PRÁCTICA 3

BASES DE DATOS

Curso 2022/2023

ACCESO PROGRAMÁTICO Y SQL

1. Objetivos Generales

- Crear una base de datos en base a un diagrama E-R e insertar el contenido mediante la creación de un programa inicial de carga de datos.
- Acceder a una Base de Datos y realizar consultas SQL mediante un programa implementado en código Java.
- Manejar el conector JDBC (Java DataBase Conectivity), librería de clases para el acceso a un SGBD y para la realización de consultas SQL desde Java.

2. Objetivos específicos

El alumno será capaz de:

- Realizar conexiones a un SGBD desde un programa Java.
- Realizar consultas simples y complejas SQL manejando las clases adecuadas del conector JDBC.
- Tratamiento de los resultados de las consultas SQL dentro del código Java.
- Manejo de las excepciones en la gestión de consultas.
- Implementación de un sistema basado en el concepto de transacción usando código Java.

3. Práctica a realizar

Se desea realizar una aplicación Java que gestione la información de una base de datos que almacena información sobre una cadena de panaderías. En concreto, se almacena información sobre los locales de las panaderías, y de los empleados que han trabajado en cada local a lo largo del tiempo.

La Figura 1 muestra el diagrama E-R que modela la base de datos mencionada.

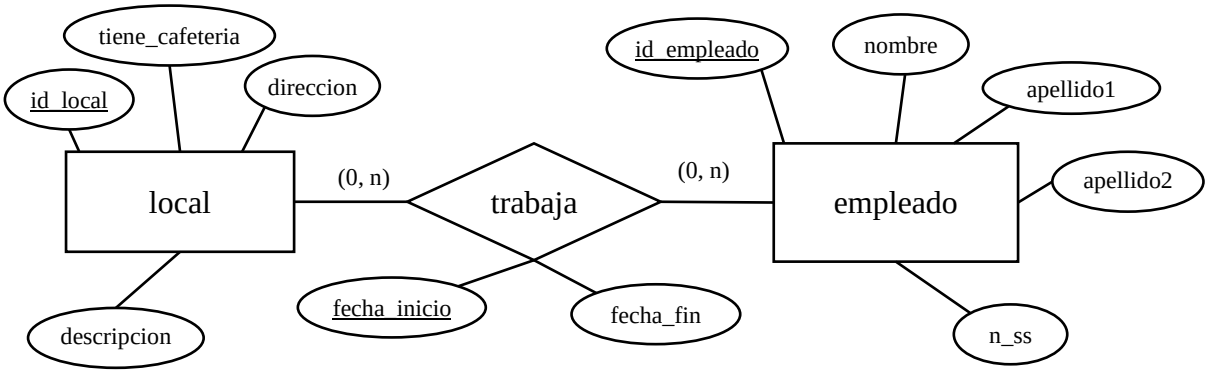


Figura 1. Diagrama entidad relación de la base de datos "panaderias".

Junto a este enunciado, encontrarás los siguientes elementos:

- Clases Java: clases del paquete Java *panaderias*. Todas las clases serán de este paquete y no se debe modificar el paquete en ninguna de ellas. Estas clases contendrán la definición de algunos atributos necesarios (podría ser necesario añadir más), constantes y las cabeceras de los métodos que se deben implementar.

Importante: queda prohibido modificar la definición de cualquier método ya existente en el fichero. Todos ellos deben hacer un correcto tratamiento de las excepciones. Si se necesita crear algún método adicional, éste debe ser privado.

- Ficheros CSV: ficheros CSV que contienen datos de ejemplo para cargar en cada una de las tablas que se generan sin que se produzcan errores. En las pruebas que se realicen en la corrección los ficheros CSV podrían ser cambiados y que éstos contengan errores de integridad de los datos (PK duplicadas, FK que hagan referencia a PK que no existan). Puede suponerse que la primera línea siempre contiene las cabeceras, que el orden de los campos siempre será el dado en estos ficheros de ejemplo y cada campo se separa del siguiente mediante un punto y coma (;).

