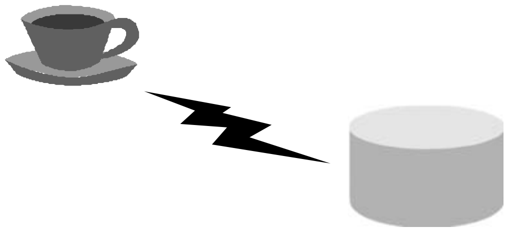
Introduction to

JDBC Programming

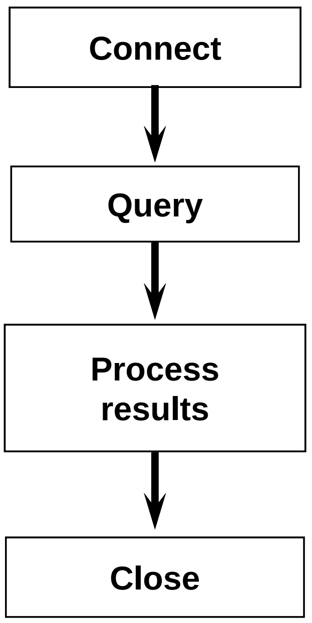
Oracle Korea

JDBC

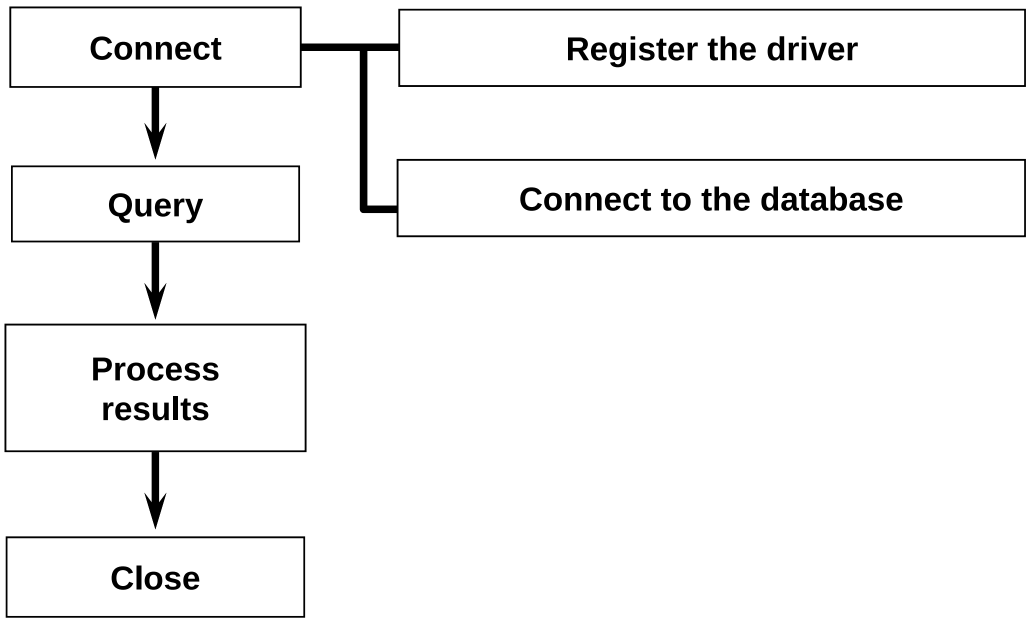
- JDBC is a standard interface for connecting to relational databases from Java.
- The JDBC classes and interfaces are in the ***java.sql*** package.



