Acceso a datos

Actividad UT 02
Acceso a base de datos SQL con JDBC

CONTENIDO

1.- Objetivos

- Familiarizarse con Git y el trabajo en equipo en el contexto de desarrollo de software.
- Recordar y practicar el acceso a datos con JDBC aprendido en primer curso.
- Practicar las nuevas técnicas de acceso a datos aprendidas:
  - Transacciones
  - Connection Pooling
- Aplicar nuevos conceptos y patrones de diseño de software como:
  - Patrón singleton
  - Principios SOLID
  - Arquitectura n-capas

2.- Visión general de la actividad

La actividad está enmarcada dentro de la UT 02 (Manejo de conectores). Además de los conceptos de esta unidad, se aplicarán conceptos generales de diseño de software.

En esta actividad se creará una arquitectura de acceso a datos basada en el patrón n-capas, con las siguientes 3 capas:

- Capa de aplicación. Programa principal para probar el resto de las capas.
- Capa de entidades. Contendrá todas las clases para el mapeo de las tablas de la base de datos.
- Capa de servicios. Capa con la lógica principal de negocio del sistema.
- Capa de acceso a datos. Capa que se encargará de las operaciones de acceso a datos.

Estas capas se implementarán como diferentes paquetes en la aplicación Java.

Para esta actividad usaremos la base de datos “ClassicModels”. Se trata de una base de datos de un ficticio negocio de venta de reproducciones a escala de coches, aviones, motocicletas, etc. Como parte de la actividad se tendrá que configurar esta base de datos en MariaDB, creando el esquema, importando los datos y creando un usuario con los permisos adecuados.

Esta actividad se ampliará más adelante, en otra actividad dentro de la UT 01, para trabajar el manejo de ficheros XML en Java.

3.- Equipos de trabajo

Esta actividad se realizará preferentemente en equipos, teniendo en cuenta las siguientes consideraciones:

- Los equipos podrán ser de dos o tres alumnos.
- Excepcionalmente, se permitirá realizar la práctica en solitario, siempre que se argumente un motivo razonable.
- El profesor no organizará los grupos, pero se reserva el derecho de ajustar la composición de algún grupo si lo considera necesario, siempre para favorecer el equilibrio dentro de los grupos en función del nivel de los integrantes.
- El número de tareas a realizar en la actividad dependerá del tamaño del equipo de trabajo. Los equipos de tres personas tendrán que realizar más trabajo que los de dos, y estos más que los que realicen la actividad en solitario. El objetivo es intentar que todos los alumnos realicen una cantidad de trabajo similar.
- Todos los equipos, incluso los unipersonales, deberán crear repositorios privados en GitHub para mantener el progreso de la actividad, y deberán dar permisos de contribución al profesor, para que pueda revisar el repositorio siempre que lo considere necesario.

4.- Requisitos previos

- Instancia de MySQL o MariaDB donde poder crear el esquema de la nueva base de datos. Cada alumno debe disponer de su instancia de la BD.
- El usuario para acceder al esquema "classicmodels" debe ser el mismo, con la misma contraseña, para todos los miembros del grupo. Se recomienda usar como usuario y contraseña "classicmodels", para que todo el mundo use los mismos datos.
- IntelliJ Idea (recomendado) u otro IDE a elección del alumno. Todos los integrantes de un grupo deben usar el mismo entrono de desarrollo, para evitar problemas al colaborar.

5.- Evaluación y entrega

La evaluación se realizará por observación en el aula. Una vez que el equipo considere que la actividad está completada, informará al profesor, que pedirá a los alumnos que realicen una serie de pruebas, y realizará algunas preguntas sobre el código, forma de trabajo, dificultades encontradas o cualquier otro asunto relacionado con la actividad. Se valorarán cosas como

- Ajuste a los requisitos de la actividad. ¿Hace lo que se pedía? ¿Tiene errores?
- Estructura y claridad, uso de convenciones y estándares, elección de nombres de paquetes, clases, variables, métodos, constantes, etc.
- Aplicación correcta de las técnicas, patrones, o arquitecturas requeridas.
- Documentación: JavaDoc y comentarios cuando sean necesarios.

Para poder registrar la calificación, los alumnos entregarán el código fuente de la actividad descargándolo de GitHub en formato ZIP. Todos los miembros del equipo deben realizar la entrega del código.

