Escuela Java - JDBC

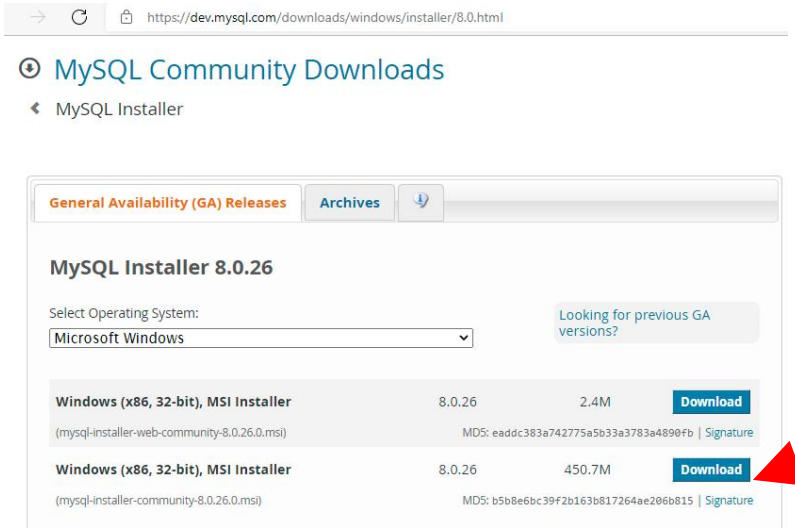
Pulse para añadir texto

TOGETHER. FREE YOUR ENERGIES

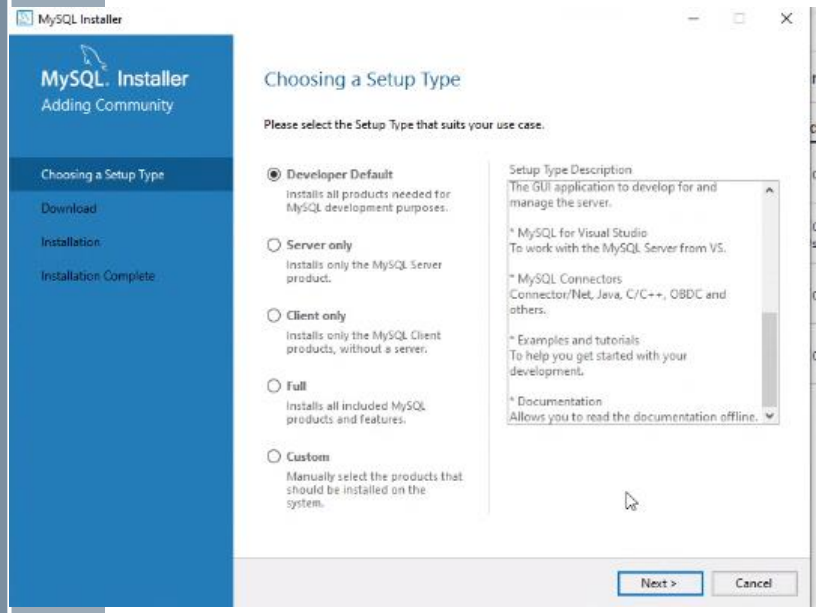
BASES DE DATOS: MYSQL

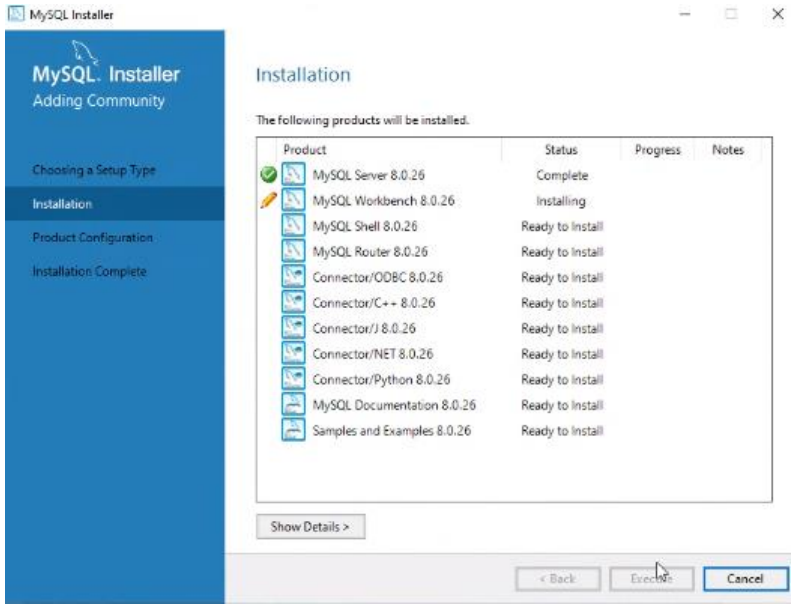
INSTALACION

- MySQL :: Download MySQL Installer
  - https://dev.mysql.com/downloads/windows/installer/8.0.html
  - PARA MAC: https://dev.mysql.com/downloads/mysql/



Instalar
- Community Server
- Workbench

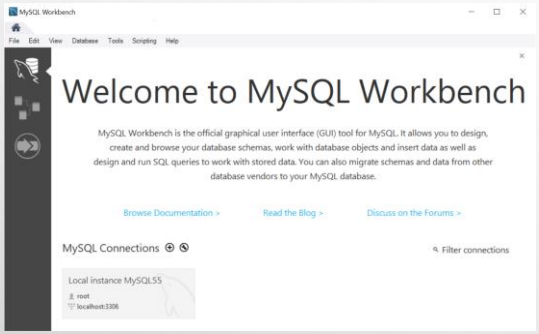

MySQL Installer
MySQL. Installer
Adding Community
Choosing a Setup Type
Download
Installation
Installation Complete
Choosing a Setup Type
Please select the Setup Type that suits your use case.
Developer Default
Installs all products needed for MySQL development purposes.
Server only
Installs only the MySQL Server product.
Client only
Installs only the MySQL Client products, without a server.
Full
Installs all included MySQL products and features.
Custom
Manually select the products that should be installed on the system.
Setup Type Description
