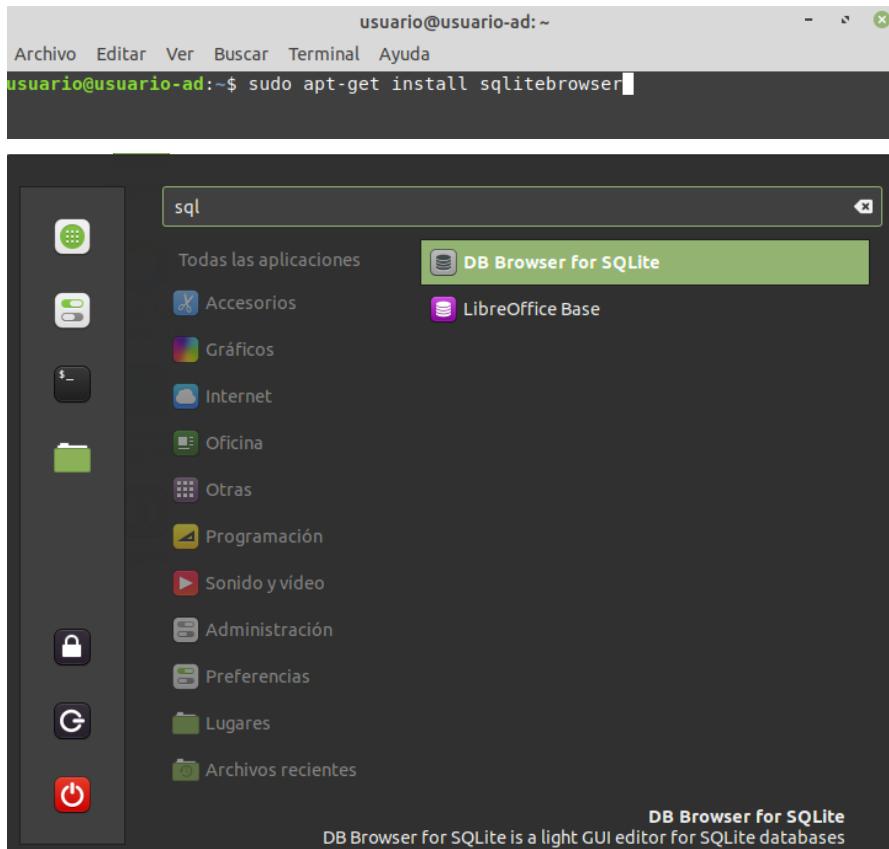
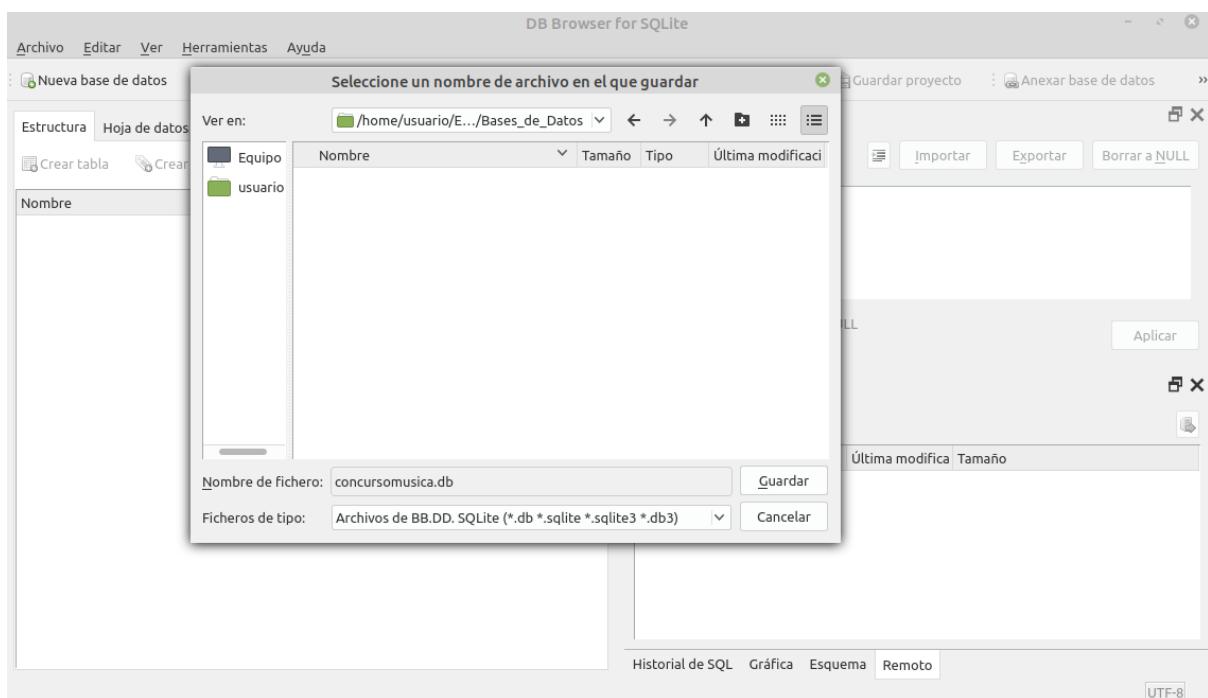


BD SQL Hoja 1

1.- Instala y ejecuta **DB Browser for SQLite** en la máquina virtual.