Además, el profesor podrá acceder en cualquier momento al repositorio para comprobar el avance del grupo, incluso antes de las fechas límite, para verificar que ha habido actividad de todos los usuarios.

La calificación será la misma para todos los integrantes del grupo, salvo casos evidentes en los que alguno de los integrantes no haya colaborado lo suficiente, en cuyo caso el profesor podrá calificar con menos nota a ese miembro del equipo.

6.- Preparación del entorno de desarrollo y del entorno de colaboración

6.1.- Creación de grupos

En clase, se formarán los grupos, y el profesor anotará los componentes de cada grupo, y les asignará un número, del 01 en adelante. Hasta que no se hayan formado los grupos no se podrá avanzar con la actividad, ya que la creación del repositorio en GitHub depende de la numeración asignada por el profesor.

6.2.- Registro en GitHub, creación de repositorio privado y asignación de permisos

Todos los alumnos deben disponer de un usuario en GitHub. Si un alumno no tiene usuario en GitHub, debe registrarse. Se puede utilizar tanto el correo electrónico de EducaMadrid como su correo personal.

Una vez registrados, uno de los miembros de cada grupo (sólo uno) debe crear un repositorio PRIVADO para la actividad. Este repositorio se seguirá usando a lo largo del curso para el resto de las actividades, así que se nombrarán en función del grupo.

El repositorio se debe llamar DM2E-AD-GRUPOXX, siendo XX el número del grupo: 01, 02, 03, etc.

Una vez creado el repositorio, el propietario añadirá como colaboradores a los miembros de su equipo y al profesor.

6.3.- Clonado del repositorio por cada miembro del equipo

Cada miembro del equipo debe clonar el repositorio en su máquina local. Si el alumno tiene más de una máquina (el PC del centro y su máquina en casa) podrá clonar el repositorio en tantas máquinas como crea necesario.

6.4.- Creación de un proyecto Java para el desarrollo de la actividad

La creación del proyecto Java debe realizarse en grupo en el aula. Uno solo de los miembros del equipo debe crear un proyecto Java con IntelliJ (o el entorno que se considere adecuado), en el repositorio creado para la actividad. Este proyecto se subirá (push) al repositorio, y así estará disponible para el resto de los miembros del equipo, que podrán obtenerlo actualizando (pull) el repositorio.

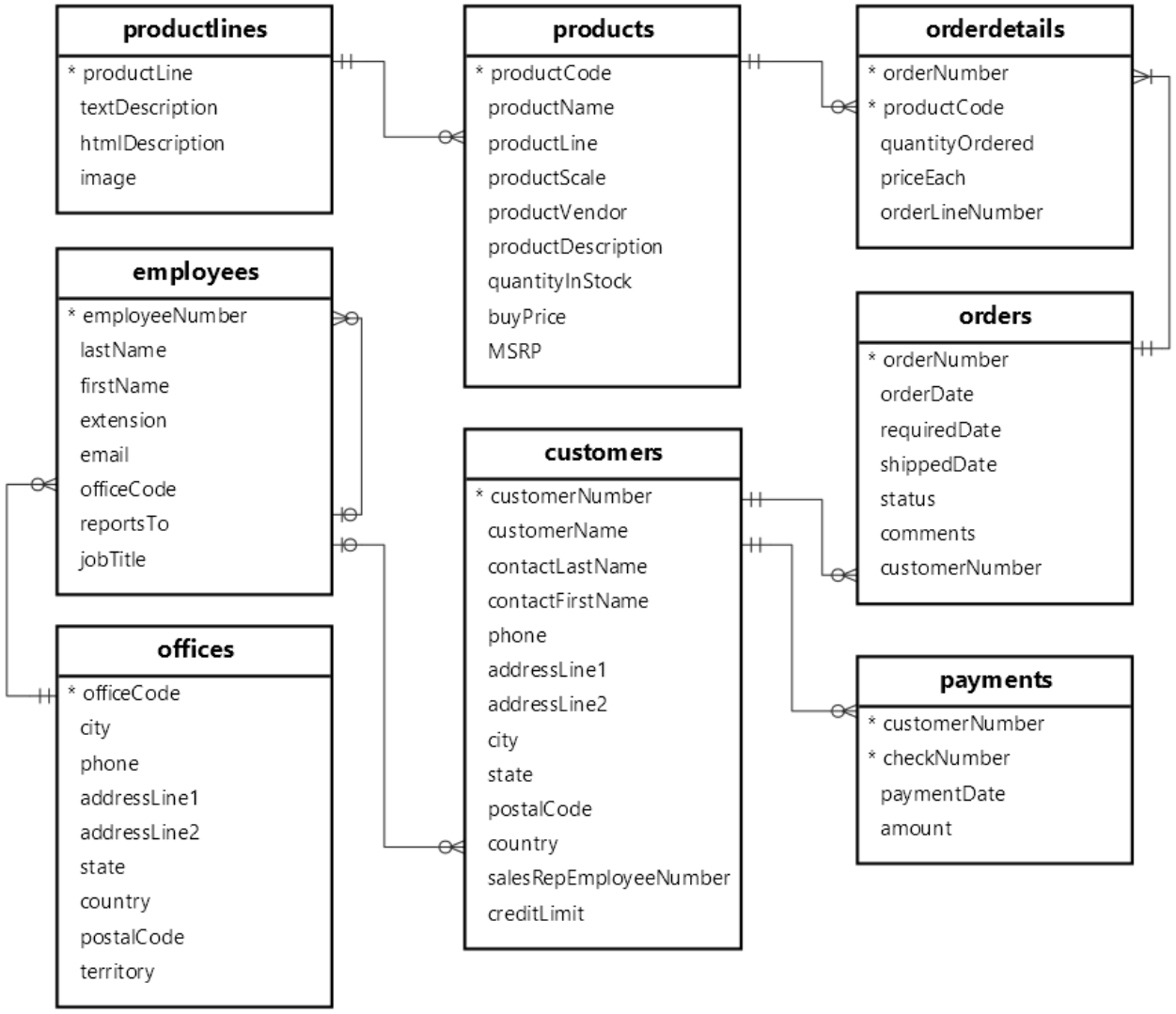
6.5.- Creación de esquema e importación de la base de datos