The GUI application to develop for and manage the server.
* MySQL for Visual Studio
To work with the MySQL Server from VS.
* MySQL Connectors
Connector/Net, Java, C/C++, OBDC and others.
* Examples and tutorials
To help you get started with your development.
* Documentation
Allows you to read the documentation offline.
Next >
Cancel

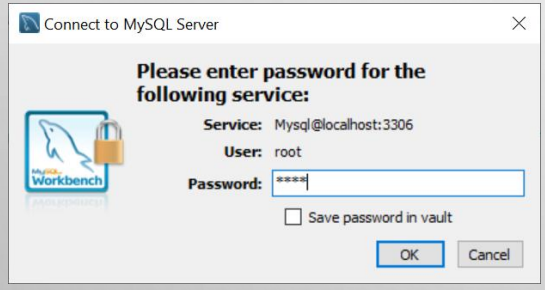

MySQL Installer
MySQL. Installer
Adding Community
Choosing a Setup Type
Installation
Product Configuration
Installation Complete
Installation
The following products will be installed.
Product
Status
Progress
Notes
MySQL Server 8.0.26
Complete
MySQL Workbench 8.0.26
Installing
MySQL Shell 8.0.26
Ready to Install
MySQL Router 8.0.26
Ready to Install
Connector/ODBC 8.0.26
Ready to Install
Connector/C++ 8.0.26
Ready to Install
Connector/J 8.0.26
Ready to Install
Connector/NET 8.0.26
Ready to Install
Connector/Python 8.0.26
Ready to Install
MySQL Documentation 8.0.26
Ready to Install
Samples and Examples 8.0.26
Ready to Install
Show Details >
< Back
Execute
Cancel