Overview of Querying a Database With JDBC

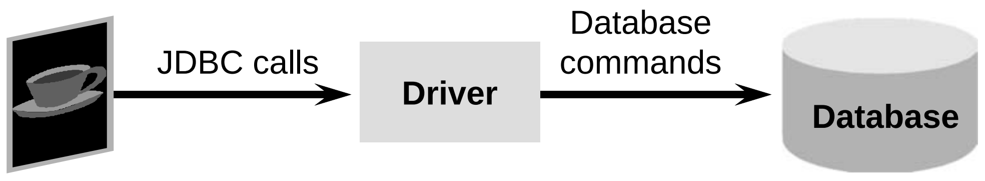


Stage 1: Connect



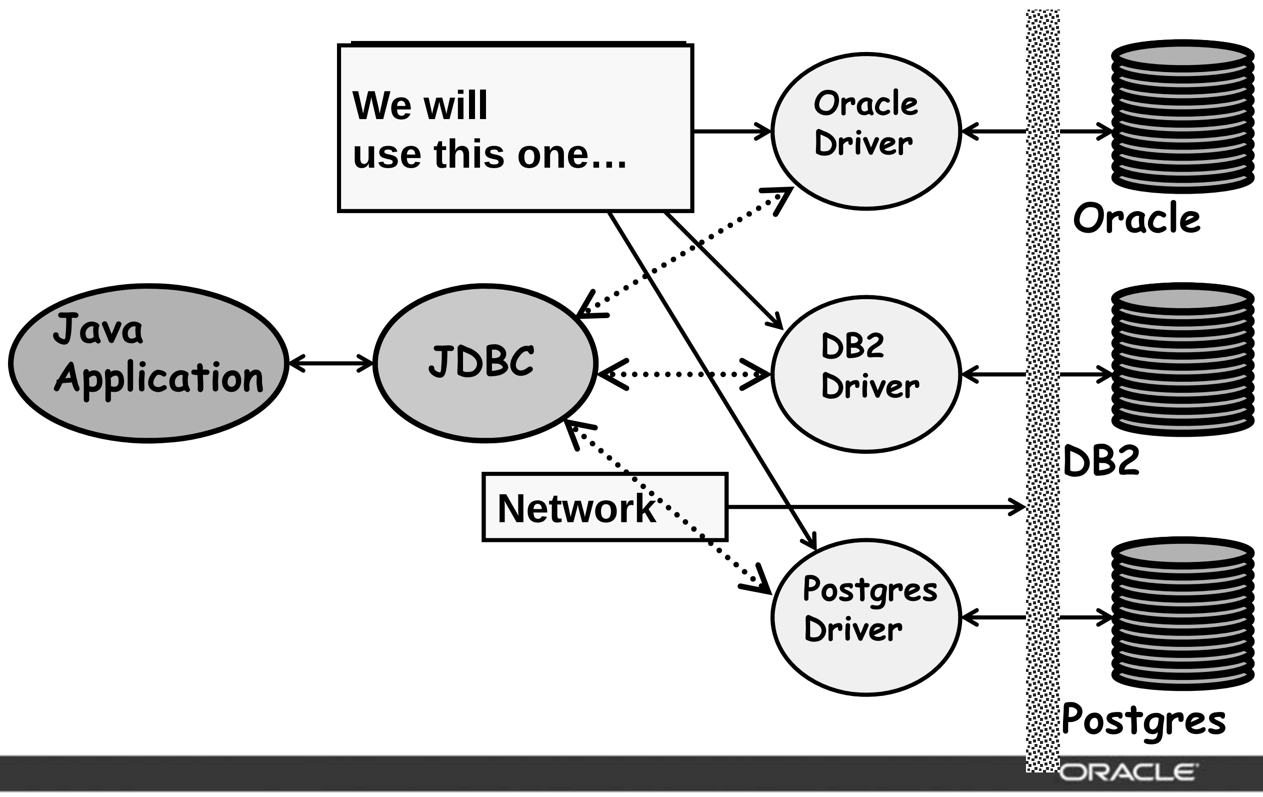
A JDBC Driver

- Is an interpreter that translates JDBC method calls to vendor-specific database commands

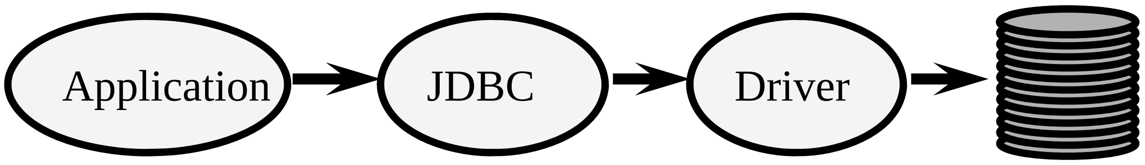


- Implements interfaces in java.sql
- Can also provide a vendor's extensions to the JDBC standard

JDBC Architecture



JDBC Architecture (cont.)



