Uso de bases de datos

Práctica 3: Programación mediante SQL

En la tercera parte de la práctica usaremos la base de datos (BD) con la que se ha trabajado en las prácticas anteriores. En concreto, usaremos el esquema de la BD de la práctica 2 con algunas modificaciones y accederemos desde Java usando JDBC.

Os facilitamos un código inicial que tenéis que completar para resolver los ejercicios propuestos. El código compila correctamente, pero es incompleto, puesto que no hace lo que se supone que tiene que hacer. Vuestro trabajo es completar el código en aquellas partes que os hemos dejado preparadas. Dentro del código hemos dejado comentarios con el prefijo TODO (de "to do" en inglés, "por hacer"). Es una convención. No hace falta que borréis estos comentarios porque os indican dónde tenéis que añadir vuestro código.

La complejidad del código a desarrollar no es grande y podéis usar cualquier entorno de desarrollo Java (Eclipse, NetBeans, IntelliJ IDEA, etc. o simplemente un editor de texto como por ejemplo Notepad, Ultraedit, etc.) para realizarla. No es el objetivo de esta práctica que aprendáis ningún entorno de programación concreto y, por lo tanto, os recomendamos trabajar con aquel con el que estéis familiarizados y que no os encalléis en este punto. Por otro lado, tal y cómo se indica en el manual de instalación, se recomienda utilizar la versión 17 de Java JDK.

En cuanto a la organización del código de la práctica, veréis que todo el código Java se encuentra en el directorio */src/main/java*, mientras que los ficheros adicionales que se utilizan se encuentran en el directorio */src/main/resources*.

Dentro del código Java suministrado está el paquete *edu.uoc.practica.bd.util* que contiene varias clases de utilidad que se utilizan para tratar la presentación de listados, la lectura de ficheros o el tratamiento de los *arrays*. De estas clases, sólo hará falta que modifiquéis la clase *DBAccessor*, que es la encargada de obtener las conexiones a la base de datos. Los parámetros que se usan para establecer las conexiones (servidor, puerto, usuario, etc.) se encuentran en el fichero */src/main/resources/db.properties*.

En todos los casos, tenéis que pensar cuál es el mejor tipo de elemento *JDBC* que podéis usar (*Statement, PreparedStatement, CallableStatement*, etc.) así como de qué manera tenéis que tratar las transacciones (*commit* y *rollback*) y las excepciones. Siempre tenéis que garantizar que se cierran todos los recursos.

Para la correcta ejecución de la tercera parte de la práctica, hay que volver a crear la base de datos ejecutando los siguientes *scripts*, que os facilitamos junto al enunciado, en el directorio */sql*:

1. *create_db.sql*
2. *inserts_db.sql*

Es importante que, antes de cada ejercicio, borréis el contenido de la base de datos y volváis a insertar los registros del fichero *inserts_db.sql*, para evitar interferencias entre los ejercicios.

En el siguiente enlace encontraréis información sobre el tipo numérico *SERIAL*, usado en la tabla *VISIT*.

https://www.postgresql.org/docs/12/datatype-numeric.html#DATATYPE-SERIAL

Nota importante: Dado que las diferentes versiones de PostgreSQL aceptan diferentes variantes de sintaxis de sus sentencias, sólo se considerarán válidas aquellas respuestas que obedezcan estrictamente las sentencias SQL explicadas a los módulos teóricos de la asignatura.

Pregunta 1 (50 % puntuación)

Enunciado

Se pide hacer un programa que liste la información de la tabla *REPORT_DOG* construida y rellenada en el primer ejercicio de la entrega de la segunda parte de la práctica. En el directorio *sql/PR2* del *zip* que adjuntamos con el enunciado, encontraréis el código SQL necesario para construir y rellenar la tabla *REPORT_DOG*.

En concreto, se quiere un listado de esta tabla ordenado de forma descendente por el número de visitas (*num_visits*) y, en caso de empate, alfabéticamente por el nombre del perro (*name_dog*)..

El encabezamiento del listado será:

```
Id dog  Name dog Num visits Num dif. vac. Date last vac. Num drugs Num tests
======= ======== ========== ============= ============== ========= =========
```

Si la tabla está vacía se mostrará el mensaje *“List without data”.*

Se tendrá que realizar la gestión de errores necesaria y en el supuesto de que se produzcan errores, se tendrá que devolver un error genérico *“ERROR: List not available”.*

Con la ejecución de los *scripts create_db.sql* e *inserts_db.sql*, que facilitamos conjuntamente con el enunciado, tendremos la base de datos preparada.

Como parte de este ejercicio también se evaluará la implementación que realicéis de la clase *DBAccessor*.

