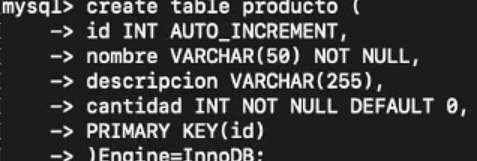
**Curso de  
Java y JDBC: Trabajando con una Base de Datos**

**INTRODUCCION A JDBC:**

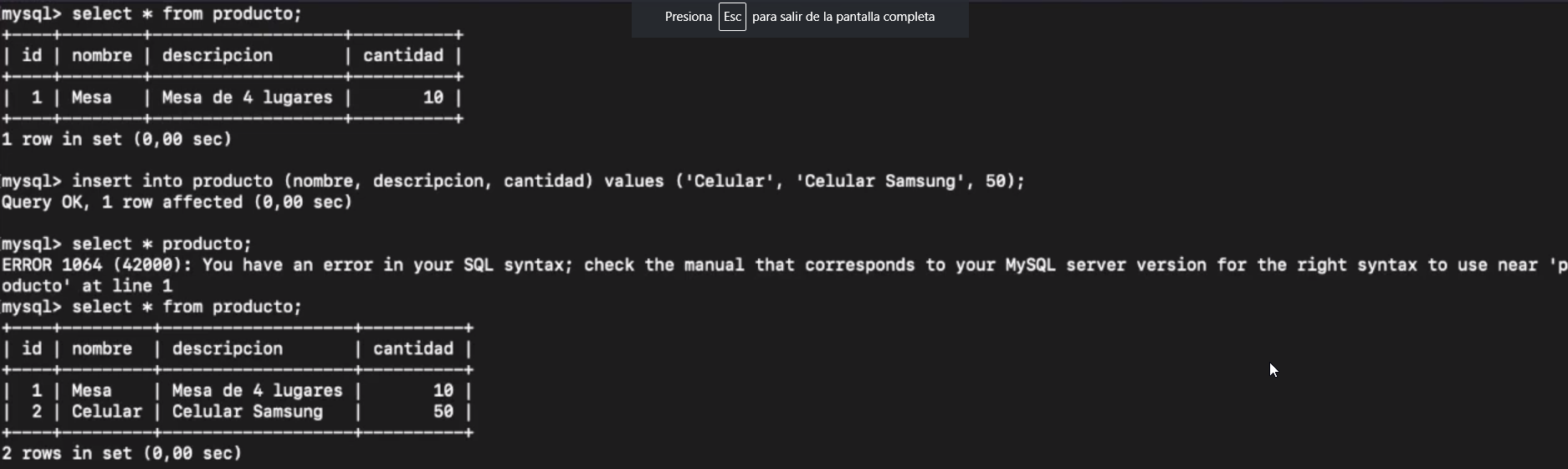
Entorno y versiones:



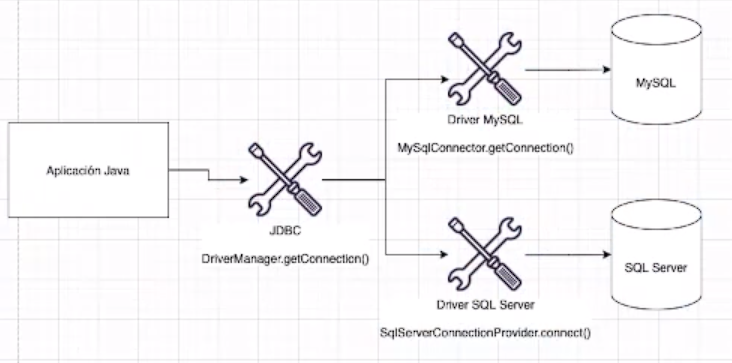


ENGINE INNODB = acepte transacciones en la tabla.

APLICACIÓN MANEJA EL CONTROL DE STOCK DE UNA TIENDA.



Como conectar una aplicación a base de datos:



JAVA DATA BASE CONECTIBITY



Los primeros pasos con JDBC:



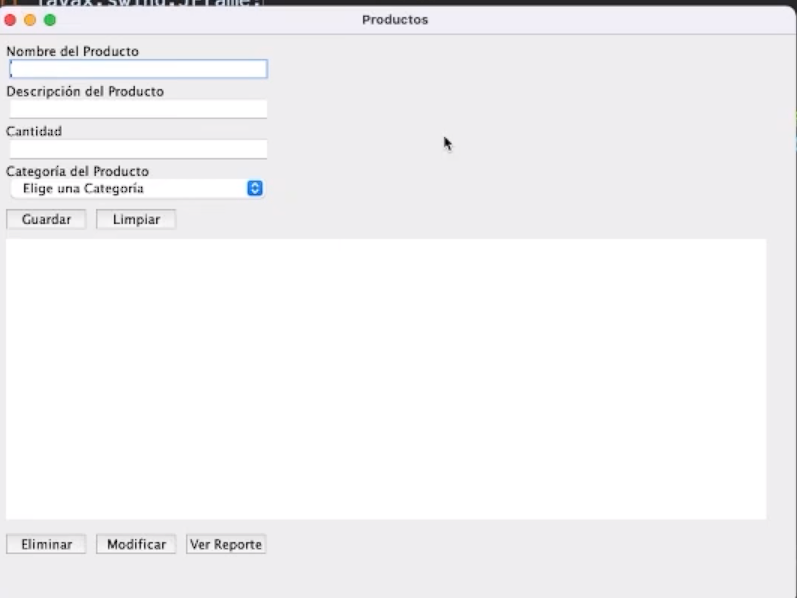
Lo que Aprendimos:

Lo que aprendimos en esta aula:

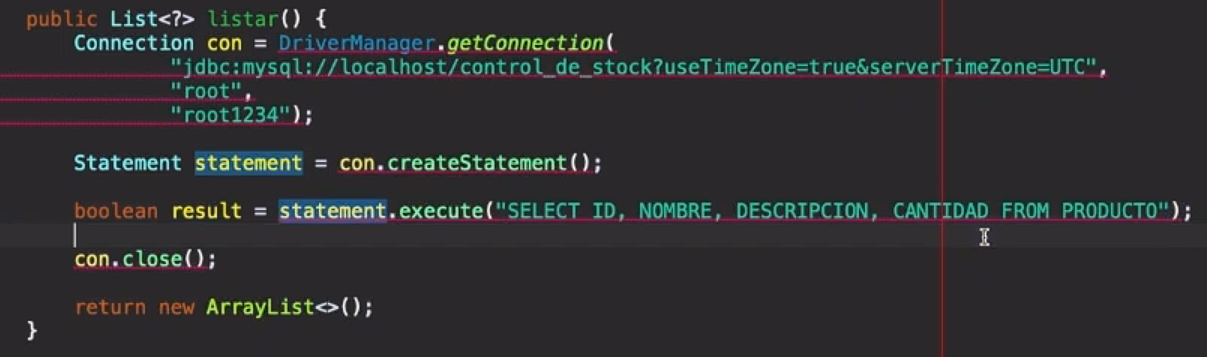
* Para acceder a una base de datos necesitamos del **driver de conexión**;
  + Un driver es simplemente una librería **.jar**.
* **JDBC** significa *Java DataBase Connectivity*;
  + El JDBC define una capa de abstracción entre la aplicación y el driver de la base de datos.
  + Esta capa es compuesta de interfaces que el driver implementa.
* Para abrir una conexión con la base de datos debemos utilizar el método getConnection de la clase DriverManager;
  + El método getConnection recibe tres parámetros. Son ellos la URL de conexión JDBC, el usuario y la contraseña.

**EJECUTANDO COMANDOS SQL EN JAVA:**

Conociendo la view de la aplicación:

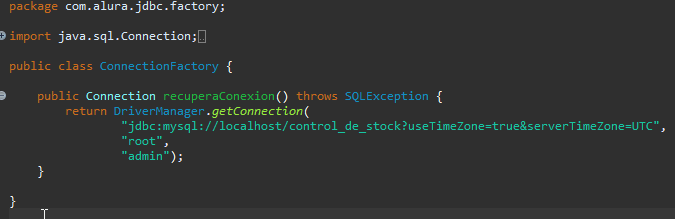


Listado con Statement:

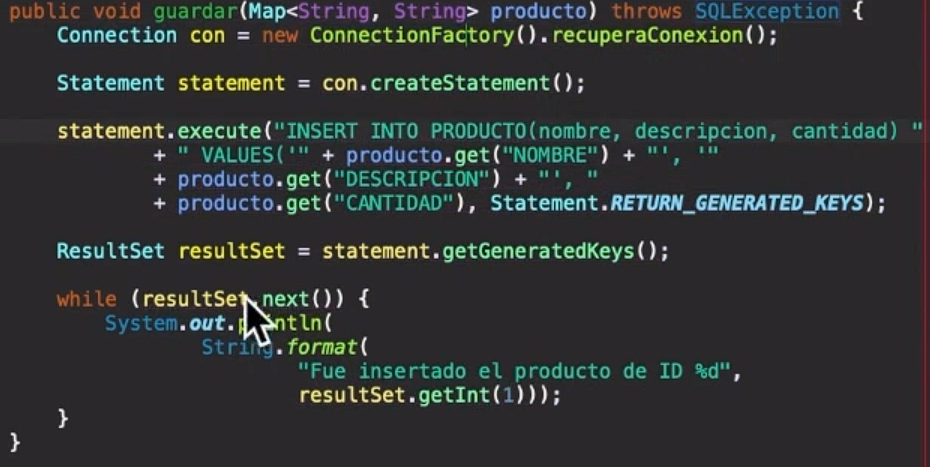


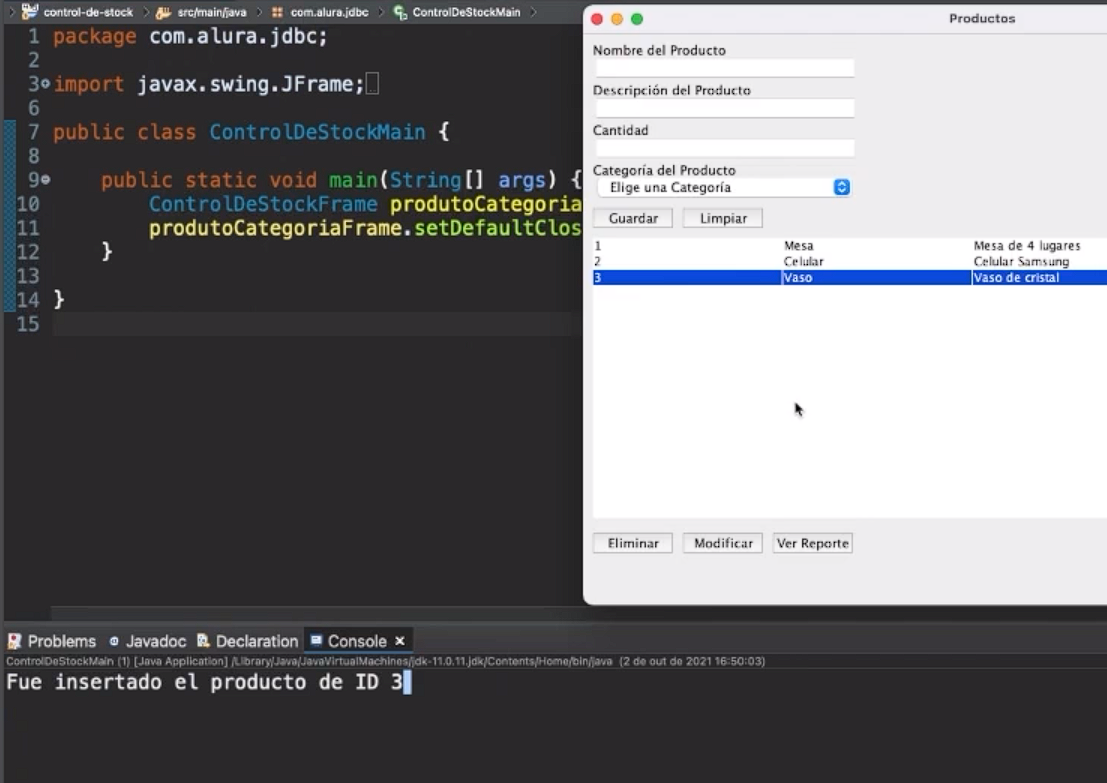
Si es listado o no.

Creando la ConnectionFactory:

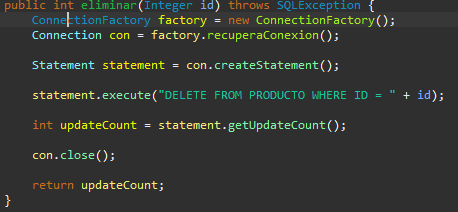


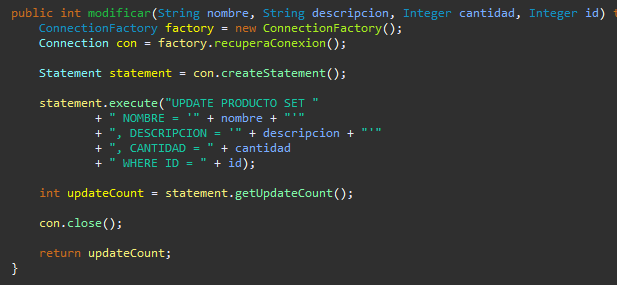
INSERT con Statement:



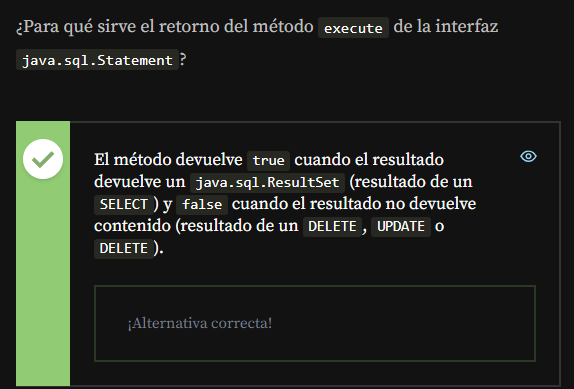


DELETE con Statement:





Lo que Aprendimos:

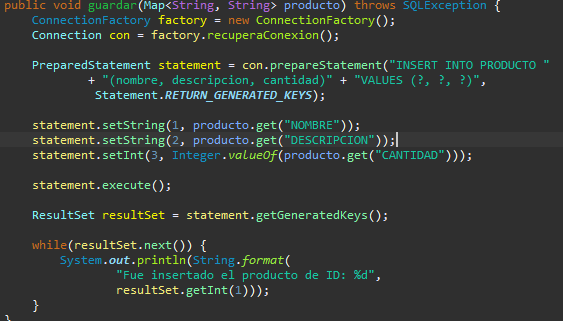


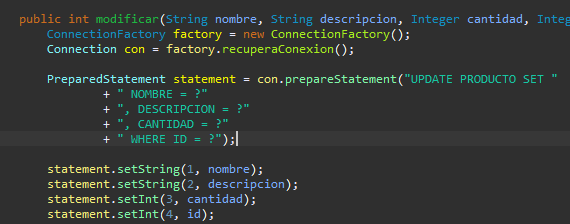
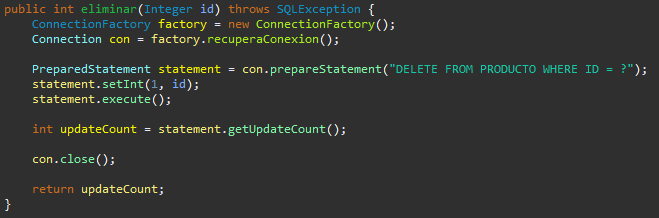
Lo que aprendimos en esta aula:

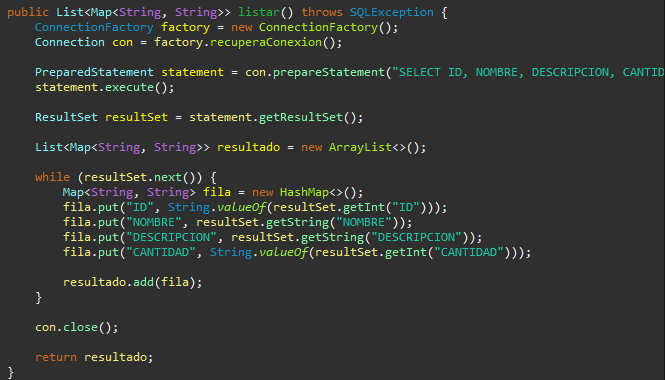
* Para simplificar y encapsular la creación de la conexión debemos utilizar una clase ConnectionFactory;
  + Esta clase sigue el estándar de creación *Factory Method*, que encapsula la creación de un objeto.
* Podemos utilizar la interfaz java.sql.Statement para ejecutar un comando SQL en la aplicación;
  + El método execute envía el comando para la base de datos.
  + A depender del comando SQL, podemos recuperar la clave primaria o los registros buscados.

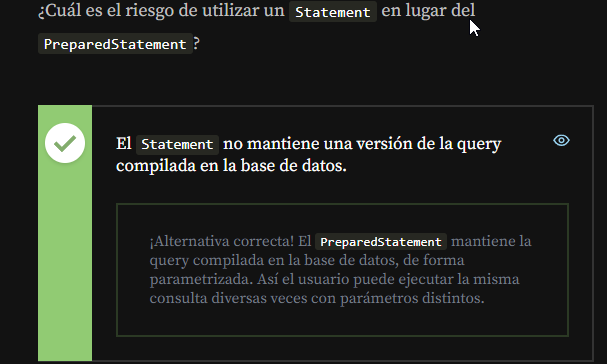
**REALIZANDO MEJORAS EN EL PROYECTO:**

Evitando SQL injection utilizando PreparedStatement:

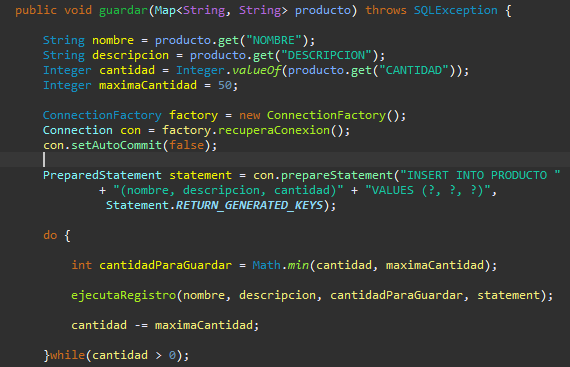


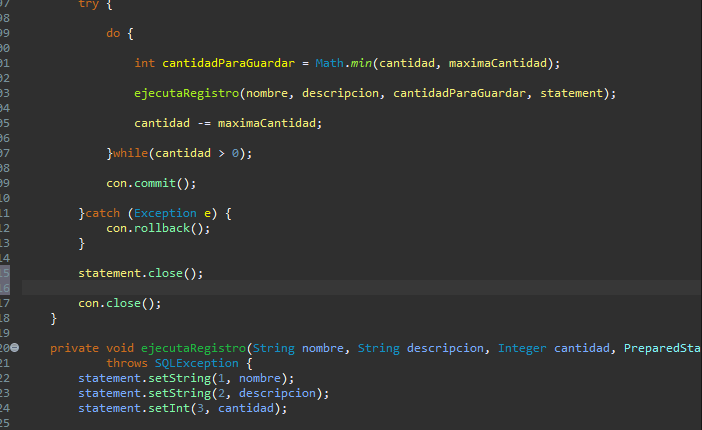




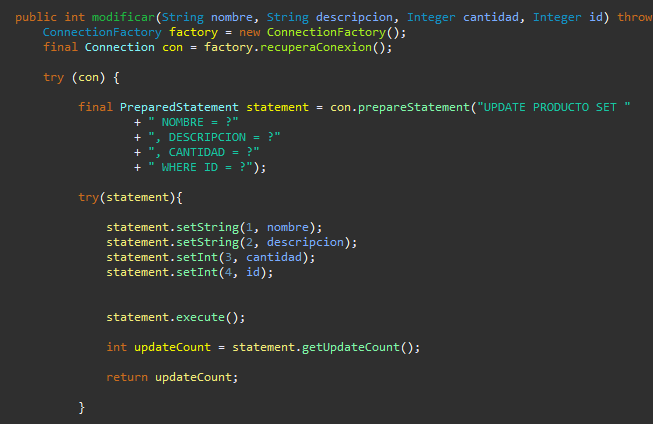
Tomando el control de la transacción:

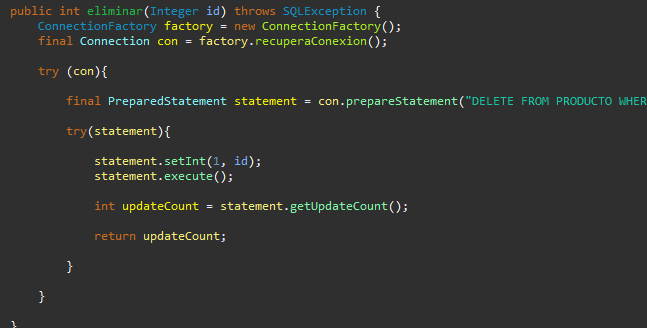


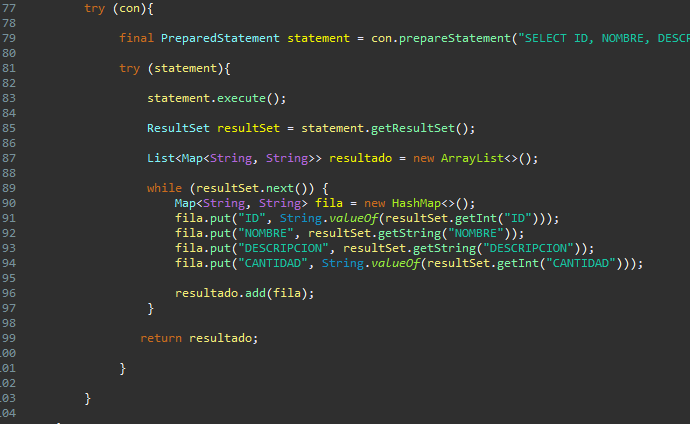
Manejando el Commit y el rollback:



Utilizando Try-whit-resources:









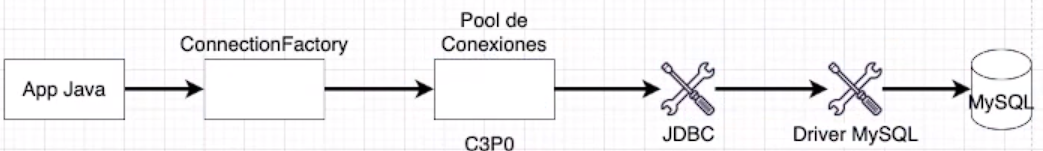
Lo que Aprendimos:

Lo que aprendimos en esta aula:

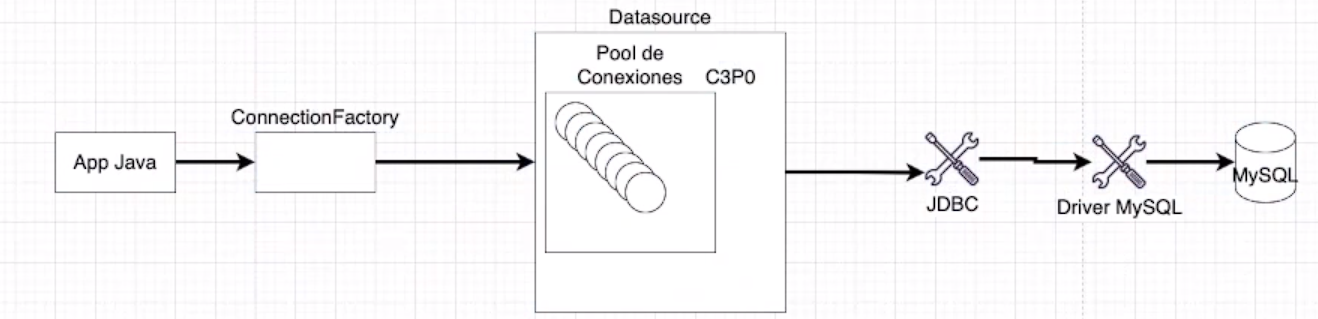
* Cuando ejecutamos una query SQL como Statement tenemos un riesgo de seguridad llamado **SQL Injection**;
  + **SQL Injection** es el hecho de enviar comandos SQL como parámetro de las solicitudes en una aplicación.
* Para evitar el fallo por **SQL Injection** debemos utilizar la interfaz PreparedStatement;
  + Diferente del Statement, el PreparedStatement trata los parámetros del comando SQL para que caracteres y comandos especiales sean tratados como strings.
* Las bases de datos ofrecen un recurso llamado **transacción**, que junta muchas operaciones SQL como un conjunto de ejecución;
  + Si el conjunto falla no es aplicada ninguna modificación y ocurre el *rollback* de la transacción.
  + Todos los comandos del conjunto necesitan funcionar para que la transacción sea finalizada con un *commit*.
* Para garantizar el cierre de los recursos abiertos en el código, Java provee un recurso llamado *try-with-resources* para ayudarnos;
  + Para utilizar este recurso es necesario que la clase utilizada (como la Connection) implemente la interfaz Autocloseable.