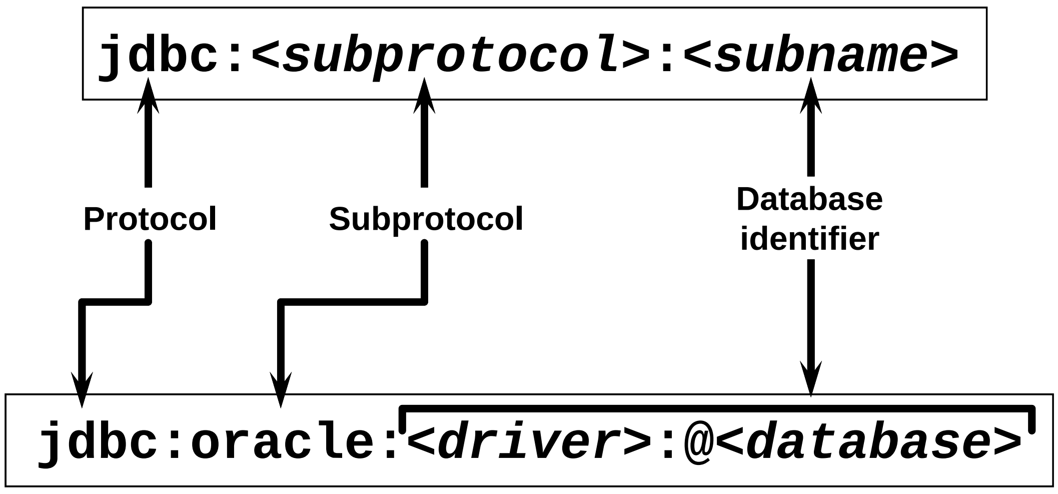
- Java code calls JDBC library
- JDBC loads a *driver*
- Driver talks to a particular database
- An application can work with several databases by using all corresponding drivers
- Ideal: can change database engines *without changing any application code* (not always in practice)

Now, It’s Setup Time!!!

- Create a simple database Java application (With Eclipse)
  - 별도의 문서 “Project 03 - Eclipse - JDBC 연동”을 참고

About JDBC URLs

- JDBC uses a URL to identify the database connection.


jdbc:<subprotocol>:<subname>
Protocol
Subprotocol
Database identifier
jdbc:oracle:<driver>:@<database>

JDBC URLs with Oracle Drivers

- Thin driver

```
jdbc:oracle:thin:@<host>:<port>:<SID>
```

How to Make the Connection

1. Register the driver.

```
DriverManager.registerDriver (new
     oracle.jdbc.driver.OracleDriver());
```

2. Connect to the database.

```
Connection conn = DriverManager.getConnection
     (URL, userid, password);
```

```
Connection conn = DriverManager.getConnection
   ("jdbc:oracle:thin:@localhost:1521:orcl”,
    “system", “YOUR_PASSWORD");
```

Using Connection

java.sql.Connection

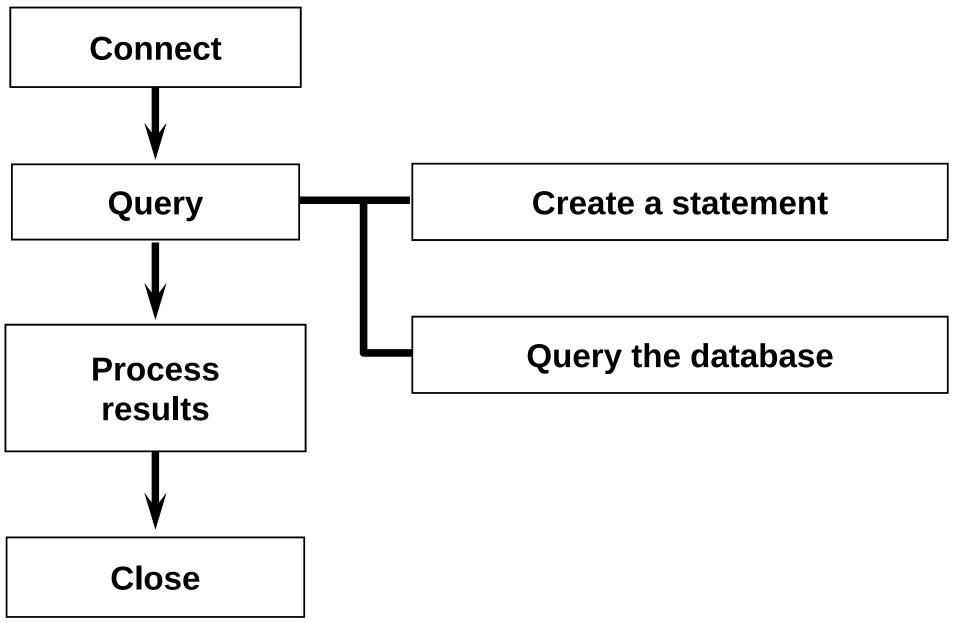
createStatment()
prepareStatment(String)
prepareCall(String)
	Creating Statement

commit()
rollback()
	Transaction Management

getMetaData()	Get database metadata

close()
isClosed()
	Conneciton related

Stage 2: Query



The Statement Object

- A Statement object sends your SQL statement to the database.
- You need an active connection to create a JDBC statement.
- Statement has three methods to execute a SQL statement:
  - **executeQuery()** for QUERY statements
  - **executeUpdate()** for INSERT, UPDATE, DELETE, or DDL statements
  - **execute()** for either type of statement

