

		IDBC

## Contenidos

- El desfase objeto-relacional.
- Protocolos de acceso a bases de datos. Conectores.
- Ejecución de sentencias de definición de datos.
- Ejecución de sentencias de manipulación de datos.
- Ejecución de consultas
- Clase CallableStatement
- Gestión de transacciones

# El desfase objeto-relacional

- El problema del desfase objeto-relacional consiste en la diferencia de aspectos que existen entre la programación orientada a objetos y la base de datos.
- Surgen cuando:
  - Se realizan actividades de programación.
  - Se especifican los tipos de datos.
  - En el proceso de elaboración del software se realiza una traducción del modelo orientado a objetos al model E-R.

# Protocolos y conectores

- Muchos servidores de bases de datos utilizan protocolos de comunicación específicos que facilitan el acceso a los mismos.
- Estas interfaces de alto nivel ofrecen facilidades para:
  - Establecer una conexión a una base de datos.
  - Ejecutar consultas sobre una base de datos.
  - Procesar los resultados de las consultas realizadas.

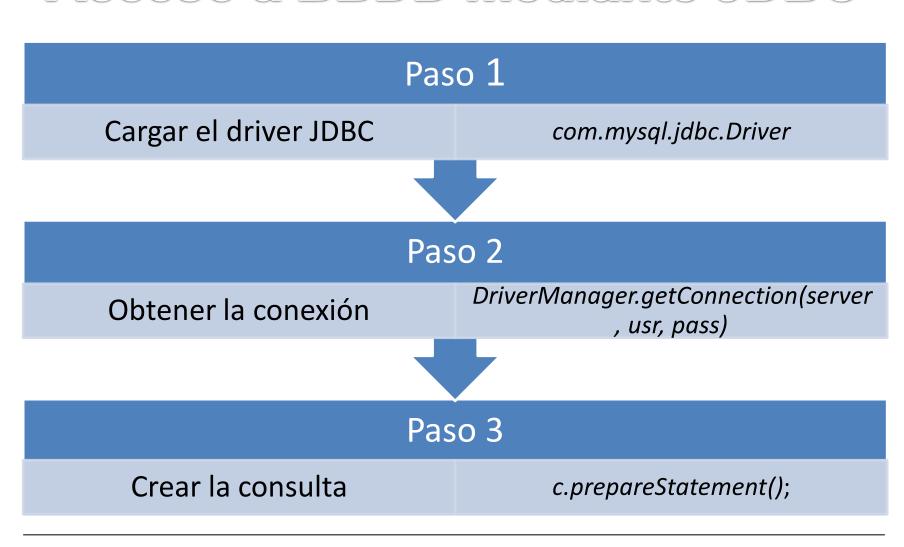
# Protocolos y conectores

- Al conjunto de clases encargadas de implementar la interfaz de programación de aplicaciones (API) y facilitar, con ello, el acceso a una base de datos se le denomina conector o driver.
- Cuando se construye una aplicación de base de datos, el conector oculta los detalles específicos de cada base de datos.
- La mayoría de fabricantes ofrecen conectores para acceder a sus bases de datos.
- Un ejemplo de conector muy extendido es el conector JDBC.

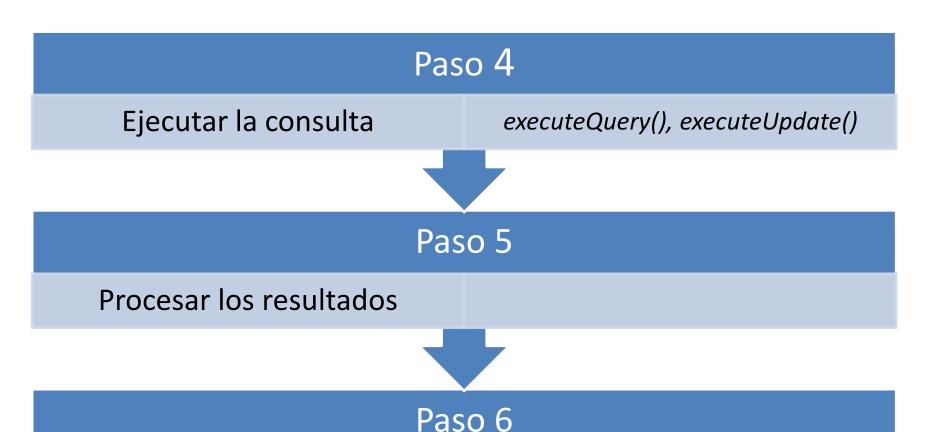
# Componentes JDBC

- El conector JDBC incluye cuatro componentes principales:
  - El propio API JDBC, que facilita el acceso desde el lenguaje de programación Java a bases de datos relacionales.
  - El gestor del conector JDBC (driver manager), que conecta una aplicación java con el driver correcto de JDBC.
  - La suite de pruebas JDBC, encargada de comprobar si un conector (driver) cumple con los requisitos JDBC.
  - El driver o puente JDBC-ODBC, que permite que se puedan utilizar los drivers ODBC como si fueran de tipo JDBC.