Con todo ello, se deben implementar un sistema Object-Relation Mapping (ORM), de forma que exista una clase por tabla y que cada objeto de la clase se corresponda con una fila de la tabla. Cada vez que se modifique el valor de un atributo de un objeto, éste debe ser actualizado en su fila correspondiente y cuando se solicite el valor de un atributo de un objeto, éste debe ser actualizado desde la base de datos.

En todos los casos se prohíbe modificar las cabeceras de los métodos que se proporcionan y, especialmente, deben tratarse las excepciones en los mismos, quedando prohibido lanzarlas hacia arriba.

Para realizar esta implementación, se pide implementar los métodos de las siguientes clases (cada clase se evaluará sobre 10 puntos, los cuáles se dividirán entre los diferentes métodos según se especifica):

Clase DBConnection

Esta primera clase pretende proveer una interfaz simplificada para la interacción con la base de datos por parte de los objetos que representen cada una de las filas. Para ello se proporciona el fichero DBConnection.java, en el que se definen:

- Tres constantes que se usarán como valores "centinela" para representar valores nulos en algunas situaciones.
- Cuatro variables de clase para almacenar información relativa a la conexión.
- Un constructor para almacenar los parámetros de la conexión.
- Siete métodos para implementar la funcionalidad.

En concreto en esta sesión, se pide implementar el constructor y los métodos de acuerdo con la siguiente descripción:

1. `DBConnection`: constructor de la clase que debe inicializar los valores de las variables "user", "pass" y "url" (ésta última debe ser algo del tipo "jdbc:mysql://server:3306/database"). La implementación del constructor no tiene una puntuación como tal, pero es condición necesaria para seguir trabajando.
2. `connect()` [1 punto]: este método debe implementar la apertura de la conexión con la base de datos. El método debe devolver *true* si la conexión está disponible para su uso tras llamarlo (aunque ya lo estuviera previamente) y *false* si la conexión no está disponible. Se debe llamar a este método cuando se vayan a ejecutar operaciones contra la base de datos en otras partes del código. Nunca deben abrirse varias conexiones.

3. `close()` [1 punto]: este método debe implementar el cierre de la conexión con la base de datos. El método debe devolver *true* si se ejecuta sin errores y *false* si ocurre alguna excepción. Sólo debería ser llamado desde el programa principal para terminar la conexión.
4. `update(String sql)` [1.5 puntos]: este método debe ejecutar el comando SQL (que modifica el contenido de la base de datos y que no tiene parámetros) pasado como parámetro. Debe retornar -1 en caso de que se produzca alguna excepción y el número de filas afectadas en caso contrario.
5. `update(String sql, ArrayList<Object> a)` [2 puntos]: este método debe ejecutar el comando SQL (que modifica el contenido de la base de datos y que tiene parámetros) pasado como parámetro. Debe retornar -1 en caso de que se produzca alguna excepción y el número de filas afectadas en caso contrario.

   Los parámetros se pasan en un ArrayList de Object, debiendo llamar al método set correspondiente de la clase PreparedStatement para cada uno de ellos y hacerles un casting adecuado. Es posible obtener un String con el nombre de la clase a la que un objeto `element` pertenece con `element.getClass().getName()`. En el caso de tipos primitivos y String, el nombre de la clase retornada es "java.lang.Integer", "java.lang.Float", "java.lang.String", etc.

