Bienvenidos

Trabajando con Base de Datos
Relaciones

David R. Luna G.
davidrlunag@cibersys.com
0412-3111011

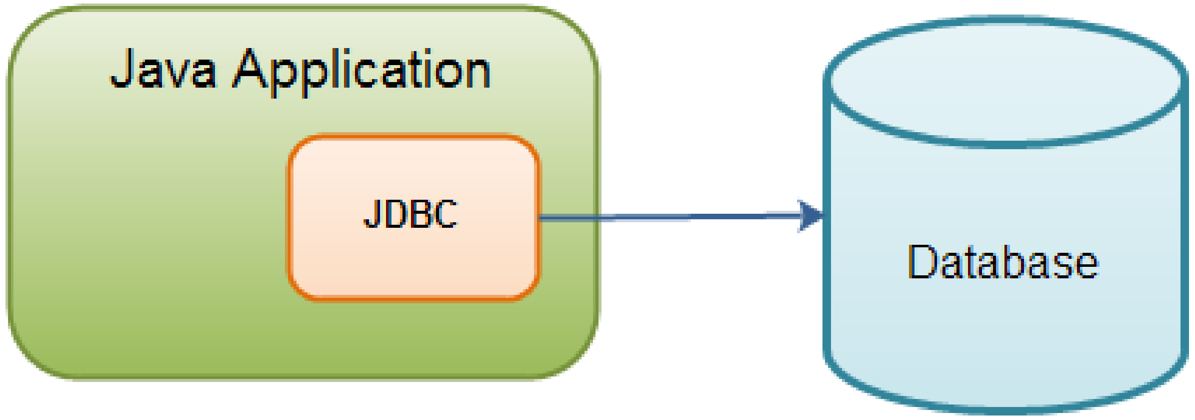
Temario del Curso

Trabajando con Base de Datos

JDBC database access
Leveraging the JDBC API
Choosing database drivers
Connecting to a database
Improving performance with prepared statements and stored procedures
Submitting SQL statements
Retrieving and processing results

Trabajando con Base de Datos

The Java JDBC API enables Java applications to connect to relational databases via a standard API, so your Java applications become independent (almost) of the database the application uses.



JDBC standardizes how to connect to a database, how to execute queries against it, how to navigate the result of such a query, and how to exeucte updates in the database.

Trabajando con Base de Datos

The JDBC API consists of the following core parts:

- ✓ JDBC Drivers
- ✓ ConnectionsV
- ✓ Statements
- ✓ Result Sets

JDBC Drivers

A JDBC driver is a collection of Java classes that enables you to connect to a certain database. For instance, MySQL will have its own JDBC driver. A JDBC driver implements a lot of the JDBC interfaces. When your code uses a given JDBC driver, it actually just uses the standard JDBC interfaces. The concrete JDBC driver used is hidden behind the JDBC interfaces. Thus you can plugin a new JDBC driver without your code noticing it.

Of course, the JDBC drivers may vary a little in the features they support.

Trabajando con Base de Datos

Connections

Once a JDBC driver is loaded and initialized, you need to connect to the database. You do so by obtaining a Connection to the database via the JDBC API and the loaded driver. All communication with the database happens via a connection. An application can have more than one connection open to a database at a time. This is actually very common.

