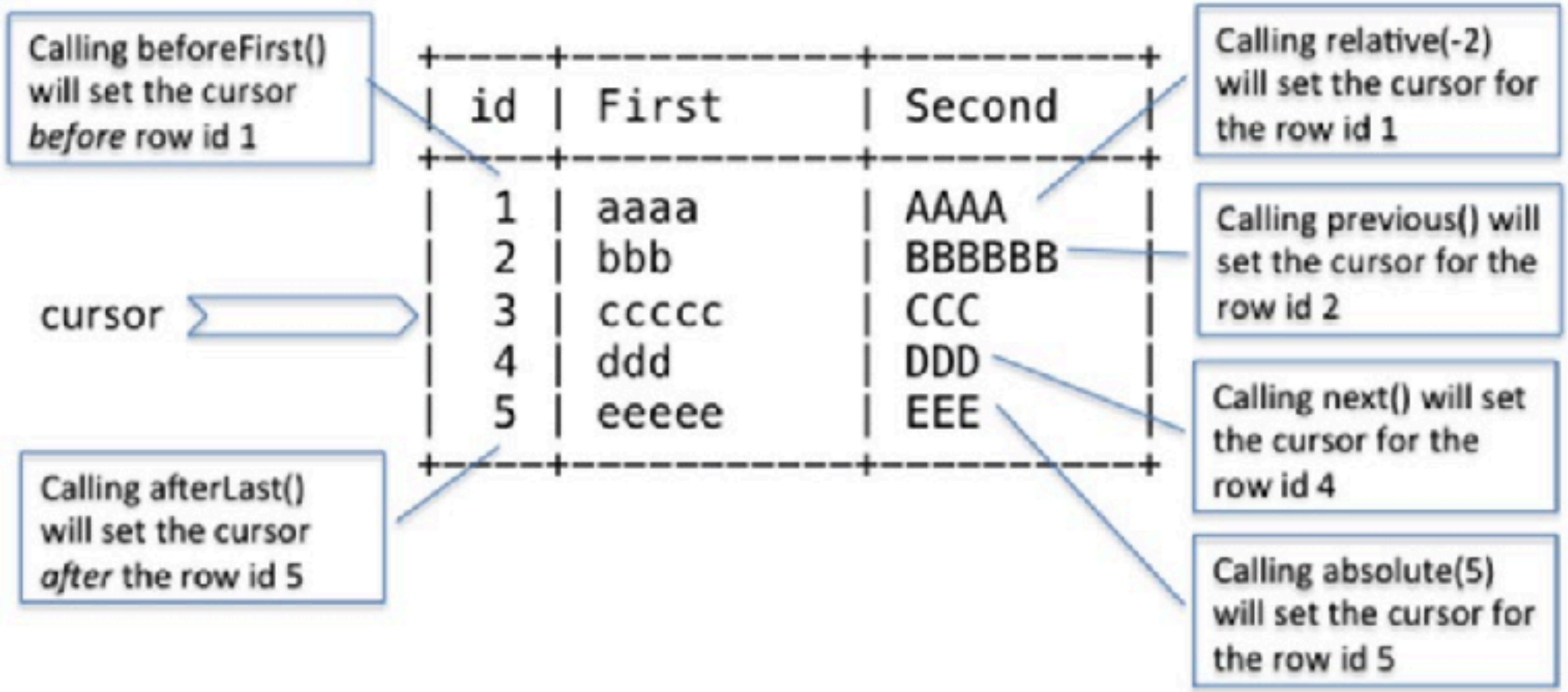
A resultset is a table with column headings and associated values requested by the query. A resultset maintains a cursor pointing to the current row.

