# Uso de bases de datos

## Práctica 3: Programación mediante SQL

En esta tercera práctica usaremos la base de datos (BD) con la que se ha trabajado en prácticas anteriores (en concreto, usaremos el esquema de la BD de la práctica 2) 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, ya 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, "para hacer"). Es una convención. No hace falta que borréis estos comentarios puesto que os indican donde tenéis que añadir vuestro código.

La complejidad del código que se va 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. El objetivo de esta práctica no consiste en aprender ningún entorno de programación en concreto, por lo que os recomendamos trabajar con aquel con el que estéis familiarizados para que no os atasquéis en este punto. Por otro lado, podéis usar cualquier versión de Java superior a 1.5.

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 hay 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 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 usan para establecer las conexiones (servidor, puerto, usuario, etc.) se encuentran en el fichero */src/main/resources/db.properties*.

En todos los casos debéis pensar cuál es el mejor tipo de sentencia *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 debéis 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 scripts siguientes, que os facilitamos junto al enunciado:

1. *create_db.sql*
2. *insert_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.**

**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_BAND* construida y rellenada en el primer ejercicio de la entrega de la segunda parte de la práctica. En la carpeta sql/PR2 del zip que adjuntamos junto al enunciado, encontraréis el código sql necesario para construir y rellenar la tabla *REPORT_BAND*.

En concreto se quiere un listado de esta tabla ordenado de forma ascendente por identificador de la banda.

La cabecera será:

```
Id band  Num instruments  Num members alive  Big albums  Num short songs
=======  ===============  =================  ==========  ===============
```

Si la tabla está vacía se dará un mensaje *“List without data”.*

Se tendrá que realizar la gestión de errores necesaria y en el supuesto de que se produzcan, 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 junto al enunciado, tendremos la base de datos preparada.

Como parte de este ejercicio también se evaluará la implementación que hagáis 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 del *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 provoque errores de ejecución cuando se invoque desde el código 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 donde no exista la tabla o la tabla esté vacía.
    - Que se haga 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 sentencia *Statement* para ejecutar la sentencia SQL que necesitamos y almacenar los datos en un *ResultSet*, el cual se deberá ir recorriendo fila a fila y dando la salida.

La sentencia SQL a ejecutar es:

```
SELECT * FROM report_band ORDER BY id_band;
```

Las pruebas de ejecución son:

1. Si la tabla no existe:

```
ERROR: List not available

ERROR: relation "report_band" does not exist

Position: 15
```

2. Si la tabla no tiene datos:

```
List without data
```

3. Si la tabla tiene datos:

```
Id band  Num instruments  Num members alive  Big albums       Num short songs
=======  ===============  =================  ============== ===============
      3                1                  2  Fire and Water                 0
```

Para rellenar la tabla con datos, ejecutamos:

```
SELECT * FROM update_report_band('Free');
```

# Pregunta 2 (50% puntuación)

## Enunciado

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

Nos hacen llegar un fichero llamado `exercise2.data`, para ir actualizando información de nuestra base de datos. Este fichero contiene datos nuevos que se quieren insertar en la base de datos.

El fichero se carga en el sistema tal 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 de los grupos con los cuales se actualizarán las tablas BAND y MEMBER. Concretamente, hace falta que hagáis un programa que, por cada una de estas líneas, haga lo siguiente:

1. Actualice los datos del grupo si el *id_band* existe en la base de datos. En caso contrario, insertará un nuevo grupo con los datos del fichero.

2. Registre la participación del músico en el grupo en el supuesto de que el músico no esté ya registrado como miembro del grupo con el instrumento que aparece en el fichero de entrada.

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

- El *id_musician* no existe.

- Valores incorrectos para una columna.

En estos casos, hay que abortar la carga del fichero, y deshacer los cambios realizados 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 del JDBC.
- Que las operaciones INSERT y UPDATE se ejecuten haciendo lo que pide el ejercicio.
- Que se haga correctamente el *Rollback*.
- Que se haga 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 librar.

### Solución

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

- *UPDATE* de *BAND*
- *INSERT* en *BAND*
- *SELECT* en *MEMBER*
- *INSERT* en *MEMBER*

En primer lugar, se intenta actualizar el grupo. Si la función `executeUpdate()` devuelve 0, quiere decir que no existe y que no se ha actualizado ninguna entrada, por lo que se procede a hacer el *INSERT* en la tabla *BAND*.

Posteriormente, se hace una *SELECT* para ver si el miembro ya está registrado en la banda, en caso de que no sea así, se procede a realizar un insert.

La validación de la transacción (*commit*) se hace fuera del bucle, porque de este modo la demarcación de la transacción (que desharemos 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_band,name,year_formed,year_dissolution,style,origin,id_musician,instrument
1,Metallica,1982,,Heavy,US,1,Guitar
5,Ramones,1975,1996,Punk,US,24,Guitar
11,Mad Season,1994,1995,Rock,US,27,Vocals
```

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

```
SELECT *
FROM BAND NATURAL JOIN MEMBER
WHERE id_band = 1 OR id_band = 5 OR id_band = 11;
```

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

| | id_band smallint | name character varying (255) | year_formed smallint | year_dissolution smallint | style character varying (255) | origin character varying (255) | id_musician smallint | instrument character varying (255) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Metallica | 1982 | [null] | Heavy | US | 1 | Vocals |
| 2 | 1 | Metallica | 1982 | [null] | Heavy | US | 3 | Guitar |
| 3 | 1 | Metallica | 1982 | [null] | Heavy | US | 6 | Drums |
| 4 | 1 | Metallica | 1982 | [null] | Heavy | US | 7 | Bass |
| 5 | 5 | Ramones | 1974 | 1995 | Punk | US | 23 | Vocals |
| 6 | 5 | Ramones | 1974 | 1995 | Punk | US | 24 | Guitar |
| 7 | 5 | Ramones | 1974 | 1995 | Punk | US | 25 | Bass |
| 8 | 5 | Ramones | 1974 | 1995 | Punk | US | 26 | Drums |

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

| | id_band smallint | name character varying (255) | year_formed smallint | year_dissolution smallint | style character varying (255) | origin character varying (255) | id_musician smallint | instrument character varying (255) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Metallica | 1982 | [null] | Heavy | US | 1 | Vocals |
| 2 | 1 | Metallica | 1982 | [null] | Heavy | US | 3 | Guitar |
| 3 | 1 | Metallica | 1982 | [null] | Heavy | US | 6 | Drums |
| 4 | 1 | Metallica | 1982 | [null] | Heavy | US | 7 | Bass |
| 5 | 5 | Ramones | 1975 | 1996 | Punk | US | 23 | Vocals |
| 6 | 5 | Ramones | 1975 | 1996 | Punk | US | 24 | Guitar |
| 7 | 5 | Ramones | 1975 | 1996 | Punk | US | 25 | Bass |
| 8 | 5 | Ramones | 1975 | 1996 | Punk | US | 26 | Drums |
| 9 | 1 | Metallica | 1982 | [null] | Heavy | US | 1 | Guitar |
| 10 | 11 | Mad Season | 1994 | 1995 | Rock | US | 27 | Vocals |

Como vemos, los datos se han actualizado e insertado correctamente.

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

```
11,Mad Season,1994,1995,Rock,US,28,Vocals
```

Donde el `id_musician` es de un músico que no existe, obtenemos el siguiente error:

```
ERROR: Executing SQL on BAND or MEMBER

ERROR: insert or update on table "member" violates foreign key constraint
"fk_musician_member"

  Detail: Key (id_musician)=(28) is not present in table "musician".
```