Criterios de evaluación

- Os recomendamos compilar y ejecutar el fichero que entreguéis, por si habéis hecho alguna modificación de última hora (reordenación del código, algún comentario adicional, etc.).
- No se puntuará la entrega de ningún fichero si genera errores de compilación.
- La calificación de la programación de *DBAccessor* será un 30% del valor de esta pregunta. Adicionalmente, se valorará:
    - Que funcione la conexión.
    - Que se seleccione el *path* correctamente.
    - Que se utilicen correctamente los parámetros del fichero *db.properties*.
    - Que no provoquen errores de ejecución cuando se invoque desde el código solución de los ejercicios.
- Para la calificación del resto de la pregunta (70%), se valorará:
    - La corrección en el uso de las clases de JDBC.
    - Que el listado sea correcto y que se controlen las situaciones en que no exista la tabla y en que la tabla esté vacía.
    - Que se realice el cierre de todos los recursos.
    - Se tendrá en cuenta en la calificación que el código entregado presente comentarios que permitan comprender su funcionamiento.
    - La calificación se verá penalizada si os dejáis “espías” a lo largo del código.
    - Para obtener la calificación máxima se tendrá en cuenta tanto la claridad del código como la eficiencia del mismo.
    - Igualmente, para obtener la máxima calificación, se tendrán que dar los ejemplos que muestren el funcionamiento del programa en todas las situaciones pedidas.

Solución

El método `getConnection()` de la clase *DBAccessor* crea la conexión a la BD con los valores que se han inicializado con la ejecución del método `init()`. En el supuesto de que se produzca alguna excepción en la ejecución del método `getConnection()`, se mostrará un mensaje por pantalla y se devolverá el valor NULL.

Dentro del método `getConnection()`, y justo después de haber obtenido la conexión, se hace una configuración del `search_path`, con el `"SET search_path to " + schema` usando `executeUpdate`.

Habrá que utilizar simplemente la clase `Statement` para invocar la ejecución de la sentencia SQL que necesitamos y almacenar los datos en un `ResultSet` el cual habrá que ir recorriendo fila a fila para proporcionar la salida.

La sentencia SQL a ejecutar desde Java es la siguiente:

```
SELECT * FROM report_dog ORDER BY num_visits DESC, name_dog;
```

Las pruebas de ejecución son:

1. Si la tabla no existe:

```
ERROR: List not available

ERROR: relation "report_dog" does not exist

Position: 15
```

2. Si la tabla no contiene datos:

```
List without data
```

3. Si la tabla tiene datos:

```
Id dog  Name dog Num visits Num dif. vac. Date last vac. Num drugs Num tests
======= ======== ========== ============= ============== ========= =========
      6 Max                6             1 2021-01-23             2         7
      2 Rocky              5             1 2020-03-14             1         3
     34 Rocky              3             1 2018-01-22             2         2
     36 Max                2             0 -                      0         1
      8 Rocky              2             0 -                      1         3
```

Para rellenar la tabla con datos, ejecutamos:

```
SELECT * FROM update_report_dog('Rocky');
SELECT * FROM update_report_dog('Max');
```

Pregunta 2 (50 % puntuación)

Enunciado

Leer de un fichero y hacer *INSERT* o *UPDATE.*

Nos hacen llegar un fichero llamado `exercise2.data` que se encuentra en el directorio `src/main/resources` para ir actualizando información de nuestra base de datos. Este fichero contiene nuevos datos que se quieren insertar en la base de datos.

El fichero se carga en el sistema tal y como se detalla a continuación.

La primera línea del fichero es un comentario (que el código que os damos ya se encarga de ignorar) y que contiene la descripción de cada uno de los elementos del fichero, que van separados por comas.

Las líneas posteriores de este fichero son los datos con los cuales se actualizarán las tablas *DOG* y *VISIT*. Concretamente, tenéis que hacer un programa que, para cada una de estas líneas, haga lo siguiente:

1. Actualice los datos del perro si *id_dog* existe en la base de datos. En caso contrario, insertará un nuevo perro con los datos del fichero.
2. Registre automáticamente una visita de tipo '*follow-up*' a cargo del veterinario con id '4465' para hacerle una revisión el día que el perro celebre su primer aniversario, y que por lo tanto lleve el comentario '*First year follow-up*'. Además, se tiene que mostrar por pantalla el *id_visit* que el SGBD asigna a las visitas añadidas.

Tenemos que controlar los posibles errores de ejecución, incluidas las excepciones que puedan producirse por restricciones violadas, como por ejemplo:

- El *id_owner* no existe.
- Valores incorrectos para una columna.

En estos casos, hay que abortar la carga del fichero, y descartar los cambios hechos hasta el momento.