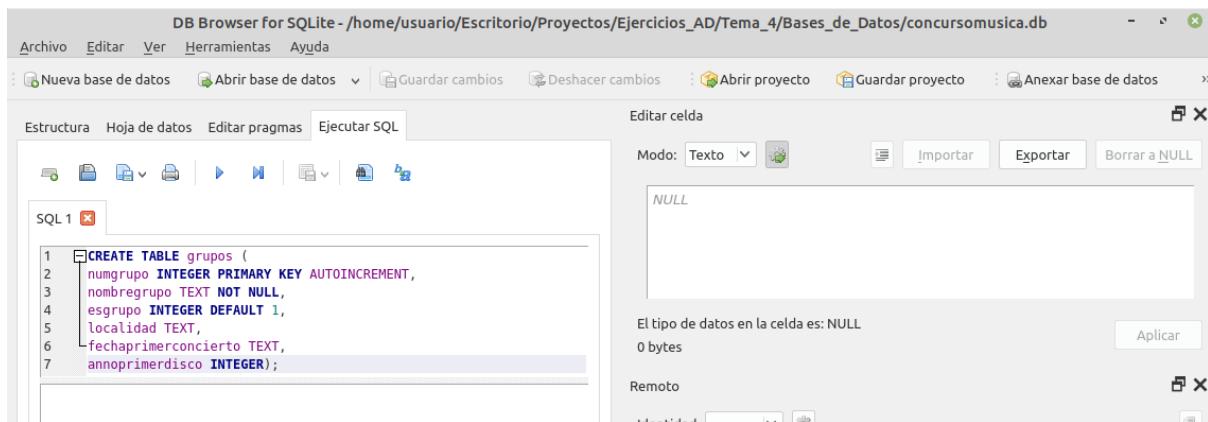


2.- Crea una base de datos llamada **concursemusica.db**



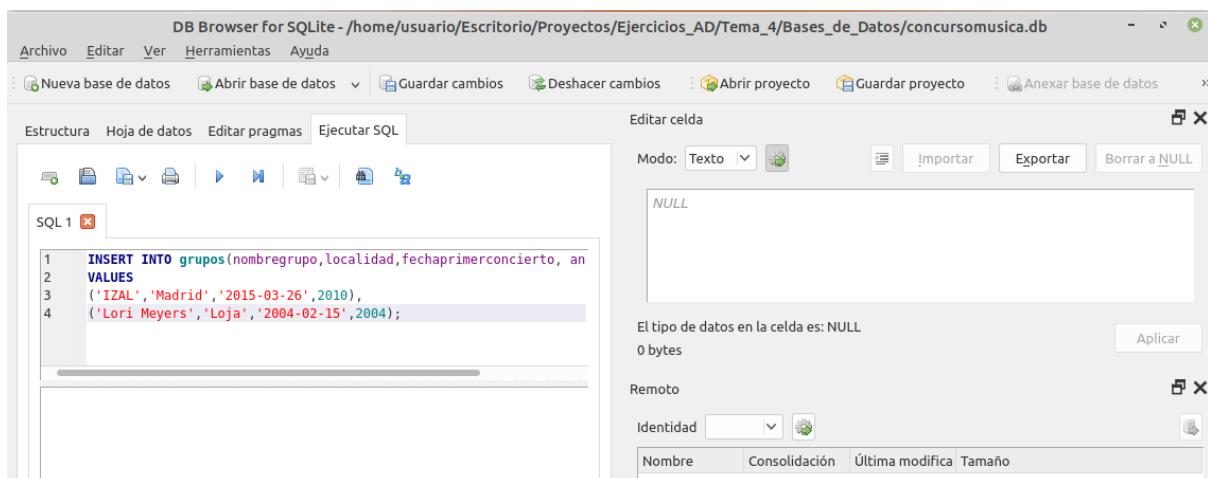
3.- Añade una tabla para almacenar información de grupos de música. Sólo hay tipos de dato **INTEGER, REAL, TEXT y BLOB**

```
CREATE TABLE grupos (
    numgrupo INTEGER PRIMARY KEY AUTOINCREMENT,
    nombregrupo TEXT NOT NULL,
    esgrupo INTEGER DEFAULT 1,
    localidad TEXT,
    fechaprimerconcierto TEXT,
    annoprimerdisco INTEGER);
```



4.- Añadir dos filas a la tabla de grupos. No dar valores a las columnas autoincrementada y con valor por defecto.

```
INSERT INTO grupos(nombregrupo,localidad,fechaprimerconcierto, annoprimerdisco)
VALUES
('IZAL','Madrid','2015-03-26',2010),
('Lori Meyers','Loja','2004-02-15',2004);
```



5.- Cierra ahora DB Browser for SQLite. Descarga SQLite y ejecuta por consola sqlite3 y abre la base de datos concursomusica.db

```
usuario@usuario-ad:~$ sudo apt-get install sqlite
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no
son necesarios.
  bridge-utils ubuntu-fan
Utilice «sudo apt autoremove» para eliminarlos.
Se instalarán los siguientes paquetes adicionales:
  libsqlite0 sqlite3
Paquetes sugeridos:
  sqlite-doc sqlite3-doc
Se instalarán los siguientes paquetes NUEVOS:
  libsqlite0 sqlite sqlite3
0 actualizados, 3 nuevos se instalarán, 0 para eliminar y 219 no actualizados.
Se necesita descargar 1.037 kB de archivos.
Se utilizarán 3.259 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 libsqlite0 amd64 2.8
.17-15fakesync1build1 [160 kB]
Des:2 http://archive.ubuntu.com/ubuntu focal/universe amd64 sqlite amd64 2.8.17-
usuario@usuario-ad:~/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Bases_de_Datos$ sqlite3 concursomusica.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> 
```