## Acceso a BBDD mediante JDBC



### Acceso a BBDD mediante JDBC



Liberar los recursos

close();

#### Clases básicas del API JDBC

Los objetos de la clase Connection
 ofrecen un enlace activo a una base de
 datos a través del cual un programa en
 Java puede leer y escribir datos, así como
 explorar la estructura de la base de datos
 y sus capacidades.

### Clases básicas del API JDBC

- Interfaz DriverManager, complementaria de la clase Connection, con ella se registran los controladores JDBC y se proporcionan las conexiones que permiten manejar las URL específicas de JDBC.
- La clase Statement proporciona los métodos para que las sentencias, utilizando el lenguaje de consulta (SQL), sean realizadas sobre la base de datos.

#### Clases adicionales del API JDBC

- Además de las clases básicas anteriores, el API
   JDBC también ofrece la posibilidad de acceder a los
   metadatos de una base de datos.
- Con ellos se pueden obtener información sobre la estructura de la base de datos, y gracias a ellos, se pueden desarrollar aplicaciones independientes del esquema que tenga la base de datos.
- Principales clases: DatabaseMetaData y ResultSetMetaData.

# Ejecución de sentencias de definición de datos

- Las principales sentencias asociadas con el lenguaje DDL son CREATE, ALTER y DROP.
- Para enviar comandos SQL a la base de datos con JDBC se usa un objeto de la clase Statement.
- Métodos interesantes de Statement: executeUpdate() y executeQuery().

# Ejecución de sentencias de manipulación de datos

- Las sentencias de manipulación de datos son las utilizadas para insertar, borrar, modificar y consultar los datos que hay en una base de datos.
- SELECT, INSERT, UPDATE y DELETE.

## Ejecución de consultas

- La ejecución de consultas sobre bases de datos desde aplicaciones Java descansa en dos tipos de clases disponibles en el API JDBC y en dos métodos.
- Las clases son: Statement y ResultSet y los métodos son: executeQuery y executeUpdate.

### **Clase Statement**

- Las sentencias Statement son las encargadas de ejecutar las sentencias SQL estáticas con Connection.createStatement().
- El método executeQuery() de Statement está diseñado para sentencias que producen como resultado un único resultado (ResultSet), como es el caso de las sentencias SELECT.

### **Clase Statement**

```
try {
   Statement stmt = con.createStatement();
   String query = "SELECT * FROM album WHERE titulo like 'B%';";
   ResultSet rs = stmt.executeQuery(query);
   while (rs.next()) {
      System.out.println("ID - " + rs.getInt("id") + ", Titulo " +
      rs.getString("titulo") + ", Autor " +
       rs.getString("autor"));
   rs.close();
   stmt.close();
} catch (SQLException ex) {
   // tratar el error
```

## Clase PreparedStatement

- Una variante a la sentencia *Statement* es la sentencia *PreparedStatement*.
- Se utiliza para ejecutar las sentencias SQL precompiladas.
- Permite que los parámetros de entrada sean establecidos de forma dinámica, ganando eficiencia.

## Clase PreparedStatement

```
try {
   String query = "SELECT * FROM album WHERE titulo like ?;";
   PreparedStatement pst = con.prepareStatement(query);
   pst.setString(1, "B%");
   ResultSet rs = pst.executeQuery(query);
   while (rs.next()) {
      System.out.println("ID - " + rs.getInt("id") + ", Titulo " +
      rs.getString("titulo") + ", Autor " +
       rs.getString("autor"));
   rs.close();
   pst.close();
} catch (SQLException ex) {
   // tratar el error
```

### Clase CallableStatement

- Otro tipo de sentencias son las asociadas a los objetos de la clase CallableStatement.
- Son sentencias preparedStatement que llaman a un procedimiento almacenado, es decir, métodos incluidos en la propia base de datos.
- No todos los gestores de bases de datos admiten este tipo de procedimientos

### Gestión de transacciones

- Una transacción en un SGBD es un conjunto de órdenes que se ejecutan como una unidad de trabajo.
- Una transacción se inicia cuando se encuentra una primera DML y finaliza cuando se ejecuta:
  - Un COMMIT o un ROLLBACK.
  - Una sentencia DDL, por ejemplo CREATE.
  - Una sentencia DCL, por ejemplo GRANT o REVOKE.

### Gestión de transacciones

- Una transacción que termina con éxito se puede confirmar con una sentencia COMMIT, en caso contrario puede abortarse utilizando la sentencia ROLLBACK.
- En JDBC por omisión cada sentencia SQL se confirma tan pronto se ejecuta, es decir, una conexión funciona por defecto en modo auto-commit. Para ejecutar varias sentencias en una misma transacción es preciso deshabilitar el modo auto-commit.

```
try {
 con.setAtutoCommit(false);
 Statement sta = con.createStatement();
 sta.executeUpdate("insert into album values (1, "uno","art1","1988");
 sta.executeUpdate("insert into cancion values("tit", "190", "letra", 1);
 con.commit();
} catch (SQLException ex) {
 System.out.println("Error al insertar".);
 try {
   con.rollback();
 catch (SQLException ex2) {
    ex2.printStackTrace();
```