Trabajando con Base de Datos

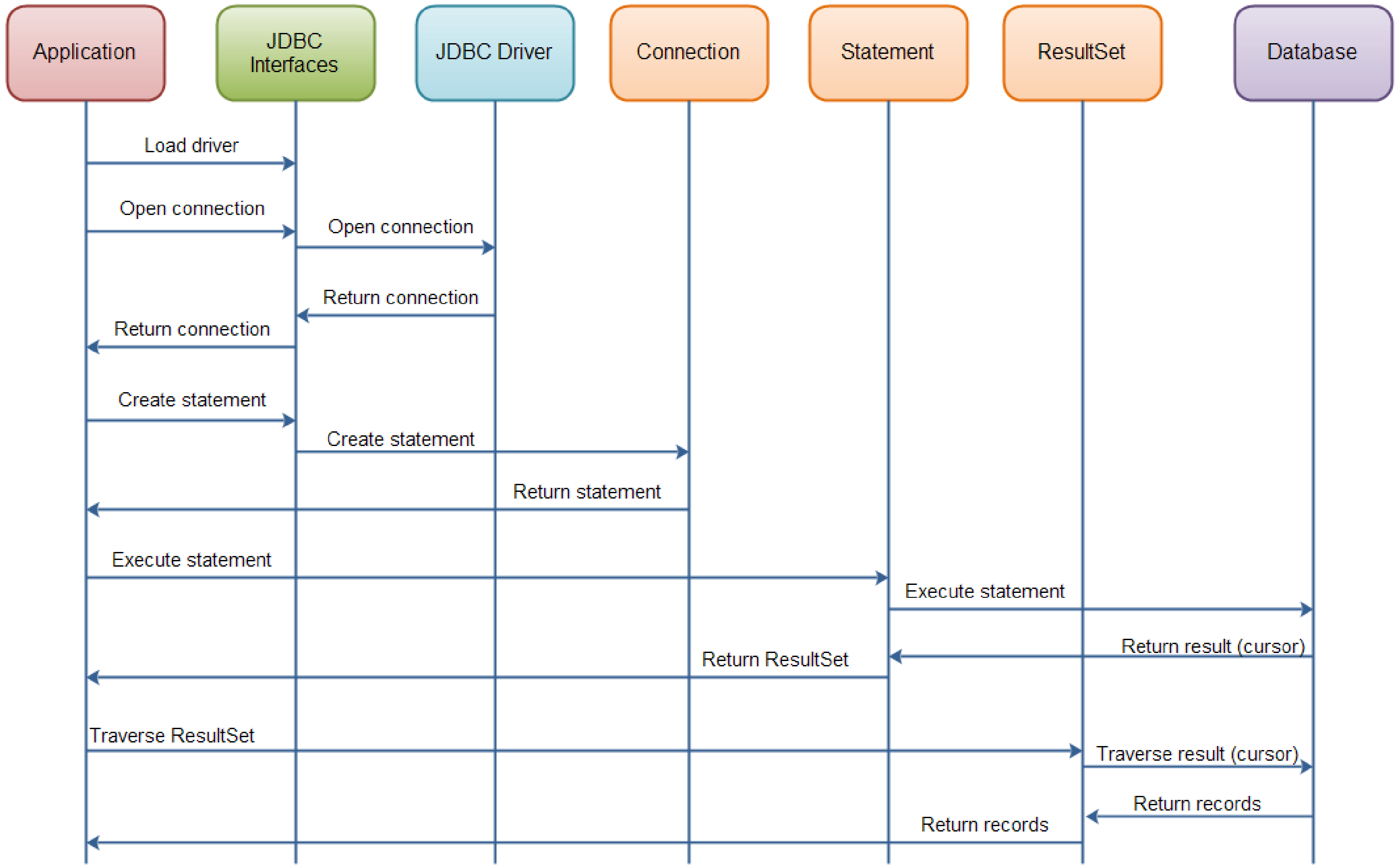
Statements

A Statement is what you use to execute queries and updates against the database. There are a few different types of statements you can use. Each statement corresponds to a single query or update.

ResultSets

When you perform a query against the database you get back a ResultSet. You can then traverse this ResultSet to read the result of the query.

Trabajando con Base de Datos

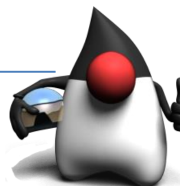


Trabajando con Base de Datos

A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database. The JDBC interfaces comes with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC driver.