Esta tarea la deben realizar todos los miembros del equipo, en todas las máquinas en las que vayan a realizar desarrollo. Si alguien usa un equipo del centro, y además otro en su casa, tendrá que preparar la base de datos en ambos equipos, si quiere usar los dos para desarrollo.

Para importar la base de datos (los comandos deben ejecutarse con un usuario con suficientes privilegios):

- Crear un esquema “classicmodels” en el servidor de MariaDB / MySQL.
- Ejecutar el fichero de inicialización de la base de datos “classicmodels.sql”, que se encontrará en el aula virtual.
- Crear un usuario “classicmodels”, con contraseña “classicmodels” en el servidor SQL, y concederle permisos de selección, inserción, actualización, borrado y ejecución, para todos los objetos en el esquema “classicmodels” en el servidor local.
- Verificar que se han creado correctamente las siguientes tablas, y el número de filas indicado en cada una de ellas:
    - Customers: Clientes – 122 filas
    - Products: Productos – 110 filas
    - ProductLines: Líneas de producto (categorías) – 7 filas
    - Orders: Pedidos – 326 filas
    - OrderDetails: Productos en un pedido – 2996 filas
    - Payments: Pagos – 273 filas
    - Employees: Empleados y relaciones entre ellos (jefe/subordinado)– 23 filas
    - Offices: Tiendas– 7 filas

El diagrama de la base de datos es el siguiente:



6.5.- Lectura de requisitos y reparto de tareas

El equipo, también reunido, debe analizar las tareas de desarrollo y los requisitos reflejados en el punto 7 de este documento. Una vez analizadas, se debe repartir el trabajo de forma equitativa entre los distintos miembros.

7.- Desarrollo de la aplicación

En este punto se puede comenzar a desarrollar la aplicación, siendo cada miembro del equipo responsable del desarrollo de sus tareas asignadas.

Los miembros del equipo deben apoyarse entre sí. Si uno tiene problemas para desarrollar una tarea, puede pedir ayuda:

- Primero, al resto de los miembros del equipo. La idea es que seáis lo más autónomos posible.
- Segundo, al profesor. Antes de preguntar, intentad buscar la solución por vuestra cuenta. No obstante, podéis consultar con el profesor la solución planteada, para verificar que es la más adecuada.