Method	Description
`boolean absolute(int row)`	Moves the cursor to the given row number in the result set, counting from the beginning (if the argument is positive) or the end (if negative). If the argument is zero, the cursor is moved before the first row. It returns true if the cursor is moved to a valid position or false if the cursor is before the first row or after the last row.
`void afterLast()`	Moves the cursor after the last row.
`void beforeFirst()`	Moves the cursor before the first row.
`boolean first()`	Moves the cursor to the first row. It returns true if the cursor is on a valid row or false if there are no rows in the result set.
`boolean last()`	Moves the cursor to the last row. Returns true if the cursor is on a valid row or false if there are no rows in the result set.
`boolean next()`	Moves the cursor to the next row. It returns true if the new current row is valid or false if there are no more rows.
`boolean previous()`	Moves the cursor to the previous row. It returns true if the new current row is valid or false if the cursor is before the first row.
`boolean relative(int rows)`	Moves the cursor a relative number of rows, either positive or negative. Moving beyond the first/last row in the result set positions the cursor before/after the first/last row. It returns true if the cursor is on a valid row, false otherwise.

```java
public static void main(String[] args) throws SQLException {
    DBInit.init();
    String dbURL = DBInit.URL, userName = null, passWord = null;

    Connection conn = DriverManager.getConnection(dbURL, userName, passWord);
    Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

    boolean hasResults = stmt.execute("SELECT * FROM employee");
    if (hasResults) {
        System.out.println("have data");
    } else {
        System.out.println("don't have data");
    }

    int result1 = stmt.executeUpdate("INSERT INTO EMPLOYEE (ID,NAME,PHONENO) VALUES (114,'George','+999000043210')"); // Returns 1
    int result2 = stmt.executeUpdate("UPDATE EMPLOYEE SET NAME='Joe' " + "WHERE id = 111"); // Returns 1
    int result3 = stmt.executeUpdate("DELETE FROM EMPLOYEE " + "WHERE id = 112"); // Returns 1
    System.out.println("executeUpdate -> INSERT::" + result1 + " UPDATE::" + result2 + " DELETE::" + result3);

    ResultSet rs = stmt.executeQuery("SELECT * FROM employee");
    System.out.println("ID \tNAME \tPHONENO");
    while (rs.next()) {
        System.out.println(rs.getInt("id") + "\t" + rs.getString("NAME") + "\t" + rs.getString("PHONENO"));
    }
    System.out.println("--------After update---------");
    rs.beforeFirst();
    while (rs.next()) {
        if (rs.getInt(1) == 112) {
            rs.updateString(2, "Jack");
        }
        System.out.println(rs.getInt("id") + "\t" + rs.getString("NAME") + "\t" + rs.getString("PHONENO"));

    }
    rs.absolute(2);
    System.out.println("result: -->" + rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString("PHONENO"));
}
```

ID	NAME	PHONENO
111	Michael	+919876543210
112	William	+449876543210
113	Trump	+99999543210

```
have data
executeUpdate -> INSERT::1 UPDATE::1 DELETE::1
ID      NAME    PHONENO
111     Joe     +919876543210
113     Trump   +99999543210
114     George  +999000043210
--------After update---------
111     Joe     +919876543210
113     Trump   +99999543210
114     George  +999000043210
result: -->113 Trump +99999543210
```


Calling beforeFirst() will set the cursor *before* row id 1
Calling relative(-2) will set the cursor for the row id 1
Calling previous() will set the cursor for the row id 2
Calling next() will set the cursor for the row id 4
Calling absolute(5) will set the cursor for the row id 5
Calling afterLast() will set the cursor *after* the row id 5
cursor
id
First
Second
1
aaaa
AAAA
2
bbb
BBBBBB
3
ccccc
CCC
4
ddd
DDD
5
eeeee
EEE

Important ResultSet Methods to get the data

```
getInt() returns an int
getLong() returns a Long
getString() returns a String
getObject() returns an Object
getDate() returns a java.sql.Date
getTime() returns a java.sql.Time
getTimeStamp() returns java.sql.Timestamp
```

```
Result rs = stmt.executeQuery(
      "SELECT insertion_date FROM user"
);
while(rs.next()) {
   // Getting the date part
   java.sql.Date sqlDate = rs.getDate(1);
   // Getting the time part
   java.sql.Time sqlTime = rs.getDate(1);
   // Getting both, the date and time part
   java.sql.Timestamp sqlTimestamp =
                        rs.getTimestamp(1);
   // Converting date
   LocalDate localDate = sqlDate.toLocalDate();
   // Converting time
   LocalTime localTime = sqlTime.toLocalTime();
   // Converting timestamp
   Instant instant = sqlTimestamp.toInstant();
   LocalDateTime localDateTime =
                sqlTimestamp.toLocalDateTime();
}
```

4.Close the connection to the database.

Before your program finishes, you need to close the connection

- The **(1)ResultSet** object is closed first, then the **(2)Statement** object, then the **(3)Connection** object.
- A ResultSet is automatically closed when another ResultSet is executed from the same Statement object.
- Closing a Statement also closes the ResultSet.
- Closing a Connection also closes the Statement and ResultSet objects.

```java
String user = null;
String passw = null;
DBInit.init();
try (Connection con = DriverManager.getConnection(DBInit.URL, user, passw);
        Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
    ) {

    ResultSet rs = stmt.executeQuery("SELECT * FROM employee");
    System.out.println("ID \tNAME \tPHONENO");
    while (rs.next()) {
        System.out.println(rs.getInt("id") + "\t"
                + rs.getString("NAME") + "\t"
                + rs.getString("PHONENO"));
    }

    System.out.println("----------------");

    ResultSet rs2 = stmt.executeQuery("SELECT * FROM employee Where id = 113"); // ResultSet rs is automatically closed
    System.out.println("ID \tNAME \tPHONENO");
    int numOfColumns = rs2.getMetaData().getColumnCount();
    while (rs2.next()) {
        for(int i = 1; i <= numOfColumns; i++) {
            System.out.print(rs2.getObject(i) + "\t");
        }
        System.out.println("");
    }

} catch (SQLException e) {
    e.printStackTrace();
}
```

```
ID      NAME    PHONENO
111     Michael +919876543210
112     William +449876543210
113     Trump   +99999543210
----------------
ID      NAME    PHONENO
113     Trump   +99999543210
```

X