6.- Realizar las siguientes consultas

- Obtener el número, nombre y localidad de todos los grupos.
- Obtener todos los datos de los grupos de Madrid.

```
sqlite> select * from grupos;
1|IZAL|1|Madrid|2015-03-26|2010
2|Lori Meyers|1|Loja|2004-02-15|2004
sqlite> 
sqlite> select * from grupos where localidad = "Madrid";
1|IZAL|1|Madrid|2015-03-26|2010
sqlite> 
```

7.- Prueba también los siguientes comandos de SQLite:

- Obtener las tablas de la base de datos abierta
 .tables
- Obtener la estructura de la tabla grupos
 .schema grupos
- Salir de sqlite
 .quit

```
sqlite> .tables
grupos
sqlite> .schema grupos
CREATE TABLE grupos (
    numgrupo INTEGER PRIMARY KEY AUTOINCREMENT,
    nombregrupo TEXT NOT NULL,
    esgrupo INTEGER DEFAULT 1,
    localidad TEXT,
    fechaprimeroconcierto TEXT,
    annoprimerocono DISCO INTEGER);
sqlite> .quit
usuario@usuario-ad:~/Escritorio/Proyectos/Eje
```

Inserta grupos nuevos

```
sqlite> INSERT INTO grupos(nombregrupo,localidad,fechaprimeroconcierto, annoprimerocono)
...> VALUES
...> ('La Fuga','Reinosa','1997-04-08',1998),
...> ('Avicii','Suecia','2007-07-26',2013),
...> ('Xenon','Alicante','2007-02-23',2008),
...> ('Iron Maiden','UK','1976-06-1',1980);
sqlite>
```

8.- A través de la consola crea una tabla canciones que contendrá para cada canción:

- Un número de canción autoincrementado.
- El título
- La duración en segundos.
- El número de grupo al que pertenece (clave ajena)

```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> CREATE TABLE canciones(
...>     numcancion INTEGER PRIMARY KEY AUTOINCREMENT,
...>     titulo TEXT NOT NULL,
...>     duracion INTEGER,
...>     localidad TEXT,
...>     numgrupo INTEGER,
...>     FOREIGN KEY(numgrupo) REFERENCES grupos(numgrupo));
sqlite> .tables
canciones  grupos
sqlite> 
```

9.- Añade a la tabla dos canciones de cada uno de los grupos insertados anteriormente.

```
sqlite> INSERT INTO canciones(titulo,duracion,numgrupo) VALUES
...> ("Hasta Nunca",310,3),
...> ("Trampas al sol", 246,3),
...> ("The Nights",176,4),
...> ("Hey Brother",255,4),
...> ("Mas de 10 anios",184,5),
...> ("No Tengo Rival",233,5),
...> ("Wasted Years",309,6),
...> ("Aces High",270,6);
sqlite> □
```

Estructura Hoja de datos Editar pragmas Ejecutar SQL

Tabla: **canciones** Nuevo registro Borrar registro

	numcancion	titulo	duracion	numgrupo	
	Filtro	Filtro	Filtro	Filtro	
1	1	Hasta Nunca	310	3	
2	2	Trampas al sol	246	3	
3	3	The Nights	176	4	
4	4	Hey Brother	255	4	
5	5	Mas de 10 anios	184	5	
6	6	No Tengo Rival	233	5	
7	7	Wasted Years	309	6	
8	8	Aces High	270	6	

10.- Obtén los títulos de las canciones del grupo de nombre 'Izal' (en comparaciones de texto SQLite diferencia entre mayúsculas y minúsculas. Si quiero que no diferencie debemos añadir COLLATE NOCASE al final de la consulta SELECT).

BD SQL Hoja 2

1- Partiendo del ejemplo anterior, desarrolla y ejecuta los siguientes métodos:

```
public class NewMain
{
    public static void main(String[] args)
    {
        Connection conexion;
        try
        {
            Class.forName("org.sqlite.JDBC");
            conexion = DriverManager.getConnection("jdbc:sqlite:concursumusica.db");
            System.out.println("Conexion OK con concursomusica.db");
            //-----

            consultarGrupos(conexion);
            System.out.println("-----");
            insertarVariosGrupos2(conexion);
            consultarCancionesGrupo(conexion);
            System.out.println("-----");
            consultarGrupos(conexion);

            //-----
        } catch (ClassNotFoundException ex)
        {
            System.out.println("Error al cargar el driver " + ex.toString());
        } catch (SQLException ex)
        {
            System.out.println("Error al crear la conexion " + ex.toString());
        }
    }

    private static void consultarGrupos(Connection conexion)
    {
        try
        {
            //Creamos un objeto Statement para enviar instrucciones SQL sin parámetros
            Statement st = conexion.createStatement();
            //Crear instrucción SQL
            String txtSQL = "SELECT nombregrupo, localidad FROM grupos";
            //Devuelve un conjunto de resultados
            ResultSet result = st.executeQuery(txtSQL);
            //Procesar ResultSet...
            while (result.next())
            {
                // Saca el valor de la columna nombregrupo y de la segunda columna
                String nombre = result.getString("nombregrupo");
                String localidad = result.getString(2);
                System.out.println("NOMBRE: " + nombre + ", LOCALIDAD: " + localidad);
            }
        } catch (SQLException e)
        {
            System.err.println("Error en el método consultarGrupos");
        }
    }
}
```

```
private static void insertarGrupo(Connection conexion)
{
    //Recoger datos (lo ideal sería haber recibido un objeto Grupo por parámetro)
    Scanner teclado = new Scanner(System.in);
    //Crear texto de consulta con parámetros sustituibles
    System.out.println("Nombre");
    String nombre = teclado.nextLine();
    System.out.println("Localidad");
    String localidad = teclado.nextLine();
    System.out.println("¿Es grupo? 0/1");
    int esGrupo = teclado.nextInt();
    teclado.nextLine();
    System.out.println("Fecha");
    String fecha = teclado.nextLine();
    System.out.println("Año");
    int año = teclado.nextInt();

    //Crear texto de consulta con parámetros sustituibles
    String sql = "INSERT INTO grupos (nombregrupo,localidad,"
        + "esgrupo,fechaPrimerConcierto,annoPrimerDisco) "
        + "VALUES (?,?,?,?,?);";

    //Construir un PreparedStatement para sustituir valores en consulta parametrizada
    try ( PreparedStatement consultaPreparada = conexion.prepareStatement(sql))
    {
        // Sustituye la ? primera por el contenido de nombre
        consultaPreparada.setString(1, nombre);
        // Sustituye la ? segunda por el contenido de localidad
        consultaPreparada.setString(2, localidad);
        consultaPreparada.setInt(3, esGrupo);
        consultaPreparada.setString(4, fecha);
        consultaPreparada.setInt(5, año);

        int numeroGruposInsertados = consultaPreparada.executeUpdate();
        System.out.println("Se han insertado " + numeroGruposInsertados + " grupos");
    } catch (SQLException ex)
    {
        System.err.println("Error al insertar un grupo");
    }
}
```