**ESCALABILIDAD CON POOL DE CONEXIONES:**

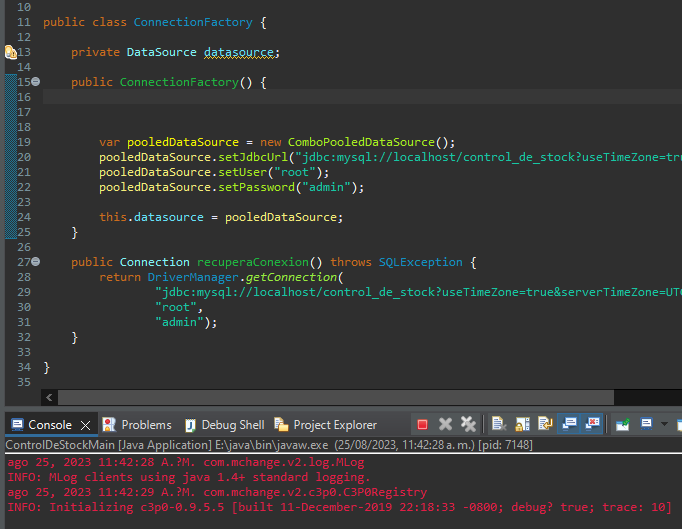
Que es un pool y un datasource:



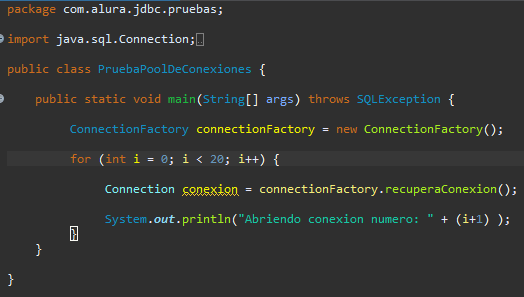
DATA SOURCE = INTERFAZ QUE ASTRAE LA IMPLEMENTACION DEL POOL DE CONEXIONES PARA NOSOTROS.

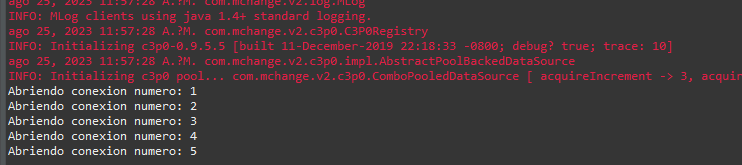


Creando un pool de conexiones:



Probando el pool de conexiones:





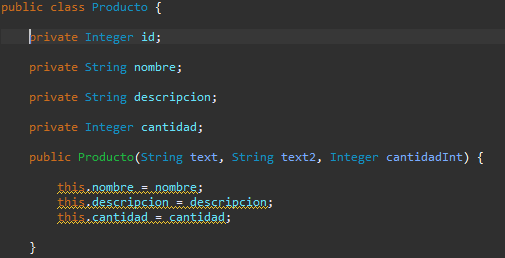
Lo que Aprendimos:

Lo que aprendimos en esta aula:

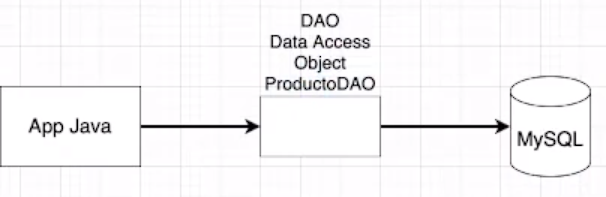
* Utilizar el ***pool de conexiones*** es una buena práctica;
* El pool de conexiones controla la cantidad de conexiones abiertas entre la aplicación y la base de datos;
  + Es normal que haya un mínimo y un máximo de conexiones.
* De la misma forma que hay, en JDBC, una interfaz para representar la conexión (java.sql.Connection), también hay una interfaz que representa el pool de conexiones (javax.sql.DataSource);
* **C3P0** es una implementación Java de un pool de conexiones.

**CAPA DE PERSISTENCIA CON DAO:**

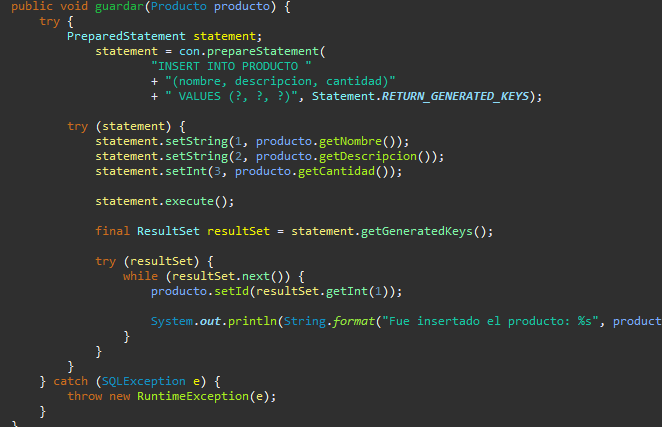
Creando el modelo del producto:

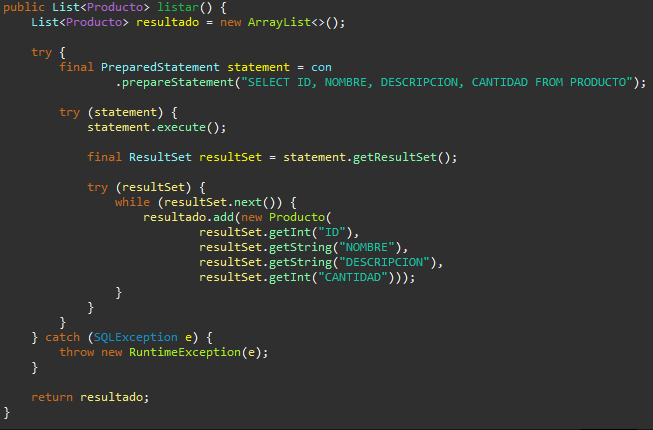


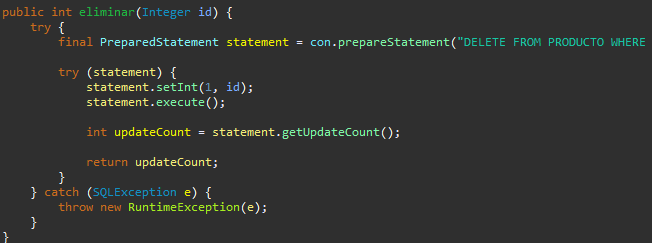
DAO con INSERT del producto:

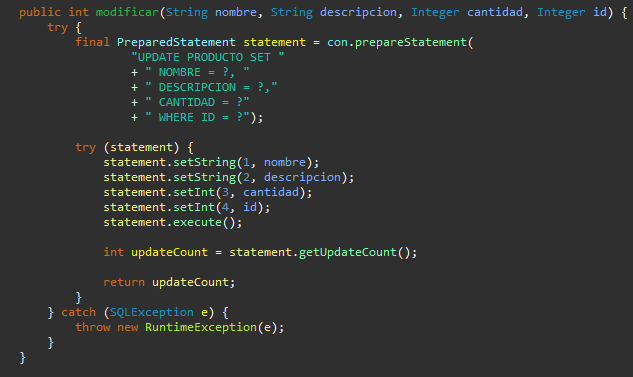


Operación de listado en ProductoDAO:









El estándar MVC:

Controller, une a Vista y Modelo.

Lo que Aprendimos:

Lo que aprendimos en esta aula:

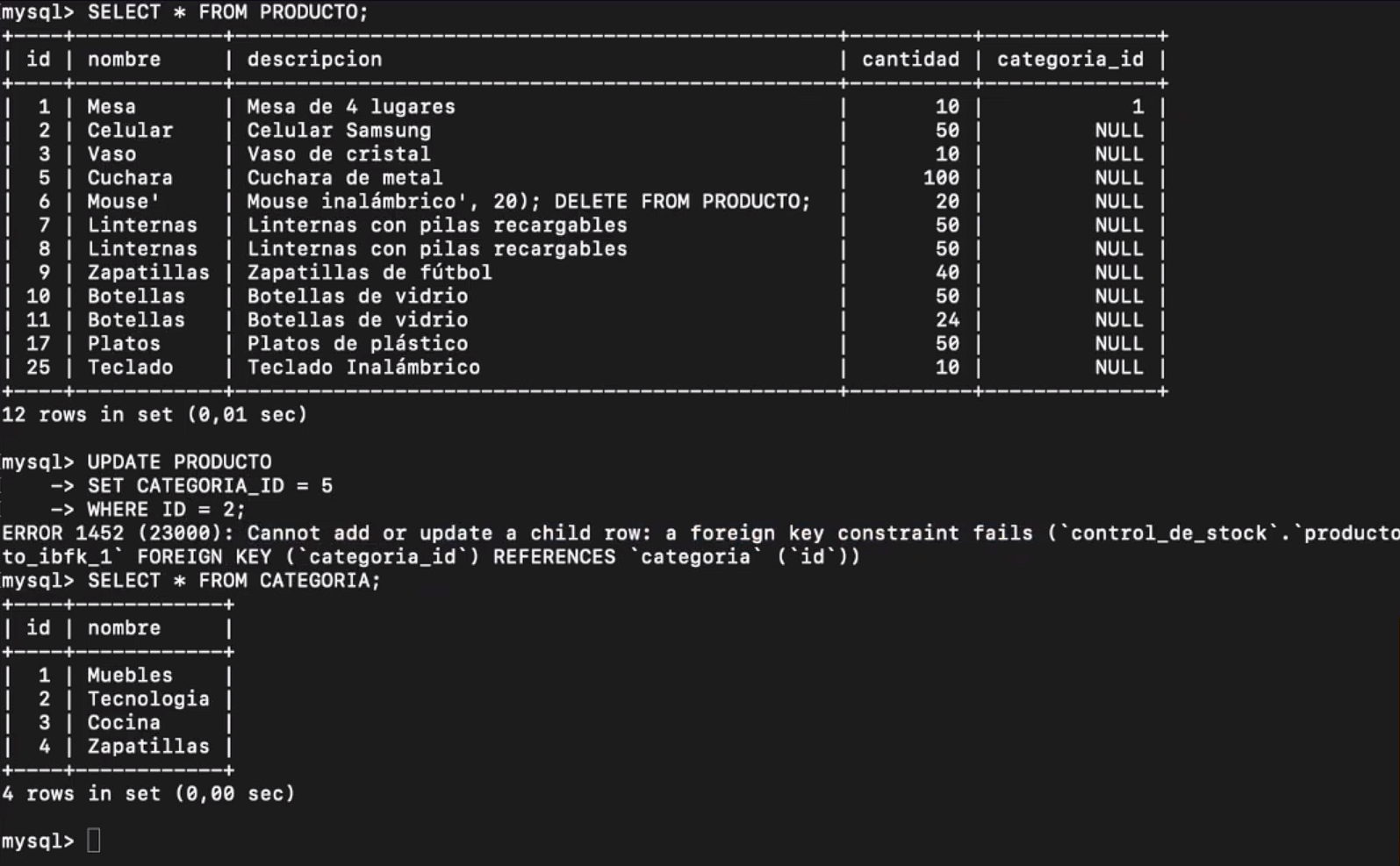
* Para cada tabla del modelo tenemos una clase de dominio;
  + Para la tabla de producto tenemos una clase Producto asociada.
  + Los objetos del tipo Producto representan un registro de la tabla.
* Para acceder a la tabla vamos a utilizar el estándar llamado ***Data Access Object*** (**DAO**);
  + Para cada clase de dominio hay un DAO asociado. Por ejemplo, la clase Producto posee la clase ProductoDAO.
  + Todos los métodos JDBC relacionados al producto deben estar encapsulados en ProductoDAO.
* Una aplicación es escrita en capas;
  + Las capas más conocidas son las de *view*, *controller*, *modelo* y *persistencia*, que componen el estándar MVC.
* El flujo de una requisición entre las capas es el siguiente;