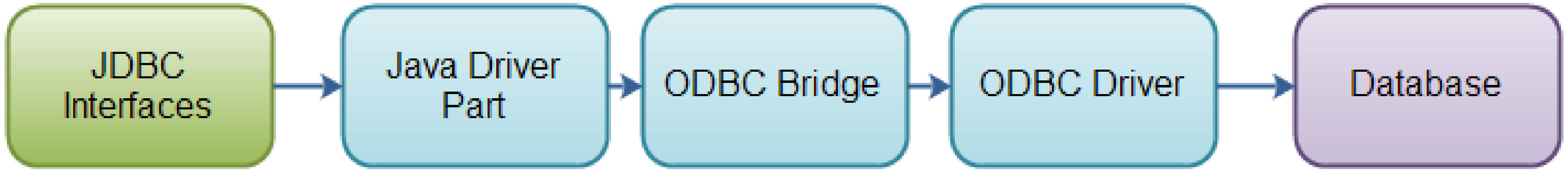
There are 4 different types of JDBC drivers:

Type 1: JDBC-ODBC bridge driver
Type 2: Java + Native code driver
Type 3: All Java + Middleware translation driver
Type 4: All Java driver.



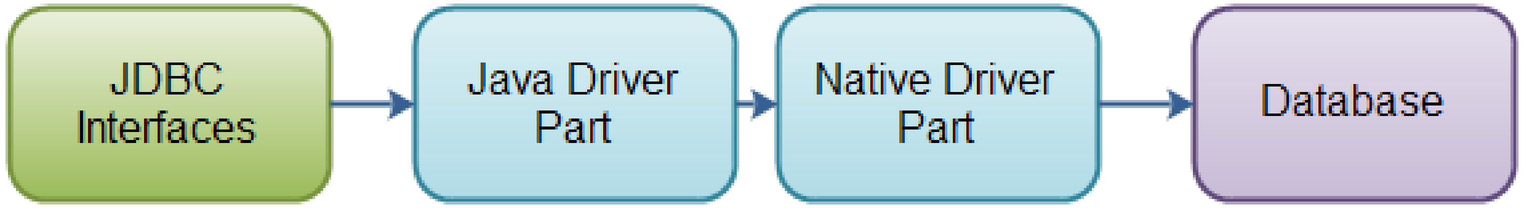
Type 1: JDBC-ODBC bridge driver

A type 1 JDBC driver consists of a Java part that translates the JDBC interface calls to ODBC calls. An ODBC bridge then calls the ODBC driver of the given database. Type 1 drivers are (were) mostly intended to be used in the beginning, when there were no type 4 drivers (all Java drivers).



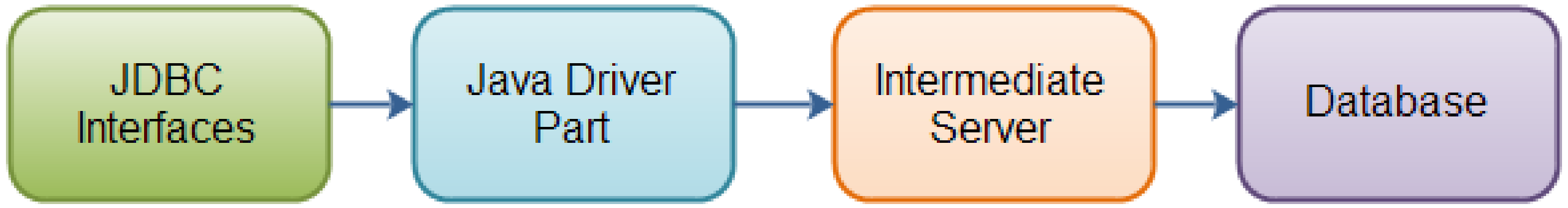
Type 2: Java + Native code driver

A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead. The native code part is targeted at a specific database product.



Type 3: All Java + Middleware translation driver

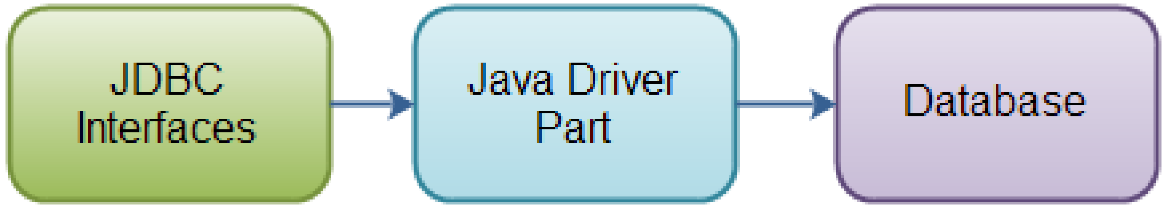
A type 3 JDBC driver is an all Java driver that sends the JDBC interface calls to an intermediate server. The intermediate server then connects to the database on behalf of the JDBC driver.