- Un método que permite insertar varios grupos cuyos datos se recogen por teclado. Donde tenías la llamada para ejecutar el método para insertar un solo grupo, sustituye para llamar a este nuevo método.

```

private static void insertarVariosGrupos( Connection conexion)
{
    boolean continuar = true;
    do
    {
        //Recoger datos (lo ideal sería haber recibido un objeto Grupo por parámetro)

        //Crear texto de consulta con parámetros sustituibles
        String nombre = Teclado.introString("Nombre");
        String localidad = Teclado.introString("Localidad");
        int esGrupo = Teclado.introInt("¿Es grupo? 0/1");
        String fecha = Teclado.introString("Fecha");
        int año = Teclado.introInt("Año");

        //Crear texto de consulta con parámetros sustituibles
        String sql = "INSERT INTO grupos (nombregrupo,localidad,"
            + "esgrupo,fechaPrimerConcierto,annoPrimerDisco) "
            + "VALUES (?,?,?,?,?,?);";

        //Construir un PreparedStatement para sustituir valores en consulta parametrizada
        try ( PreparedStatement consultaPreparada = conexion.prepareStatement(sql))
        {
            // Sustituye la ? primera por el contenido de nombre
            consultaPreparada.setString(1, nombre);
            // Sustituye la ? segunda por el contenido de localidad
            consultaPreparada.setString(2, localidad);
            consultaPreparada.setInt(3, esGrupo);
            consultaPreparada.setString(4, fecha);
            consultaPreparada.setInt(5, año);

            int numeroGruposInsertados = consultaPreparada.executeUpdate();
            System.out.println("Se han insertado " + numeroGruposInsertados + " grupos");
        } catch (SQLException ex)
        {
            System.err.println("Error al insertar un grupo");
        }

        continuar = Teclado.introBoolean("Insertar otro grupo?");
    } while (continuar == true);
}

```

```
private static void insertarVariosGrupos2(Connection conexion)
{
    boolean continuar = true;
    int numeroGruposInsertados = 0;
    //Crear texto de consulta con parámetros sustituibles
    String sql = "INSERT INTO grupos (nombregrupo,localidad,"
        + "esgrupo,fechaPrimerConcierto,annoPrimerDisco) "
        + "VALUES (?,?,?,?,?,?);";

    //Construir un PreparedStatement para sustituir valores en consulta parametrizada
    try ( PreparedStatement consultaPreparada = conexion.prepareStatement(sql))
    {
        do
        {
            //Recoger datos (lo ideal sería haber recibido un objeto Grupo por parámetro)
            //Crear texto de consulta con parámetros sustituibles
            String nombre = Teclado.introString("Nombre");
            String localidad = Teclado.introString("Localidad");
            int esGrupo = Teclado.introInt("¿Es grupo? 0/1");
            String fecha = Teclado.introString("Fecha");
            int año = Teclado.introInt("Año");

            // Sustituye la ? primera por el contenido de nombre
            consultaPreparada.setString(1, nombre);
            // Sustituye la ? segunda por el contenido de localidad
            consultaPreparada.setString(2, localidad);
            consultaPreparada.setInt(3, esGrupo);
            consultaPreparada.setString(4, fecha);
            consultaPreparada.setInt(5, año);
            consultaPreparada.executeUpdate();
            numeroGruposInsertados++;

            continuar = Teclado.introBoolean("Insertar otro grupo?");
        } while (continuar == true);

        System.out.println("Se han insertado " + numeroGruposInsertados + " grupos");
    } catch (SQLException ex)
    {
        System.err.println("Error al insertar un grupo");
    }
}
```