Los cambios en la aplicación deben subirse (push) regularmente a GitHub, y se debe hacer un pull también de forma regular. No cometer el error de tener todo el desarrollo asignado en local y hacer push el último día, y no hacer pull de los compañeros, por varios motivos:

- Si no se actualiza frecuentemente (pull) el repositorio local, no tendremos los cambios de los compañeros. Si no se tienen estos cambios, puede que algo que "rompe" nuestra parte del desarrollo aparezca de repente y luego sea tarde para solucionarlo antes de la entrega.
- Si no hacemos push de nuestro código con frecuencia nuestros compañeros no tendrán la última versión, y pueden verse afectados por el mismo problema que el punto anterior, pero por una causa diferente.
- El repositorio en Git sirve también como copia de seguridad. Ante la perdida de datos por cualquier clase de accidente, si hemos hecho push frecuentemente la pérdida de trabajo será menos importante.

El objetivo es crear una arquitectura n-capas para acceso a datos, aplicando conceptos de diseño de software como el patrón singleton o los principios SOLID.

La idea es que, para ciertas tablas de la base de datos se creen una serie de clases:

- Clases de entidad (capa de entidades, paquete "entities"), que serán la clases POJO que servirán para mapear las filas de las tablas a objetos:
  - Para la tabla "products", la entidad "Product" – Todos los equipos
  - Para la tabla "orders", la entidad "Order" – Todos los equipos
  - Para la tabla "customers", la entidad "Customer" – Solo equipos de 2 y 3 personas
  - Para la tabla "employees", la entidad "Employee" – Solo equipos de 3 personas
- Interfaces para clases de acceso a datos (capa de acceso a datos, paquete "dataaccess"):
  - Para la entidad "Product", la interfaz "ProductDataAccess" – Todos los equipos
  - Para la entidad "Order", la interfaz "OrderDataAccess" – Todos los equipos
  - Para la entidad "Customer", la interfaz "CustomerDataAccess" – Solo equipos de 2 y 3 personas
  - Para la entidad "Employee", la interfaz "EmployeeDataAccess" – Solo equipos de 3 personas
- Clases para acceso a datos (capa de acceso a datos, paquete "dataaccess"), una por cada entidad, que implementan las interfaces de acceso a datos anteriormente mencionadas:
  - Para la entidad "Product", la clase "ProductDataAccessImpl" – Todos los equipos
  - Para la entidad "Order", la clase "OrderDataAccessImpl" – Todos los equipos
  - Para la entidad "Customer", la clase "CustomerDataAccessImpl" – Solo equipos de 2 y 3 personas
  - Para la entidad "Employee", la clase "EmployeeDataAccessImpl" – Solo equipos de 3 personas

- Interfaces de servicios (capa de servicios, paquete “services”), una por cada entidad:
  - Para la entidad “Product”, la interfaz “ProductService” – Todos los equipos
  - Para la entidad “Order”, la interfaz “OrderService” – Todos los equipos
  - Para la entidad “Customer”, la interfaz “CustomerService” – Solo equipos de 2 y 3 personas
  - Para la entidad “Employee”, la interfaz “EmployeeService” – Solo equipos de 3 personas
- Clases de servicios (capa de servicios, paquete “services”), una por cada entidad, que implementan las interfaces de servicios anteriormente mencionadas:
  - Para la entidad “Product”, la clase “ProductServiceImpl” – Todos los equipos
  - Para la entidad “Order”, la clase “OrderServiceImpl” – Todos los equipos
  - Para la entidad “Customer”, la clase “CustomerServiceImpl” – Solo equipos de 2 y 3 personas
  - Para la entidad “Employee”, la clase “EmployeeServiceImpl” – Solo equipos de 3 personas
- Clase de pruebas (capa de aplicación, paquete “application”), una por entidad:
  - Para la entidad “Product”, la clase “ProductTests” – Todos los equipos
  - Para la entidad “Order”, la clase “OrderTests” – Todos los equipos
  - Para la entidad “Customer”, la clase “CustomerTests” – Solo equipos de 2 y 3 personas
  - Para la entidad “Employee”, la clase “EmployeeTests” – Solo equipos de 3 personas
- Programa principal (capa de programa, paquete “program”), un programa principal que instancie las clases de pruebas para lanzar las pruebas.

Además, se implementará un pool de conexiones como singleton, usando HikariCP. Este pool de conexiones se ubicará dentro del paquete de la capa de acceso a datos.