Trabajando con Base de Datos

Type 4: All Java driver.

A type 4 JDBC driver is an all Java driver which connects directly to the database. It is implemented for a specific database product. Today, most JDBC drivers are type 4 drivers. Here is an illustration of how a type 4 JDBC driver is organized:



Trabajando con Base de Datos

Open Database Connection

The first thing you need to do before you can open a database connection is to load the JDBC driver for the database.

```
Class.forName("driverClassName");
```

Each JDBC driver has a primary driver class that initializes the driver when it is loaded.

You only have to load the driver once. You do not need to load it before every connection opened. Only before the first connection opened.

Trabajando con Base de Datos

Open Database Connection

To open a database connection you use the java.sql.DriverManager class

```
Connection connection =
        DriverManager.getConnection(url, user, password);
```

jdbc:**subprotocolo**:**servidor.dominio**:**puerto/DB**

jdbc:mysql://localhost:3306

jdbc:sqlserver://localhost:1433;" +"databaseName=DB; user=userName;password=*****;

jdbc:postgresql://*localhost*:*5432/database*

Trabajando con Base de Datos

Query the Database

Querying a database means searching through its data. You do so be sending SQL statements to the database.

```
Statement statement = connection.createStatement();
String sql = "select * from clientes";
ResultSet result = statement.executeQuery(sql);
```

Query the Database

When you execute an SQL query you get back a ResultSet. The ResultSet contains the result of your SQL query. The result is returned in rows with columns of data.

```
while(result.next()) {
    String name = result.getString("name");
    long age = result.getLong ("age");
}
```

Trabajando con Base de Datos

Query the Database

You can get column data for the current row by calling some of the getXXX() methods, where XXX is a primitive data type. For instance:

```
result.getString ("columnName");
result.getLong ("columnName");
result.getInt ("columnName");
result.getDouble ("columnName");
result.getBigDecimal("columnName")
;
etc.
```

You can also pass an index of the column instead

```
result.getString (1);
result.getLong (2);
result.getInt (3);
result.getDouble (4);
result.getBigDecimal(5);
etc.
```

Trabajando con Base de Datos

Query the Database

For that to work you need to know what index a given column has in the ResultSet. You can get the index of a given column by calling the ResultSet.findColumn() method

```
int columnIndex =
result.findColumn("columnName");
```

When you are done iterating the ResultSet you need to close both the ResultSet and the Statement object that created it (if you are done with it, that is). You do so by calling their close() methods

Trabajando con Base de Datos

Query the Database

```
Statement statement = connection.createStatement(); String
sql = "select * from clientes";
ResultSet result = statement.executeQuery(sql);
while(result.next()) {
        String name = result.getString("name");
        long age = result.getLong("age");
        System.out.println(name);
         System.out.println(age);
}
result.close();
statement.close();
```

Trabajando con Base de Datos

Query the Database

```
Statement statement = null;
try{
    statement = connection.createStatement();
    ResultSet result = null;
    try{
             String sql = "select * from clientes";
             ResultSet result = statement.executeQuery(sql);             while(result.next()) {
                          String name = result.getString("name");
                          long age = result.getLong("age");
             System.out.println(name);
                          System.out.println(age); } }
    finally {
              if(result != null) result.close();
    }
 } finally { if(statement != null) statement.close(); }
```

Trabajando con Base de Datos

Query the Database

```
Statement statement = null;
try{
    statement = connection.createStatement();
    ResultSet result = null;
    try{
            String sql = "select * from clientes";
            ResultSet result = statement.executeQuery(sql);            while(result.next()) {
                        String name = result.getString("name");
                        long age = result.getLong("age");
            System.out.println(name);
                        System.out.println(age); } }
    finally {
             if(result != null) result.close();
    }
 } finally { if(statement != null) statement.close(); }
```

Trabajando con Base de Datos

Query the Database

The ability to use a try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement

```
Statement statement = null;
try{
    statement = connection.createStatement();
    ResultSet result = null;
    try{
            String sql = "select * from clientes";
            ResultSet result = statement.executeQuery(sql);
            while(result.next()) {
                    String name = result.getString("name");
                    long age = result.getLong("age");
                            System.out.println(name);
                    System.out.println(age); } }
    finally {
            if(result != null) result.close();
```

Trabajando con Base de Datos

Query the Database

In order to update the database you need to use a Statement. But, instead of calling the executeQuery()method, you call the executeUpdate() method.
There are two types of updates you can perform on a database:

- ✓ Update record values
- ✓ Delete records

The executeUpdate() method is used for both of these types of updates.