- Un método que permite obtener los títulos y la duración en formato mm:ss de las canciones del grupo cuyo nombre se dé por teclado.

```

private static void consultarCancionesGrupo(Connection conexion)
{
    String grupo = Teclado.introString("Nombre del grupo");
    try
    {
        //Creamos un objeto Statement para enviar instrucciones SQL sin parámetros
        Statement st = conexion.createStatement();
        //Crear instrucción SQL
        String txtSQL = "SELECT titulo, duracion "
                        + "FROM canciones INNER JOIN grupos "
                        + "ON canciones.numgrupo = grupos.numgrupo "
                        + "WHERE grupos.nombregrupo = '" + grupo + "' ";
        //Devuelve un conjunto de resultados
        ResultSet result = st.executeQuery(txtSQL);
        //Procesar ResultSet...
        while (result.next())
        {
            // Saca el valor de la columna nombregrupo y de la segunda columna
            String titulo = result.getString("titulo");
            int duracion = Integer.parseInt(result.getString("duracion"));
            System.out.println("TITULO: "+titulo+", DURACION: "+calcularTiempo(duracion));
        }
    } catch (SQLException e)
    {
        System.err.println("Error en el método consultarGrupos");
    }
}

private static String calcularTiempo(int tsegundos)
{
    int minutos = tsegundos / 60;
    int segundos = tsegundos - (minutos * 60);
    return minutos + " min " + segundos + " seg";
}
}

```

DB SQL Hoja 3

1.- En primer lugar vamos a crear un contenedor con PostgreSQL. Para ello puedes usar el siguiente docker-compose.yml

version: '3.1'

services:

```
postgresql-db:  
  image: postgres:latest  
  container_name: postgresql  
  environment:  
    POSTGRES_PASSWORD: PassWd!10  
    POSTGRES_HOST_AUTH_METHOD: trust  
  restart: always  
  ports:  
    - '5432:5432'  
  volumes:  
    - postgres_data:/var/lib/postgresql/data
```

volumes:

```
  postgres_data:
```

```
usuario@usuario-ad:~$ mkdir PostgreSQL  
usuario@usuario-ad:~$ ls -l  
total 44  
drwxr-xr-x 2 usuario usuario 4096 sep 10 00:28 Descargas  
drwxr-xr-x 2 usuario usuario 4096 sep  7 10:42 Documentos  
drwxr-xr-x 3 usuario usuario 4096 dic 10 18:25 Escritorio  
drwxr-xr-x 2 usuario usuario 4096 sep 16 12:05 Imágenes  
drwxr-xr-x 2 root   root   4096 nov 27 10:38 MongoDB  
drwxr-xr-x 2 usuario usuario 4096 sep  7 10:42 Música  
drwxrwxr-x 2 usuario usuario 4096 sep 10 00:44 NetBeansProjects  
drwxr-xr-x 2 usuario usuario 4096 sep  7 10:42 Plantillas  
drwxrwxr-x 2 usuario usuario 4096 dic 14 18:06 PostgreSQL  
drwxr-xr-x 2 usuario usuario 4096 sep  7 10:42 Público  
drwxr-xr-x 2 usuario usuario 4096 sep  7 10:42 Vídeos  
usuario@usuario-ad:~$ cd PostgreSQL  
usuario@usuario-ad:~/PostgreSQL$ sudo nano docker-compose.yml  
[sudo] contraseña para usuario:  
usuario@usuario-ad:~/PostgreSQL$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
usuario@usuario-ad:~/PostgreSQL$ docker-compose up -d  
Creating network "postgresql_default" with the default driver  
Creating volume "postgresql_postgres_data" with default driver  
Pulling postgresql-db (postgres:latest)...  
latest: Pulling from library/postgres  
e5ae68f74026: Pull complete  
7b8fcc7elad0: Pull complete  
7527d03e2f77: Pull complete  
80e55689f4d0: Pull complete  
8a79eb6d69c9: Pull complete  
397705f2d093: Pull complete  
de36ec4eb0a5: Pull complete  
08d878a022c1: Pull complete  
7677029670ff: Pull complete  
1d24b3d9557e: Pull complete  
e085b018338c: Pull complete  
063b09ff12e9: Pull complete  
a39fee215a44: Pull complete  
Digest: sha256:f76241d07218561e3d1a334eae6a5bf63c70b49f35ffecb7f020448e30e37390  
Status: Downloaded newer image for postgres:latest  
Creating postgresql ... done  
usuario@usuario-ad:~/PostgreSQL$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
6e94e1f2b759 postgres:latest "docker-entrypoint.s..." 37 seconds ago Up 15 seconds 0.0.0.0:5432->5432/tcp, :::5432->5432/tcp postgresql  
usuario@usuario-ad:~/PostgreSQL$ █
```

Archivo Editar Ver Buscar Terminal Ayuda

```
GNU nano 4.8
version: '3.1'

services:

postgresql-db:
    image: postgres:latest
    container_name: postgresql
    environment:
        POSTGRES_PASSWORD: PassWd!10
        POSTGRES_HOST_AUTH_METHOD: trust
    restart: always
    ports:
        - '5432:5432'
    volumes:
        - postgres_data:/var/lib/postgresql/data
volumes:
    postgres_data:
```

Download | DBeaver Core X +

< > C ☰ 🔒 dbeaver.io/download/

Would you like to make Opera your everyday browser? [How do I do that?](#)

Yes, set it as default browser x

Mac OS X

- Install from Microsoft Store
- Chocolatey (choco installdbeaver)
- Mac OS X Intel (dmg)
- Mac OS X M1 (dmg) – beta
- Brew Cask (brew install --caskdbeaver-community)
- MacPorts (sudo port installdbeaver-comm