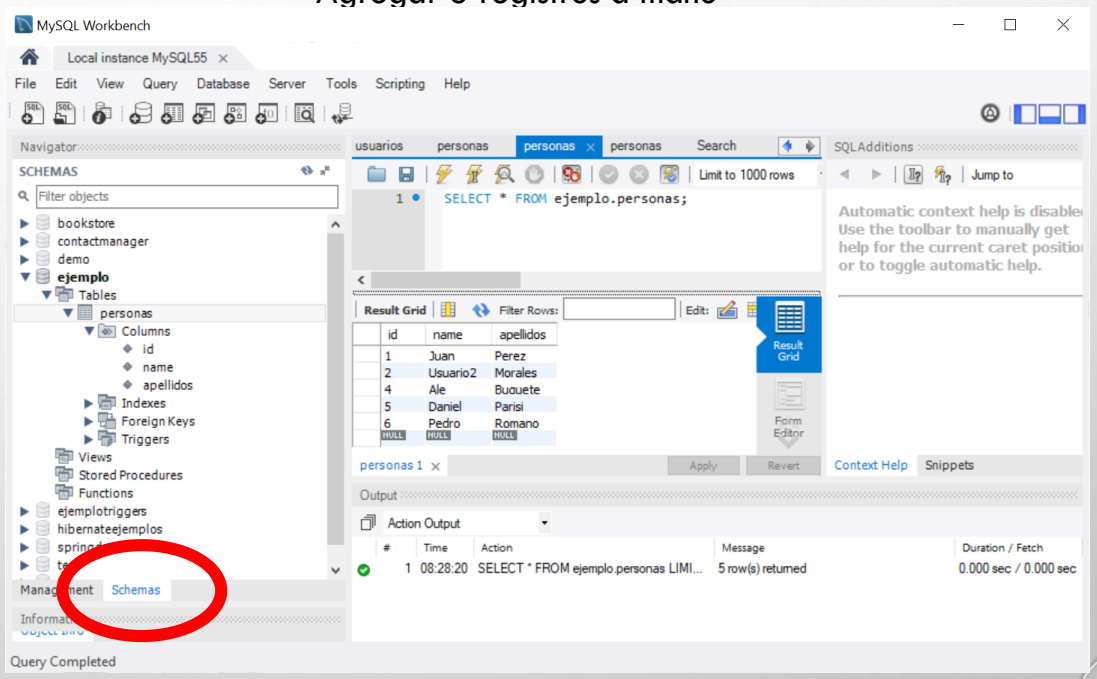
BASE DE DATOS

Crear nuevo Schema ➔ **ejemplo**
Crear nueva Table ➔ **personas**
Campos **id** (autoincrement, pk, int)
name (varchar)
apellidos (varchar)
Agregar 3 registros a mano