How to Query the Database

1. Create an empty statement object.

```
Statement stmt = conn.createStatement();
```

2. Execute the statement.

```
ResultSet rset = stmt.executeQuery(statement);          SELECT
int count = stmt.executeUpdate(statement);   UPDATE, INSERT, DELETE
boolean isquery = stmt.execute(statement);            CREATE TABLE
```

Querying the Database: Examples

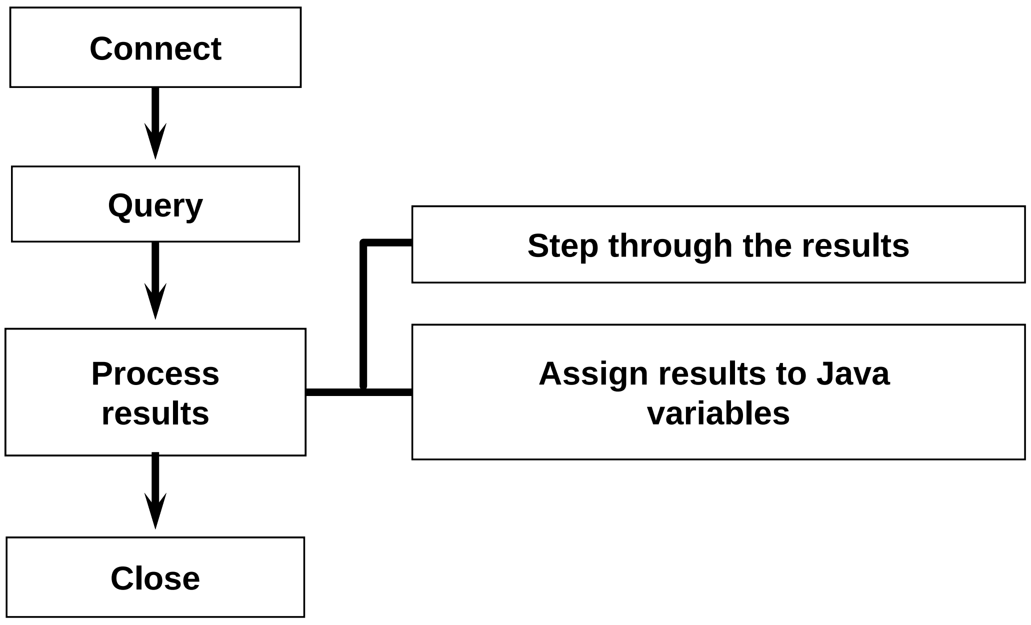
- Execute a select statement.

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
  ("select ID, NAME from INSTRUCTOR");
```

- Execute a delete statement.

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("delete from INSTRUCTOR  where ID = 14365");
```

Stage 3: Process the Results



The ResultSet Object

- JDBC returns the results of a query in a ResultSet object.
- A ResultSet maintains a cursor pointing to its current row of data.
- Use next() to step through the result set row by row.
- getString(), getInt(), and so on assign each value to a Java variable.

How to Process the Results

- 1. Step through the result set.

```
while (rset.next()) { … }
```

- 2. Use getXXX() to get each column value.

```
String val =
rset.getString(colname);
```