   Los valores nulos son siempre pasados como instancias de su tipo o clase correspondiente, haciendo uso de los valores centinela definidos (nunca como *null*) y debe hacer uso del siguiente método para establecerlos:

   ```
   PreparedStatement.setNull(int index, int sqlType);
   ```

   Como puede verse, el tipo de SQL es necesario en la llamada a setNull, por lo que este es el motivo del uso de los valores centinela. Los tipos pueden especificarse como constantes de `java.sql.Types`: `Types.VARCHAR`, `Types.INTEGER`, etc.
6. `query(String sql)` [1.5 puntos]: este método debe ejecutar la consulta SQL (que no modifica el contenido de la base de datos y que no tiene parámetros) pasada como parámetro. Debe retornar null en caso de que se produzca alguna excepción y el ResultSet correspondiente en caso contrario.
7. `query(String sql, ArrayList<Object> a)` [2 puntos]: este método debe ejecutar la consulta SQL (que no modifica el contenido de la base de datos y que tiene parámetros) pasada como parámetro. Debe retornar null en caso de que se produzca alguna excepción y el ResultSet correspondiente en caso contrario. Debe considerarse las mismas indicaciones dadas en el apartado 5.
8. `tableExists(String tableName)` [1.5 puntos]: este método debe retornar true si la tabla cuyo nombre se pasa como parámetro existe en la base de datos y falso en caso contrario. Puede ser de utilidad el comando SQL "`SHOW TABLES`", que retorna las tablas existentes en la base de datos como si fuera una consulta SELECT.

Clases Empleado, Local y Trabaja

Estas clases representan a cada una de las tablas. Se proporcionan los esqueletos de las mismas, así como el código de una clase abstracta de la que todas heredan y que no hay que modificar: DBTable. En esta clase abstracta se definen dos atributos: un objeto de la clase DBConnection

desarrollada en la primera sesión y un booleano que indica si el objeto debe sincronizarse con la base de datos o no.

En cada clase se pide implementar los siguientes métodos, haciendo siempre uso de las funcionalidades de BDConnection (no se permite crear nuevas conexiones, Statements o PreparedStatements, aunque sí puede ser necesario definir ResultSet para recoger datos):

1. `createTable()` [1 punto]: ejecuta el comando para crear la tabla correspondiente. Retorna true si el comando se ejecuta sin errores y false en caso contrario (intentar crear una tabla ya existente debe considerarse un error y, por tanto, retornar false).
2. `insertEntry()` [1 punto]: ejecuta el INSERT el elemento actual en la base de datos. Recordatorio: si alguno de los datos es nulo, debe pasarse el valor centinela correspondiente. Retorna true si el elemento se ha insertado y false en caso contrario.
3. `updateEntry()` [1 punto]: ejecuta el UPDATE del elemento actual en la base de datos. Deben actualizarse en la base de datos todos los campos que no formen parte de la clave primaria, usando los que forman parte de ésta para filtrar el elemento a actualizar. Retorna true si el elemento se ha actualizado y false en caso contrario.
4. `deleteEntry()` [1 punto]: ejecuta el DELETE del elemento actual en la base de datos. Debe filtrarse por los valores de la clave primaria para localizar el elemento a borrar. Retorna true si el elemento se ha borrado y false en caso contrario.
5. `getEntryChanges()` [1 punto]: ejecuta el SELECT para recuperar los datos almacenados en la base de datos sobre el elemento actual. Deben actualizarse en el objeto todos los atributos que no formen parte de la clave primaria, usando los que forman parte de ésta para filtrar el elemento a buscar en la base de datos. No retorna nada.
6. Constructores de la clase [1.5 puntos]: por cada clase deben implementarse dos constructores. Uno de ellos inicializa solamente los valores de la clave primaria (poniendo a nulo el resto) y el otro todos los atributos del objeto. Siempre que se cree un elemento, si la sincronización con la base de datos está activada, debe crearse la tabla correspondiente si no existe previamente e insertarse el elemento. De no poder insertar el elemento, deben ponerse a nulo también los elementos que forman parte de la clave primaria y desactivar la sincronización con la base de datos.
7. Getters de los atributos [1.25 puntos]: deben implementarse los getters de todos los atributos, actualizando los valores de los todos los atributos del objeto previamente desde la base de datos si la sincronización está activada.
8. Setters de los atributos [1.25 puntos]: deben implementarse los setters de todos los atributos que no formen parte de la clase primaria. Si la sincronización está activada, antes de realizar la modificación, deben actualizarse los valores de todos los atributos del objeto desde la base de datos y, tras la modificación de acuerdo al parámetro proporcionado, actualizar el contenido de la base de datos.
9. `destroy()` [1 punto]: borra el elemento de la base de datos si la sincronización está activada, pone a nulo (o valores centinela para tipos básicos) todos los atributos del objeto y desactiva la sincronización. No retorna nada.