Trabajando con Base de Datos

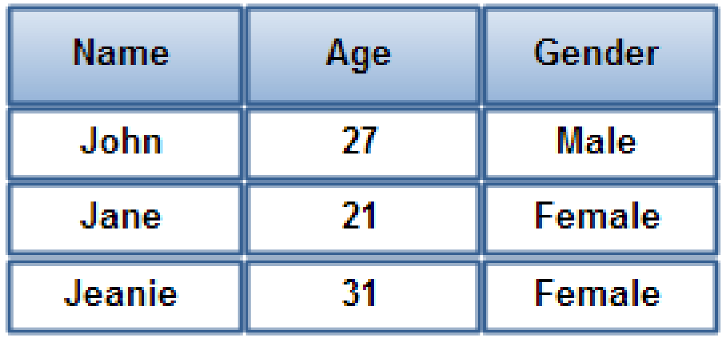
Query the Database

```
Statement statement = connection.createStatement();
String sql = "update people set name='John' where id=123";
int rowsAffected = statement.executeUpdate(sql);

Statement statement = connection.createStatement();
String sql = "delete from people where id=123";
int rowsAffected = statement.executeUpdate(sql);
```

Trabajando con Base de Datos

A ResultSet consists of records. Each records contains a set of columns. Each record contains the same amount of columns, although not all columns may have a value. A column can have a null value.



Name	Age	Gender
John	27	Male
Jane	21	Female
Jeanie	31	Female

Trabajando con Base de Datos

You create a ResultSet by executing a Statement or PreparedStatement

```
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery("select * from people");

String sql = "select * from people";
PreparedStatement statement = connection.prepareStatement(sql);
ResultSet result = statement.executeQuery();
```

Trabajando con Base de Datos

Query the Database

You can get column data for the current row by calling some of the getXXX() methods, where XXX is a primitive data type. For instance:

```
result.getString ("columnName");
result.getLong ("columnName");
result.getInt ("columnName");
result.getDouble ("columnName");
result.getBigDecimal("columnName")
;
etc.
```

You can also pass an index of the column instead

```
result.getString (1);
result.getLong (2);
result.getInt (3);
result.getDouble (4);
result.getBigDecimal(5);
etc.
```

PreparedStatement and CallableStatement

A PreparedStatement is a special kind of Statement object with some useful features. Remember, you need aStatement in order to execute either a query or an update. You can use a PreparedStatement instead of aStatement and benefit from the features of the PreparedStatement.

- ✓ Easy to insert parameters into the SQL statement.
- ✓ Easy to reuse the PreparedStatement with new parameters.
- ✓ May increase performance of executed statements.
- ✓ Enables easier batch updates.

PreparedStatement and CallableStatement

```
String sql = "update people set firstname=? , lastname=? where id=?";
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, "Gary");
preparedStatement.setString(2, "Larson");
preparedStatement.setLong (3, 123);
int rowsAffected = preparedStatement.executeUpdate();
```

Trabajando con Base de Datos

Once a PreparedStatement is prepared, it can be reused after execution. You reuse a PreparedStatement by setting new values for the parameters and then execute it again.

```
String sql = "update people set firstname=? , lastname=? where id=?";
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, "Gary");
 preparedStatement.setString(2, "Larson");
preparedStatement.setLong (3, 123);
 int rowsAffected = preparedStatement.executeUpdate();
preparedStatement.setString(1, "Stan");
preparedStatement.setString(2, "Lee");
preparedStatement.setLong (3, 456);
 int rowsAffected = preparedStatement.executeUpdate();
```