Linux [Search](#) [Copy](#)

- Linux Debian package 64 bit (installer)
- Linux RPM package 64 bit (installer)
- Linux 64 bit (zip)
- Linux x86 64 bit (zip without Java included)
- Linux ARM 64 bit (zip without Java included)
- Snap (sudo snap installdbeaver-ce)
- Flatpak (flatpak install flathub io.dbeaver.DBeaverCommunity)

Eclipse Plugin

- Update site URL: <https://dbeaver.io/update/latest/> (Multiplatform)
- Eclipse Marketplace direct install: <https://marketplace.eclipse.org/content/dbeaver>

Note: plugin is compatible with Eclipse platform (from Neon to 2020)

dbeaver-ce 21.3.1

Responsable: DBeaver Corp <dbeaver@jkiss.org>

Tamaño: 135280 KiB

DBeaver Community
Universal Database Manager and SQL Client

Mock data generator
Advanced schema compare/migration tools
Data compare tool
Office formats support (XLS) for data export
Advanced SQL execution plan viewer
Integrated Git (version control for scripts and

dbl

Todas las aplicaciones dbeaver-ce_21.3.1_amd64.deb
 Accesorios dbeaver-ce
 Gráficos
 Internet
 Oficina
 Otras
 Programación
 Sonido y video
 Administración
 Preferencias
 Lugares
 Archivos recientes

+ Añadir al panel
 ☐ Añadir al escritorio
 ★ Añadir a las aplicaciones favoritas
 Desinstalar

DB Browser for SQLite

dbeaver-ce
 Universal Database Manager and SQL Client.

Seleccione un asistente

Seleccione un asistente
 Database connection

Asistentes:
 escribir texto de filtro

General
 DBBeaver
Database Connection
 Diagrama ER
 Proyecto de base de datos

< Anterior Siguiente > Cancelar Finalizar

Driver settings

Download driver files
 Download PostgreSQL driver files

PostgreSQL driver files are missing.
 These files can be downloaded automatically.
 Force download / overwrite

Files required by driver

File	Version	Description
org.postgresql:postgresql:RELEASE	42.2.20	PostgreSQL JDBC Driver Pos
org.checkerframework:checker-qual:3.5.0	3.5.0	Checker Qual is the set of an
net.postgis:postgis-jdbc:RELEASE	2.5.0	

You can change driver version by clicking on version column.
 Then you can choose one of the available versions.

Or you can obtain driver files by yourself and add them in driver editor.
[Vendor website](#) [Download configuration](#)

Edit Driver Cancelar Download

Archivo Editar Navegar Buscar Editor SQL Base de Datos Ventana Ayuda

Nuevo
 Grabar Ctrl+N
 Grabar todo Ctrl+S
 Guardar todo Ctrl+W
 Cerrar Ctrl+P
 Renombrar F2
 Refrescar F5
 Importar
 Exportar
 Propiedades
 Salir
 Convertir delimitadores de línea a
 Cambiar espacio de trabajo...
 Reiniciar ajustes de interfaz
 Reiniciar estado del espacio de trabajo
 Salida de emergencia
 Scripts

DBeaver Sample Database (SQLite)

Conectarse a la base de datos

Seleccione su base de datos
 Cree una nueva conexión a la base de datos. Encuentre su driver de la base de datos en la lista inferior.

Escribe parte del nombre de la base de datos/driver para filtrar Sort by: Title Score

All Popular SQL NoSQL Analytical Timeseries Embedded Hadoop / BigData Full-text search Graph databases

SQLite DB2 DuckDB MariaDB MySQL Oracle

PostgreSQL SQL Server Apache Drill Apache Hive Apache Ignite Apache Phoenix

< Anterior Siguiente > Cancelar Finalizar

Configuración de la conexión "postgres"

Ajustes de conexión PostgreSQL ajustes de conexión

Inicialización Comandos de shell Identificación de Transacciones General Metadatos Errores y timeouts Editor de datos Editor SQL

General PostgreSQL Driver properties SSH Proxy SSL Server Host: localhost Port: 5432 Database: postgres

Authentication: Database Native Nombre de usuario: postgres Contraseña: Save password loc

Advanced Session role: Local Client: You can use variables in connection parameters.