Clase DataManager

Se pide implementar los métodos estáticos de la clase DataManager, que sirve para cargar los datos que existen en la base de datos en un ArrayList de elementos de la clase que mapea con la

tabla y para leer datos de un fichero CSV, cargándolos tanto en un ArrayList como en la tabla que mapea con cada clase. En concreto se pide implementar los siguientes métodos:

1. `getEmpleadosFromDB(DBConnection conn, boolean sync)` [2.5 puntos]: el método debe obtener todos los datos de la tabla "empleado", creando un objeto de la clase Empleado por cada fila y añadiendo cada uno de ellos a un ArrayList, que es lo que el método retorna. Si la tabla existe, pero está vacía, se debe retornar un ArrayList vacío, mientras que si se produce alguna excepción se debe retornar null.
   Dado que se va a intentar crear objetos de la clase Empleado con valores de clave primaria que ya existen en filas de la tabla correspondiente (justamente es lo que se pretende), es necesario crear los objetos sin sincronización con la base de datos en primera instancia y activar su sincronización tras la creación con el método `setSync(true)`.
2. `getEmpleadosFromCSV(String filename, DBConnection conn, boolean sync)` [2.5 puntos]: el método irá leyendo cada línea del CSV y creando objetos de la clase Empleado con dichos datos, con la sincronización indicada en el parámetro `sync` (si está activada, se irán insertando las filas correspondientes). En el fichero CSV siempre habrá una línea de cabecera con los nombres de las columnas, pero dichas columnas aparecen siempre según el mismo orden, tal y como se define en los ficheros de ejemplo proporcionados, por lo que no es necesario controlar este aspecto. Las columnas se separan unas de otras mediante un punto y coma (;). Sí podrían aparecer columnas con datos nulos (vacíos). El método debe retornar el ArrayList con las instancias de Empleado leídas (y, en su caso, insertadas en la base de datos). Si se produce una excepción debe retornarse null.
3. `getLocalesFromDB(DBConnection conn, boolean sync)` [2.5 puntos]: análogo al punto 1, pero sobre la clase Local.
4. `getLocalesFromCSV(String filename, DBConnection conn, boolean sync)` [2.5 puntos]: análogo al punto 2, pero sobre la clase Local.

El programa debe conectarse cuando se vaya a realizar una consulta/operación contra la BD por primera vez (no al arrancar el programa) y no debe desconectarse hasta que el programa finalice.

4. Evaluación y otras indicaciones

Es importante que se cumplan los requisitos establecidos en la práctica, incluyendo:

- El método connect() debe ser el encargado de: cargar el driver y realizar la conexión. Es obligatorio que se implemente esta funcionalidad dentro de dicho método. Parámetros:
  - Servidor: localhost:3306
  - Usuario: panaderia_user
  - Password: panaderia_pass
  - Nombre base de datos: panaderias
- Todos los identificadores deben almacenarse como tipo INT, la columna "tiene_cafeteria" de la tabla "local" debe ser también un INT que tome los valores 0

(falso) o 1 (verdadero), las fechas deben almacenarse como tipo DATE y el resto de campos (de texto) se almacenarán como VARCHAR(100).