view <--> controller <--> persistenciaCOPIA EL CÓDIGO

* En este curso utilizamos una aplicación con las *views* y los *controllers* ya creados y enfocamos en la capa de persistencia y modelo;
* No es una buena práctica dejar los detalles de implementación de una capa en otras que no tienen esta responsabilidad (por ejemplo la capa de *controller* lanzar una SQLException);
* Aquí estamos aprendiendo con una aplicación desktop embebida, pero hay otros tipos de aplicaciones con otros tipos de view, como **html** para aplicaciones web.

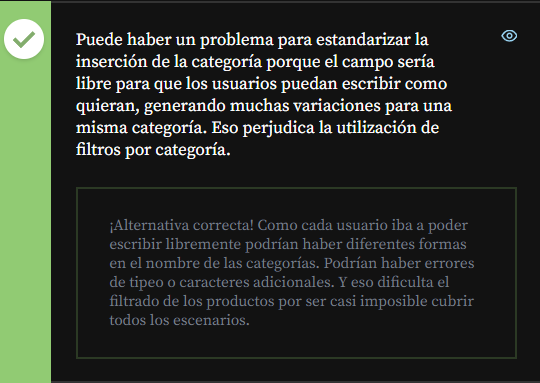
**EVITANDO QUERIES N +1:**

Relación entre tablas:

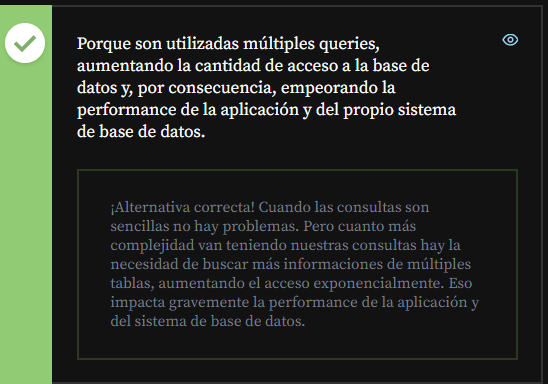


El modelo, el controller y el DAO de categorías:

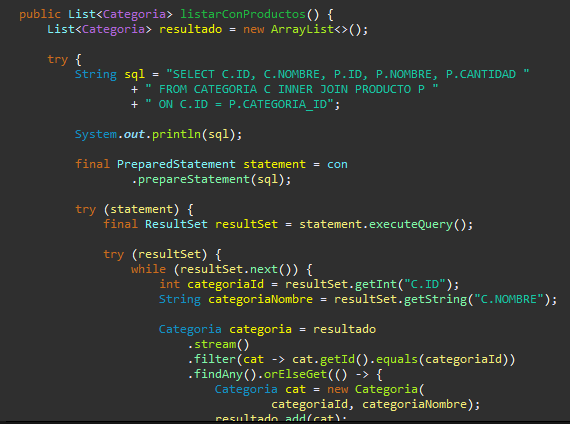
Relacionando el producto con la categoría del registro:

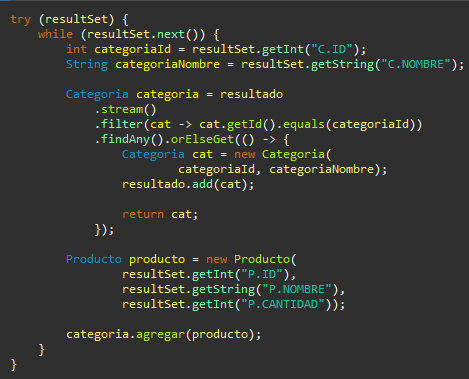


Queries N + 1:



Utilizando INNER JOIN:





Lo que Aprendimos:

Lo que aprendimos en esta aula:

* Cuando tenemos una relación entre *tables* debemos tener cuidado para no crear el problema de queries N + 1;
  + N + 1 quiere decir que, para buscar los datos de una relación, es ejecutada una query y luego una más por cada relación.
  + Este tipo de problema puede generar problemas de performance en la aplicación y en la base de datos.
  + Este tipo de problema puede ser evitado utilizando *join* en la query SQL.