Driver name: PostgreSQL Edit Driver Settings

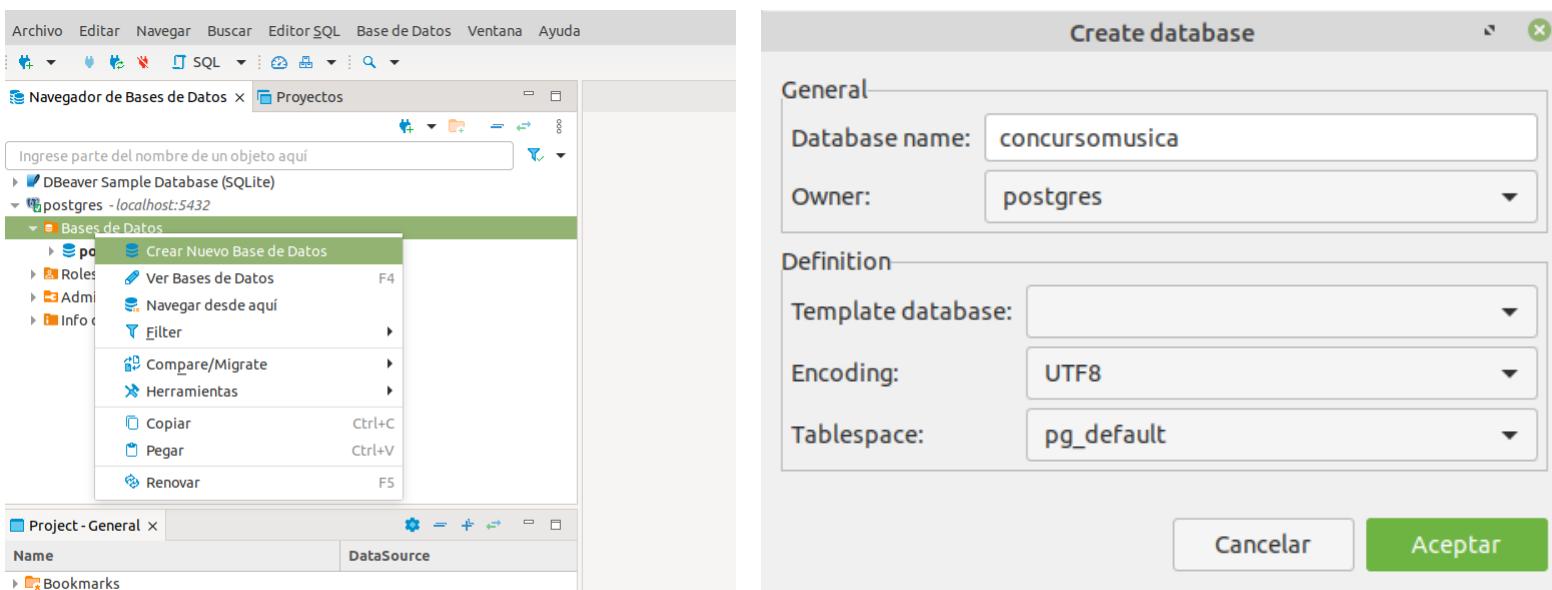
Contraseña: PassWd!10 como en el docker

2. - Ahora vamos a importar a PostgreSQL los datos de la base de datos **concursumusica.db** que teníamos en SQLite.

Antes de nada vamos a crear una base de datos llamada **concursumusica** en PostgreSQL. Para ello usamos el cliente DBeaver.

Luego instalamos pgloader (**sudo apt install pgloader**). Es una herramienta que permite convertir distintas bases de datos a PostgreSQL. Por último, ejecutamos el siguiente comando:

```
pgloader concursomusica.db postgres://postgres@localhost:5432/concursomusica
```



```
usuario@usuario-ad:~$ sudo apt install pgloader
[sudo] contraseña para usuario:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  bridge-utils ubuntu-fan
Utilice «sudo apt autoremove» para eliminarlos.
Se instalarán los siguientes paquetes adicionales:
  freetds-common freetds-dev libct4 libsybdb5
Se instalarán los siguientes paquetes NUEVOS:
  freetds-common freetds-dev libct4 libsybdb5 pgloader
0 actualizados, 5 nuevos se instalarán, 0 para eliminar y 229 no actualizados.
Se necesita descargar 27,4 MB de archivos.
Se utilizarán 33,5 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://archive.ubuntu.com/ubuntu focal/main amd64 freetds-common all 1.1.6-1.1 [24,8 kB]
Des:2 http://archive.ubuntu.com/ubuntu focal/main amd64 libct4 amd64 1.1.6-1.1 [152 kB]
Des:3 http://archive.ubuntu.com/ubuntu focal/main amd64 libsybdb5 amd64 1.1.6-1.1 [179 kB]
Des:4 http://archive.ubuntu.com/ubuntu focal/main amd64 freetds-dev amd64 1.1.6-1.1 [268 kB]
Des:5 http://archive.ubuntu.com/ubuntu focal/universe amd64 pgloader amd64 3.6.1-1 [26,7 MB]
```

```

usuario@usuario-ad:~/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Hoja_2$ ls -l
total 32
-rw-rw-r-- 1 usuario usuario 20480 dic 14 18:44 concursomusica.db
-rw-rw-rw- 1 usuario usuario 952 dic 10 19:31 pom.xml
drwx----- 4 usuario usuario 4096 dic 10 19:31 src
drwxrwxr-x 6 usuario usuario 4096 dic 10 19:35 target
usuario@usuario-ad:~/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Hoja_2$ pgloader concursomusica.db postgresql://postgres@localhost:5432/concursomusica
2021-12-14T17:47:16.018000Z LOG pgloader version "3.6.1"
2021-12-14T17:47:16.061000Z LOG Migrating from #<SQLITE-CONNECTION sqlite:///home/usuario/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Hoja_2/concursomusica.db {1005D57BD3}>
2021-12-14T17:47:16.062000Z LOG Migrating into #<PGSQL-CONNECTION postgres://postgres@localhost:5432/concursomusica {1005E9B143}>
2021-12-14T17:47:16.447000Z LOG report summary reset
      table name    errors    rows    bytes    total time
-----+-----+-----+-----+-----+-----+-----+
        fetch      0       0       0.000s
  fetch meta data      0       4       0.043s
 Create Schemas      0       0       0.002s
Create SQL Types      0       0       0.008s
  Create tables      0       4       0.069s
 Set Table OIDs      0       2       0.018s
-----+-----+-----+-----+-----+-----+
        grupos      0      13      0.5 kB      0.044s
      canciones      0       8      0.2 kB      0.037s
-----+-----+-----+-----+-----+-----+
COPY Threads Completion      0       4       0.034s
Index Build Completion      0       2       0.050s
  Create Indexes      0       2       0.031s
 Reset Sequences      0       2       0.022s
  Primary Keys      0       2       0.005s
Create Foreign Keys      0       1       0.004s
  Create Triggers      0       0       0.002s
 Install Comments      0       0       0.000s
-----+-----+-----+-----+-----+-----+
  Total import time ✓ 21      0.6 kB      0.148s
usuario@usuario-ad:~/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Hoja_2$ 