Login







JDBC I

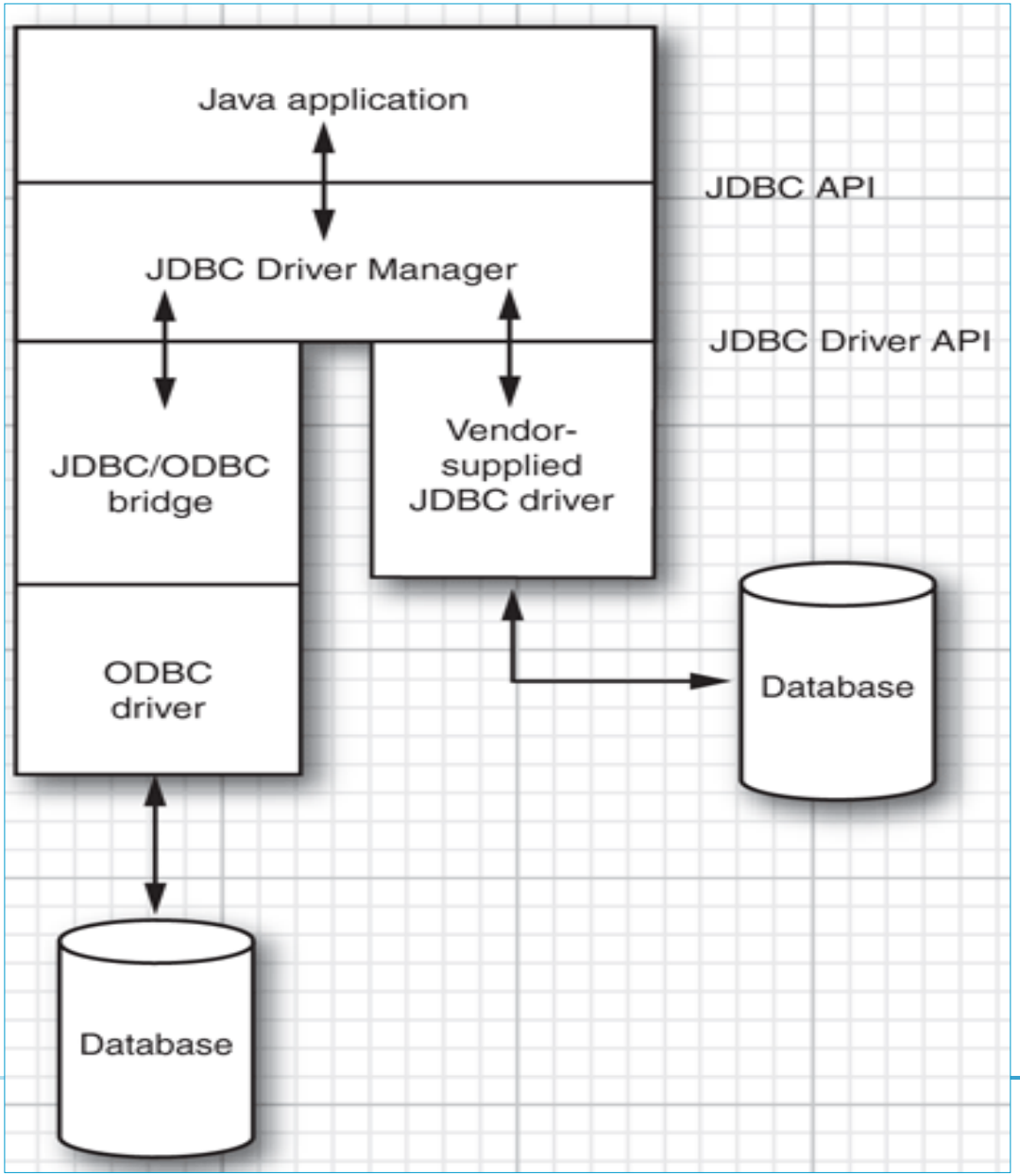
¿Qué es JDBC?

- **JDBC (Java Database Connectivity) es la API estándar de acceso a bases de datos con Java. Nos proporciona un conjunto de interfaces y clases Java para acceder de forma genérica a bases de datos independientemente del fabricante.**

- **Para trabajar con JDBC es necesario tener controladores (drivers) que permitan acceder a las distintas bases de datos. Para ello, los fabricantes de BDD deben proporcionarnos el driver.**

JDBC II

¿Qué es JDBC?



JDBC III

Esquema de acceso

Una aplicación que usa JDBC realiza los siguientes pasos:

1. Carga el driver necesario para la conexión
2. Establece una conexión con un DBMS
3. Crea y envía una sentencia SQL al DBMS
4. Procesa el resultado
5. Libera recursos

JDBC IV

1 Carga el driver necesario para la conexión

- Cargar el driver JDBC para el acceso al DBMS

```
Class.forName(String nombreDelDriver)
```

Si no se encuentra se produce la excepción ClassNotFoundException

El gestor de drivers:
java.sql.DriverManager

JDBC V

2 Establece una conexión con un DBMS

- Para abrir una conexión se utiliza el método getConnection de la clase DriverManager:

```
Connection con =DriverManager.getConnection (String url, String user, String pwd)
```

- **url:** es la cadena de conexión indicada por el fabricante para la conexión con su base de datos.
- **user y pwd:** Usuario y password de acceso a la BD

Cuando hayamos terminado de acceder a la BD, debemos cerrar la conexión con el método close():

```
con.close();
```

java.sql.Connection

JDBC VI

3 Crea y envía una sentencia SQL al DBMS

Para crear una sentencia SQL tenemos tres tipos de "statements" que utilizan los siguientes métodos de la clase Connection:

- **con.createStatement()**

  Se utiliza para sentencias sin información dinámica.