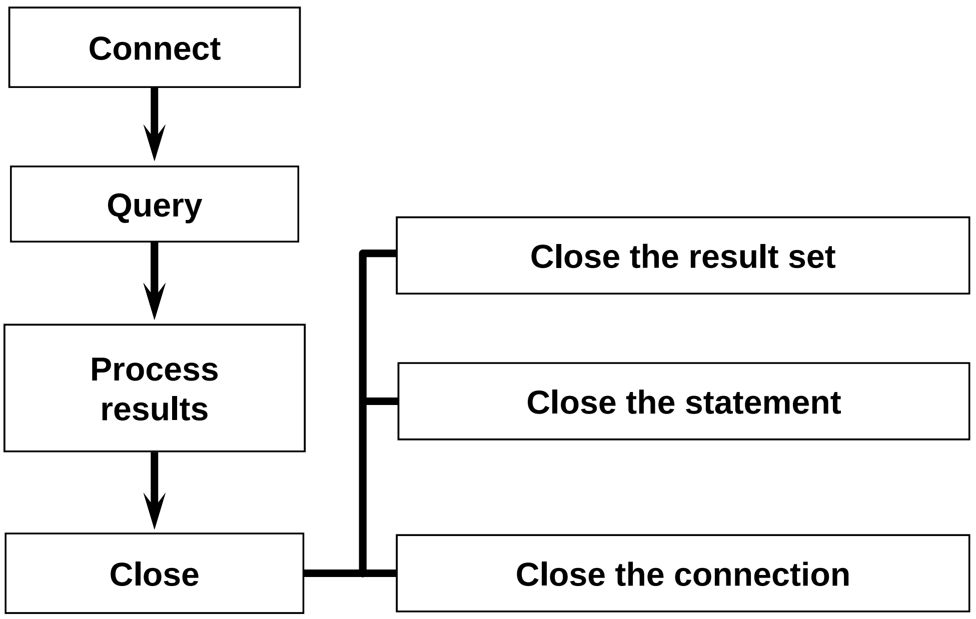
```
String val =
rset.getString(colIndex);
```

```
while (rset.next()) {
  int  id = rset.getInt(“ID");
  String name = rset.getString(“NAME");
… // Process or display the data
}
```

ResultSet Methods

- *Type* get*Type*(int columnIndex)
  - returns the given field as the given type
  - indices start at 1 and not 0!
- *Type* get*Type*(String columnName)
  - same, but uses name of field
  - less efficient
- For example: getString(columnIndex), getInt(columnName), getTime, getBoolean, getType,...
- int findColumn(String columnName)
  - looks up column index given column name

Stage 4: Close



How to Close the Connection

1. Close the ResultSet object.

```
rset.close();
```

2. Close the Statement object.

```
stmt.close();
```

3. Close the connection (not necessary for server-side driver).

```
conn.close();
```

Let’s Do It Yourself !!!

```
import java.sql.*;

public class Database {

  public static void main(String[] args) {

    String DB_URL = "jdbc:oracle:thin:@localhost:1521:orcl";

    try{
        Connection con = DriverManager.getConnection(DB_URL, “system", “YOUR_PASSWORD");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT id, name FROM instructor");
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            System.out.println(id + " : " + name);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
  }
}
```

More About ResultSet Methods

- JDBC 2.0 includes scrollable result sets. Additional methods included are : ‘first’, ‘last’, ‘previous’, and other methods.

APIs

java.sql.ResultSet

```
void beforeFirst() throws SQLException
void afterLast() throws SQLException
boolean first() throws SQLException
boolean last() throws SQLException
boolean absolute(int row) throws SQLException
boolean relative(int row) throws SQLException
```

```
void deleteRow(int row) throws SQLException
void updateXXX(int idx, XXX x) throws SQLException
void updateRow() throws SQLException
void moveToInsertRow () throws SQLException
void moveToCurrentRow() throws SQLException
void insertRow() throws SQLException
```

Example : Backward

```
Statement stmt = conn.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT empno, sal FROM emp");
rs.afterLast();
while ( rs.previous() )
{
System.out.println(rs.getString("empno") + " " + rs.getFloat("sal"));
}
...
```

Example : delete row

```
...
rs.absolute(5);
rs.deleteRow();
...
```

Example : update row

```
Statement stmt = conn.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT empno, sal FROM emp");
if (rs.absolute(10)) // (returns false if row does not exist)
{
rs.updateString(1, "28959");
rs.updateFloat("sal", 100000.0f);
rs.updateRow();
}
// Changes will be made permanent with the next COMMIT operation.
...
```

Example : insert row

```
...
Statement stmt = conn.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT empno, sal FROM emp");
rs.moveToInsertRow();
rs.updateString(1, "28959");
rs.updateFloat("sal", 100000.0f);
rs.insertRow();
// Changes will be made permanent with the next COMMIT operation.
rs.moveToCurrentRow(); // Go back to where we came from...
...
```

More About Mapping Database Types to Java Types

- ResultSet maps database types to Java types.

```
ResultSet rset = stmt.executeQuery
  ("select RENTAL_ID, RENTAL_DATE, STATUS
  from ACME_RENTALS");

int id = rset.getInt(1);
Date rentaldate = rset.getDate(2);
String status = rset.getString(3);
```