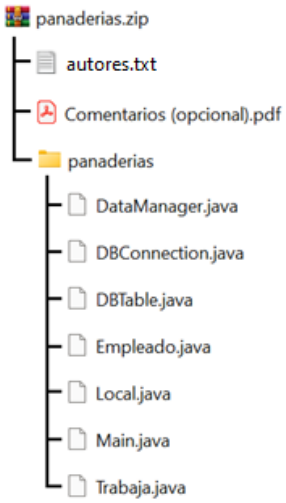
- Que sean los métodos ya dados los que, al menos, se encarguen de proporcionar el resultado final. El estudiante puede generar otros métodos adicionales si lo considera relevante, pero el método asignado a cada ejercicio debe devolver el resultado. No se pueden alterar los nombres de los ficheros, paquetes y clases que se proporcionan.
- Cada clase de evaluará de 0 a 10 puntos mediante un corrector automático. Esto quiere decir que los métodos deben funcionar perfectamente y todos los formatos deben ser acordes al presente guion. A la hora de la corrección se usarán tanto los datos especificados aquí, como otros datos para la comprobación de la corrección de las soluciones.
- Todas las clases tienen el mismo peso en la práctica.
- Esta corrección automática dará una nota preliminar de la práctica, que puede ser inferior si se dan los supuestos que se enumeran a continuación, y que se verificarán una vez sea entregada la práctica en la fecha propuesta.
    - **Limpieza y claridad del código** que, además, debe gestionar correctamente los errores, estar optimización, contener comentarios, etc.: no cumplir con estos aspectos puede suponer la deducción de hasta 1.5 puntos.
    - **Formato de entrega incorrecto:** -0.5 puntos.
    - **Conexión con un usuario que no sea el dado:** -2 puntos (si es con root), -1 punto (si es con otra combinación de usuario/contraseña inválida).
    - **Realización de más de una conexión/realización de conexión en momento inoportuno:** -1 punto.
    - **No usarse los cierres de las estructuras necesarias:** -0.5 puntos.
    - **No usar PreparedStatement cuando haya parámetro o usarlo incorrectamente (usando concatenaciones):** -3 puntos.
    - **Creación de tablas sin claves foráneas:** -1 punto.
    - **No identificar correctamente los tipos de errores:** -0.5 puntos.
    - **Lanzamiento de las excepciones hacia arriba:** para esto es necesario modificar las cabeceras de los métodos y supone la calificación automática con un 0.
    - **Usar o implementar clases adicionales:** supone una calificación con un 0.
    - Otros supuestos: Los profesores pueden aplicar otras penalizaciones en caso de encontrar errores diferentes a los aquí descritos. Quedará a discreción del profesorado la penalización a aplicar en cada caso.
- Los profesores se reservan el derecho de citar a cualquier grupo a defender la práctica si lo considerara necesario.

5. Entrega de la Práctica y resolución de dudas

El grupo de prácticas deberá **entregar mediante la plataforma Moodle en una tarea habilitada para ello un único fichero comprimido (*.zip) cuyo nombre sea (panaderias.zip) que contenga:**

1. Un fichero de texto llamado autores.txt, en el que se indiquen nombre, apellidos y número de matrícula de los integrantes de la pareja de prácticas (la no inclusión de este fichero supondrá, además de un fallo en el formato de entrega, que la práctica se considerará sólo como entregada por la persona que la ha subido).
2. El paquete *panaderias* con la implementación de las clases solicitadas.
3. Opcionalmente un documento en PDF que contenga comentarios acerca de los problemas surgidos al hacer la práctica o cualquier otro comentario que los alumnos estimen oportuno.

Es decir, la entrega debe ser exactamente de la siguiente forma (en otro caso, se aplicará la penalización descrita en la sección anterior):



La práctica debe ser entregada por **sólo uno de los miembros del grupo**.

La fecha límite para la entrega de esta práctica es el miércoles **24 de mayo de 2023 a las 23:55**.

6. Resolución de dudas

Las dudas deben plantearse en primer a instancia a través del **foro** habilitado para dicho fin en Moodle. Si fuera necesario concertar una tutoría, debe enviarse un correo electrónico al profesor de tu grupo.