- **con.prepareStatement(sql)**

  Para sentencias en las que necesitemos información dinámica (parámetros)

- **con.prepareCall(sql)**

  Para llamadas a procedimientos almacenados en la base de datos

java.sql.PreparedStatement
java.sql.Connection
java.sql.Statement
java.sql.CallableStatement

JDBC VII

3 Crea y envía una sentencia SQL al DBMS

Ejecutar la sentencia SQL

Para ejecutar la sentencia podemos utilizar uno de los siguientes métodos de la clase Statement dependiendo de su propósito:

- **Sentencias SELECT: executeQuery(String sql)**
  Devuelve una instancia de ResultSet
- **Sentencias INSERT, UPDATE y DELETE: executeUpdate(String sql)**
  Devuelve un int con el número de filas afectadas
- **Sentencias CREATE TABLE y DROP TABLE: executeUpdate(String sql)**
  Devuelve un int

java.sql.ResultSet

JDBC VIII

3 Crea y envía una sentencia SQL al DBMS

Sentencias SQL básicas

```
CREATE TABLE contacto (
nombre VARCHAR(49),
email VARCHAR(30) NOT NULL DEFAULT ‘’,
telefono VARCHAR(15),
nacimiento DATETIME,
PRIMARY KEY (email),
```

```
SELECT nombre, edad
FROM alumnos
WHERE edad>30
ORDER BY nombre
```

```
INSERT INTO contacto (nombre, telefono, email)
VALUES (“Juan José”, “999-99-99-99”, “juan@jose.com”)
```

```
UPDATE contacto SET nombre = “Juan Gómez García”
WHERE email = “juanl@gmail.com”;
```

```
DELETE FROM contacto
WHERE email = “juan@jose.com”
```

JDBC IX

4 Procesa el resultado

Resultados de la sentencia

Si hemos ejecutado un SELECT, las filas de la BDD que han concordado con la búsqueda se almacenarán en un objeto de la clase ResultSet.

ResultSet implementa varios métodos para poder acceder a los datos recuperados:

- `public boolean next() throws SQLException`

  Accede a la siguiente fila del resultado

- `public boolean previous() throws SQLException`

  Accede a la fila anterior del resultado

- `public XXXX getXXXX(int numeroColumna) SQLException`
  `public XXXX getXXXX(String nombreColumna) SQLException`

  Recupera el dato de una columna

java.sql.ResultSet

JDBC X

Correspondencia de tipos entre Java y SQL

Tipo SQL	Tipo Java	Método GET
CHAR	java.lang.String	getString()
VARCHAR	java.lang.String	getString()
LONGVARCHAR	java.lang.String	getAsciiStream()
NUMERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	boolean	getBoolean()
BOOLEAN	boolean	getBoolean()
TINYINT	byte	getByte()
SMALLINT	short	getShort()
INTEGER	int	getInt()
BIGINT	long	getLong()
REAL	float	getFloat()
FLOAT	double	getDouble()
DOUBLE	double	getDouble()
BINARY	byte[]	getBytes()
VARBINARY	byte[]	getBytes()
LONGVARBINARY	byte[]	getBinaryStream()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

JDBC XI

5 Libera recursos

- En principio, aunque no se llame a Connection.close(), cuando la conexión sea eliminada por el recolector de basura, el método *finalize()* de la clase que implementa Connection, invocará al método *close()*.

- Además
  - •Cuando se cierra una conexión, cierra todos sus Statements asociados
  - •Cuando se cierra un Statement, cierra todos sus ResultSets asociados

- Sin embargo,
  Es imprescindible cerrar las conexiones tan pronto como se pueda.
  Mejor cerrar los Statement explícitamente.

JDBC XII

Ejemplo

```
Connection conexion = null;
Statement sentenciaSQL = null;
ResultSet resultado = null;

try {
   Class.forName("com.mysql.jdbc.Driver");

   String url = "jdbc:mysql://localhost/alumnos";
   conexion = DriverManager.getConnection(url, "java", "java");

   sentenciaSQL = conexion.createStatement();

   String sqlString = "select * from alumnos";
   resultado = sentenciaSQL.executeQuery(sqlString);

   while (resultado.next()) {
      System.out.println(resultado.getString("nombre"));
   }
} catch (ClassNotFoundException e) {
   e.printStackTrace();
} catch (SQLException e) {
   e.printStackTrace();
}
```