Criterios de evaluación

- Os recomendamos compilar y ejecutar el fichero que entreguéis, por si habéis hecho alguna modificación de última hora (reordenación del código, algún comentario adicional, etc.).
- No se puntuará la entrega de ningún fichero si genera errores de compilación.

- Se valorará la corrección en el uso de las clases de JDBC.
- Que las operaciones *INSERT* y *UPDATE* se ejecuten haciendo lo que pide el ejercicio.
- Que se realice correctamente el *rollback*.
- Que se realice el cierre de todos los recursos.
- Se tendrá en cuenta en la calificación que el código entregado presente comentarios que permitan comprender su funcionamiento.
- La calificación también se verá penalizada si os dejáis “espías” a lo largo del código.
- Para obtener la calificación máxima se tendrá en cuenta tanto la claridad del código como la eficiencia del mismo.
- Igualmente, para obtener la máxima calificación, se tendrán que dar los ejemplos que muestren el funcionamiento del programa en todas las situaciones pedidas en el documento de explicaciones que tenéis que entregar.

Solución

En el método `run()` de la clase `Exercise2UpdateOrInsertDataFromFile` se utilizan tres instancias de `PreparedStatement`:

- *UPDATE* de *DOG*
- *INSERT* en *DOG*
- *INSERT* en *VISIT*

En primer lugar, se intenta actualizar el perro. Si la función `executeUpdate()` devuelve 0, significa que el perro no existe y que no se ha actualizado ninguna entrada/fila, y por tanto, se procede a hacer el *INSERT* en la tabla *DOG*.

Posteriormente, se hace un *INSERT* para registrar la visita.

La validación de la transacción (el *commit*) se realiza fuera del bucle, porque de este modo la demarcación de la transacción (lo que desharemos −*rollback*− en el supuesto de que no vaya bien la ejecución) contiene todas las actualizaciones/inserciones realizadas.

Todas las excepciones JDBC son capturadas y, si se producen, se mostrará un mensaje por pantalla. El cierre de los recursos (`PreparedStatement` y `Connection`) se hace dentro del bloque `finally` de la excepción, y las operaciones de cierre también tienen su tratamiento de excepción.

Como vemos en el fichero `exercise2.data`, se intenta modificar/insertar la siguiente información:

```
# id_dog, name_dog, breed, birth, death, sex, color, fur, id_owner,
num_vaccines

38, Yago, Bulldog, 2021-07-11, , M, dark chocolate, Short, 5, 2

46, Molly, Beagle, 2021-12-08, , F, White, Medium, 17, 0
```

```
47, Emacs, Chihuahua, 2022-01-15, , M, Gray, Long, 25, 1
```

Hacemos las consultas para ver el estado de la base de datos:

```
SELECT *
FROM dog AS d
NATURAL LEFT JOIN visit AS v
WHERE d.id_dog IN (38, 46, 47)
ORDER BY d.id_dog, v.id_visit;
```

En el estado inicial, antes de ejecutar el código del ejercicio 2, la consulta devuelve:

id_dog	name_dog	breed	birth	death	sex	color	fur	id_owner	num_vaccines	id_visit	date	reason	id_veterinary	comments
38	Yago	French Bulldog	2021-07-11		M	brown	Short	5	0					

Después de la ejecución del programa, la misma consulta devolverá:

id_dog	name_dog	breed	birth	death	sex	color	fur	id_owner	num_vaccines	id_visit	date	reason	id_veterinary	comments
38	Yago	Bulldog	2021-07-11		M	dark chocolate	Short	5	2	69	2022-07-11	follow-up	4465	First year follow-up
46	Molly	Beagle	2021-12-08		F	White	Medium	17	0	70	2022-12-08	follow-up	4465	First year follow-up
47	Emacs	Chihuahua	2022-01-15		M	Gray	Long	25	1	71	2023-01-15	follow-up	4465	First year follow-up

Como vemos, los datos se han actualizado e insertado correctamente. Además, se muestra el id de las visitas insertadas.

```
Dog updated correctly.

Visit with id_visit = 69 added to table VISIT

Dog does not exist. Inserting ...

Visit with id_visit = 70 added to table VISIT

Dog does not exist. Inserting ...

Visit with id_visit = 71 added to table VISIT
```

Si ejecutamos el programa con el fichero `exercise2.data` añadiendo la siguiente línea:

```
48, Rocky, Beagle, 2020-08-15, , M, white, Short, 30, 4
```

Donde el `id_owner` es de un propietario que no existe (30), obtenemos el siguiente error:

```
ERROR: Executing SQL on DOG or VISIT

ERROR: insert or update on table "dog" violates foreign key constraint
"fk_owner_dog"

  Detail: Key (id_owner)=(30) is not present in table "owner".

Rolling back and reverting changes
```