Col Name	Type
RENTAL_ID	NUMBER
RENTAL_DATE	DATE
STATUS	VARCHAR2

Oracle Type	Java Type
CHAR	String
VARCHAR	String
DATE	java.sql.Date java.sql.Time java.sql.Timestamp
INTEGER	short int long
NUMBER	float double java.math.BigDecimal

Oracle and Java Types

- From this table, you can see that an Oracle INTEGER is compatible with a Java int.
- So, the **id** column (INTEGER) of the customers table may be stored in a Java ***int*** ***.***
- Similarly, the **first_name, last_name,** and **phone** column values may be stored in Java ***String*** variables

Oracle and Java Types

- The Oracle DATE type stores a year, month, day, hour, minute, and second.
- You may use:
  - ***java.sql.Date*** to store the date part of the dob column value
  - ***java.sql.Time*** to store the time part
  - ***java.sql.Timestamp*** to store both the date and the time parts

Oracle and Java Types

- The int and String types are part of the core Java language
- java,.sql.Date ia part of JDBC
- However, JDBC doesn’t cover all types used by Oracle:
  - You must use oracle.sql.ROWID type to store Oracle ROWID
- So, Oracle provides a number of additional types in oracle.sql package

How to Handle SQL Null Values

- Java primitive types cannot have null values.
- Do not use a primitive type when your query might return a SQL null.
- Use ResultSet.wasNull() to determine whether a column has a null value.

```
while (rset.next()) {
  String year = rset.getString("YEAR");
  if (rset.wasNull() {
    … // Handle null value
  }
…}
```

The PreparedStatement Object

- A PreparedStatement object holds precompiled SQL statements.
- Use this object for statements you want to execute more than once.
- A prepared statement can contain variables that you supply each time you execute the statement.

How to Create a Prepared Statement

1. Register the driver and create the database connection.
2. Create the prepared statement, identifying variables with a question mark (?).

```
PreparedStatement pstmt =
  conn.prepareStatement("update INSTRUCTOR
  set NAME = ? where ID = ?");
```

```
PreparedStatement pstmt =
  conn.prepareStatement("select NAME from
  INSTRUCTOR where ID = ?");
```

How to Execute a Prepared Statement

1. Supply values for the variables.

```
pstmt.setXXX(index, value);
```

2. Execute the statement.

```
pstmt.executeQuery();
pstmt.executeUpdate();
```

```
PreparedStatement pstmt =
  conn.prepareStatement("update INSTRUCTOR
  set NAME = ? where ID = ?");
pstmt.setString(1, “Lim");
pstmt.setInt(2, 99052);
pstmt.executeUpdate();
```

Let’s Do It Yourself !!!

```
try{
   …
   PreparedStatement pstmt = conn.prepareStatement("select NAME from INSTRUCTOR where ID = ?");
   pstmt.setInt(1, 99052);
   rs = pstmt.executeQuery();
   while (rs.next()) {
      String name = rs.getString("name");
      System.out.println(name);
   }
   pstmt = conn.prepareStatement("update INSTRUCTOR set NAME = ? where ID = ?");
   pstmt.setString(1, "Lim");
   pstmt.setInt(2, 99052);
   pstmt.executeUpdate();
   int count = pstmt.executeUpdate();
   System.out.println(count);

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

Statements vs. PreparedStatements: Be Careful!

- Are these the same? What do they do?

```
String val = "abc";
PreparedStatement pstmt =
        con.prepareStatement("select * from R where A=?");
pstmt.setString(1, val);
ResultSet rs =  pstmt.executeQuery();
```

```
String val = "abc";
Statement stmt =  con.createStatement( );
ResultSet rs =
        stmt.executeQuery("select * from R where A=" + val);
```

Statements vs. PreparedStatements: Be Careful!

- Will this work?

```
PreparedStatement pstmt =
        con.prepareStatement("select * from ?");


pstmt.setString(1, myFavoriteTableString);
```

- No!!! A '?' can only be used to represent a column value

Using Transactions

- The server-side driver does not support autocommit mode.
- With other drivers:
  - **New connections are in autocommit mode**.
  - **Use conn.setAutoCommit(false) to turn autocommit off**.
- To control transactions when you are not in autocommit mode:
  - conn.commit(): Commit a transaction
  - conn.rollback(): Roll back a transaction