JDBC XIII

Ejemplo (cont.)

```
finally {
        try {
            if (resultado != null)
                resultado.close();
            if (sentenciaSQL != null)
                sentenciaSQL.close();
            if (conexion != null)
                conexion.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Ejercicio

Ejercicio I de la guía

JDBC XIV

Prepared Statements

- Las PreparedStatements son consultas precompiladas por el motor de la base de datos, por lo que su ejecución es más rápida. Además admiten el paso de parámetros.

- Suelen utilizarse cuando vemos que repetimos muchas veces la misma consulta cambiando algunos parámetros.

- Se crean de forma similar a las sentencias normales, sólo que en el lugar de cada parámetro va un símbolo ? y antes de ejecutarla tendremos que dar valor a los parámetros:

```
String cadenaSQL = "SELECT * FROM Alumnos WHERE curso=?";
sentenciaSQL = conexion.prepareStatement(cadenaSQL);
sentenciaSQL.setInt(1, i);
resultado = sentenciaSQL.executeQuery();
```

JDBC XV

1. Se prepara la instrucción SQL
2. Se asignan los valores a las interrogaciones (parámetros)
3. Se ejecuta la instrucción.

```
public boolean insertarContacto(Contacto contacto) throws SQLException {
    String cadena = "INSERT INTO contactos (email,nombrePila,telefono) VALUES (?,?,?)";
    PreparedStatement sentenciaSQL = conexion.prepareStatement(cadena);
    sentenciaSQL.setString(1, contacto.getEmail());
    sentenciaSQL.setString(2, contacto.getNombrePila());
    sentenciaSQL.setString(3, contacto.getTelefono());
    int i = sentenciaSQL.executeUpdate();
    if (i > 0)
        return true;
    return false;
}
```

JDBC XVI

Scrollable y Updatable Result Sets

```
Statement stat = conn.createStatement(type, concurrency);
```

```
PreparedStatement stat = conn.prepareStatement(command, type, concurrency);
```

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

```
Statement stat = conn.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

JDBC XVII

Moverse dentro de un RecordSet

```
ResultSet rs = stat.executeQuery(query)
```

```
if (rs.previous()) . . .
```

```
rs.relative(n);
```

```
rs.absolute(n);
```

```
int currentRow = rs.getRow();
```

first, last, beforeFirst, and afterLast

isFirst, isLast, isBeforeFirst, and isAfterLast

JDBC XVIII

Actualizar un RecordSet Updatable

UPDATE

```
Statement stat = conn.createStatement(
   ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);


String query = "SELECT * FROM Books";
ResultSet rs = stat.executeQuery(query);
while (rs.next())
{
   if (. . .)
   {
      double increase = . . .
      double price = rs.getDouble("Price");
      rs.updateDouble("Price", price + increase);
      rs.updateRow();
 }
}
```

JDBC XIX

Actualizar un RecordSet Updatable

INSERT

```
rs.moveToInsertRow();
rs.updateString("Title", title);
rs.updateString("ISBN", isbn);
rs.updateString("Publisher_Id", pubid);
rs.updateDouble("Price", price);
rs.insertRow();
rs.moveToCurrentRow();
```

DELETE

```
rs.deleteRow();
```

JDBC XX

Rowsets

La interface Rowset extiende de la interface ResultSet, pero las filas no están atadas a una conexión a la BD (Wow!)

CachedRowSet — Permite operar desconectado del DBMS

FIlteredRowSet

JoinRowSet

Soporta operaciones en Rowsets que son equivalentes al SELECT y JOIN de SQL.

JDBC XXI

CachedRowSet

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

      conn = DriverManager.getConnection(url);

      stmt = conn.createStatement();
      resultSet = stmt.executeQuery(query);

      crs = new CachedRowSetImpl();
      crs.populate(resultSet);

      conn.close();

       while (crs.next()) {
            String name = crs.getString(1);
            int id = crs.getInt(2);
            short dept = crs.getShort(4);
            System.out.println(name + "  " + id + "  " + comment + "  " + dept);
     }
  }
```

Ejercicio

Ejercicio II de la guía

Muchas gracias!