Las relaciones de dependencia entre las clases serán la siguientes:

- Las clases de la capa de entidades no pueden depender de ninguna otra capa.
- La capa de acceso a datos dependerá sólo de la capa de entidades, y de las clases de la API Java necesarias para el acceso a datos. Si es necesario, se pueden establecer dependencias entre las clases de la capa de acceso a datos. Por ejemplo, las clases “xxxDataAccessImpl” dependerán del pool de conexiones.
- La capa de servicios dependerá sólo de la capa de entidades, y de interfaces de la capa de acceso a datos. Específicamente, hay que evitar que haya dependencias de detalles de bajo nivel, como, por ejemplo, dependencia del pool de conexión o dependencia del paquete java.sql.
- Las clases de pruebas de la capa de aplicación dependerán sólo de la capa de entidades, y de interfaces de la capa de servicios. Específicamente, hay que evitar que haya dependencias de detalles de bajo nivel, como, por ejemplo, dependencia del pool de conexión o dependencia del paquete java.sql, o de la capa de acceso a datos.
- El programa principal podrá depender de todo aquello que sea necesario. Es posible que, para realizar la inyección de dependencias, tenga que depender de todas las capas, o de muchas de ellas.

7.1.- Métodos estándares de capa de datos

Cada interfaz / clase de la capa data access (salvo el pool de conexiones) debe tener los siguientes métodos (ninguno estático):

- Constructor en el que reciba las dependencias necesarias, si es que las hay. Las dependencias, siempre que sea posible, se inyectarán como interfaces.
- long count() – Cuenta el número total de entidades en la tabla.
- boolean existsById(ID id) – Verifica si una entidad con un ID dado existe.
- Optional<T> findById(ID id) – Busca una entidad por su ID. Si no existe devuelve Optional vacío.
- List<T> findAll() – Recupera todas las entidades.
- T save(T entity) – Guarda una entidad. Si la entidad ya existe, la actualiza. Recibe la entidad que se va a guardar. Devuelve la entidad guardada, con los datos actualizados (id si es nueva)

- void deleteById(ID id) – Elimina la entidad con el ID especificado.

7.2.- Método específico de OrderDataAccess

La interfaz OrderDataAccess:

- No tendrá método “save”, y, por tanto, la clase que implemente OrderDataAccess no implementará el método.
- Tendrá un método “create” que recibirá un objeto de la clase “CreateOrderDto”, que se debe crear en un paquete independiente “dto”. Esta clase “CreateOrderDto” debe tener los datos necesarios para poder crear un pedido completo. Esto es, los datos necesarios para completar todas las tablas y filas relacionadas con el pedido: tablas orders (1 registro) y orderdetails (un registro por producto adquirido). En los datos también se necesitará, al menos, el ID del cliente.
- El método “Create”, como implica modificar datos en dos tablas, debe utilizar transacciones, para que, si ha un error, no se quede un pedido a medio completar.
- No devolverá nada. Será void.

7.3.- Métodos de capa de servicios

Cada clase de la capa de servicios debe tener:

- Un constructor que reciba las dependencias necesarias siempre que sea posible en forma de interfaz. Recibirá como dependencia, al menos, de una interfaz de la clase de capa de datos correspondiente. Por ejemplo, CustomerServiceImpl recibirá una instancia de la interfaz CustomerDataAccess, que se implementa con la clase CustomerDataAccessImpl
- Un método (no estático) por cada uno de los métodos de la capa de acceso a datos. Estos métodos llamarán a los métodos de la capa de acceso a datos.

7.3.- Métodos de las clases de pruebas, de la capa de aplicación

Cada clase de pruebas debe tener métodos para probar la clase de servicios (que a su vez llamará a la capa de acceso a datos). Estos métodos se llamarán desde la clase del programa principal.

7.4.- Programa principal

El programa principal debe tener un menú para poder probar TODOS los métodos de la aplicación. La implementación de este menú queda a la elección del grupo, pero se sugiere algo similar a esto:

- Preguntar al usuario qué operación quiere hacer. Como pueden ser muchas opciones (todas las operaciones de todos los servicios), se sugiere preguntar en dos fases:
    - Primero, preguntar sobre qué entidad quiere probar (Customer, Product, Order, etc).
    - Una vez elegida la entidad, preguntar qué operación quiere realizarse (crear, modificar, buscar, etc)
- Para cada operación, preguntar al usuario los datos necesarios, y realizar la operación.
- Informar del resultado de la operación.
- Volver a preguntar entidad (dando una opción para salir del programa)