```

OJO: En el directorio donde esté el archivo .db del SQLite
Ya tenemos importada la base de datos

Table	Rows	Approximate Data
canciones	8	1 Hasta Nunca 2 Trampas al sol 3 The Nights 4 Hey Brother 5 Mas de 10 años 6 No Tengo Rival 7 Wasted Years 8 Aces High
grupos	13	1 IZAL 2 Lori Meyers 3 La Fuga 4 Avicil 5 Xenon 6 Iron Maiden 7 Don Omar 8 AC/DC 9 Nine Inch Nails 10 Marea 11 Red Hot Chilli Peppers 12 Mago de Oz 13 Desakato

3.- Añade un campo **votos** a la tabla canciones y rellénalo para las canciones que tengas.

The screenshot shows the pgAdmin interface for managing a PostgreSQL database named 'concursumusica'. The current window is focused on the 'canciones' table.

Table Properties:

- Table Name: canciones
- ID Objeto: 16394
- Propietario: postgres
- Columnas (Columns):

Column Name	#	Tipo de datos	Identidad	Collation	No Nulo
numcancion	1	bigserial		default	[]
titulo	2	text			[]
duracion	3	int8			[]
numgrupo	4	int8			[]

A context menu is open over the 'votos' column, with the option 'Crear Nuevo Columna' (Create New Column) highlighted.

Edit attribute column1 dialog box:

Name:	votos														
Properties:	<table border="1"> <tr> <td>Name</td> <td>Value</td> </tr> <tr> <td>Tipo de datos</td> <td>int8</td> </tr> <tr> <td>Identidad</td> <td></td> </tr> <tr> <td>Collation</td> <td></td> </tr> <tr> <td>No Nulo</td> <td>[]</td> </tr> <tr> <td>Por defecto</td> <td></td> </tr> <tr> <td>Comentario</td> <td></td> </tr> </table>	Name	Value	Tipo de datos	int8	Identidad		Collation		No Nulo	[]	Por defecto		Comentario	
Name	Value														
Tipo de datos	int8														
Identidad															
Collation															
No Nulo	[]														
Por defecto															
Comentario															

Buttons: Cancelar (Cancelar) and Aceptar (Aceptar).

Data View:

The main window shows the 'canciones' table data:

	numcancion	titulo	duracion	numgrupo	votos
1	1	Hasta Nunca	310	3	15
2	2	Trampas al sol	246	3	12
3	3	The Nights	176	4	9
4	4	Hey Brother	255	4	4
5	5	Mas de 10 anios	184	5	9
6	6	No Tengo Rival	233	5	7
7	7	Wasted Years	309	6	10
8	8	Aces High	270	6	11

4.- Realiza un programa que permite recoger números de canciones y, para cada canción, se incrementan sus votos en uno. El programa termina cuando se introduce el número 0. No se necesita comprobar que la canción existe, pero se tiene que notificar si se ha modificado los votos.

```
public class NewMain
{
    public static void main(String[] args)
    {
        // CREO LA CONEXION A LA BASE DE DATOS
        Connection conexion;
        try
        {
            Class.forName("org.postgresql.Driver");
            conexion = DriverManager.getConnection("jdbc:postgresql://127.0.0.1/concursomusica",
                "postgres", "PassWd!10");
            System.out.println("Conexion OK con concursomusica.db");

            // aniadirVotos(conexion);
            insertarCancionesGrupo(conexion);

        } catch (ClassNotFoundException ex)
        {
            Logger.getLogger(NewMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex)
        {
            System.out.println(ex.toString());
        }
    }

    private static void aniadirVotos(Connection conexion)
    {
        int valor;
        valor = Teclado.introInt("Numero de cancion [0 para salir]:");
        while (valor != 0)
        {

            String sql = "UPDATE canciones "
                + "SET votos = votos + 1 "
                + "WHERE numcancion = ?;";

            try ( PreparedStatement consultaPreparada = conexion.prepareStatement(sql))
            {
                consultaPreparada.setInt(1, valor);
                if (consultaPreparada.executeUpdate() > 0)
                {
                    System.out.println("Cancion numero " + valor + " actualizada.");
                } else
                {
                    System.out.println("Error al actualizar.");
                }
                valor = Teclado.introInt("Numero de cancion [0 para salir]:");
            } catch (SQLException ex)
            {
                System.out.println("Error del PreparedStatement");
                System.out.println(ex.toString());
            }
        }
    }
}
```

5.- Realiza un programa que permite recoger un número de grupo y, a continuación, pide por teclado los datos de varias canciones y las añade a la tabla canciones. En votos se carga un cero. El programa termina cuando se introduce ENTER para el título.

Para ejecutar la instrucción SQL INSERT hay que usar un objeto PreparedStatement y la instrucción SQL debe utilizar parámetros sustituibles.

```
private static void insertarCancionesGrupo(Conexion conexion)
{
    int numeroGrupo = Teclado.introInt("Numero de grupo:");

    String sql = "INSERT INTO canciones (numcancion, titulo, duracion, numgrupo, votos)"
        + "VALUES (?, ?, ?, ?, 0)";

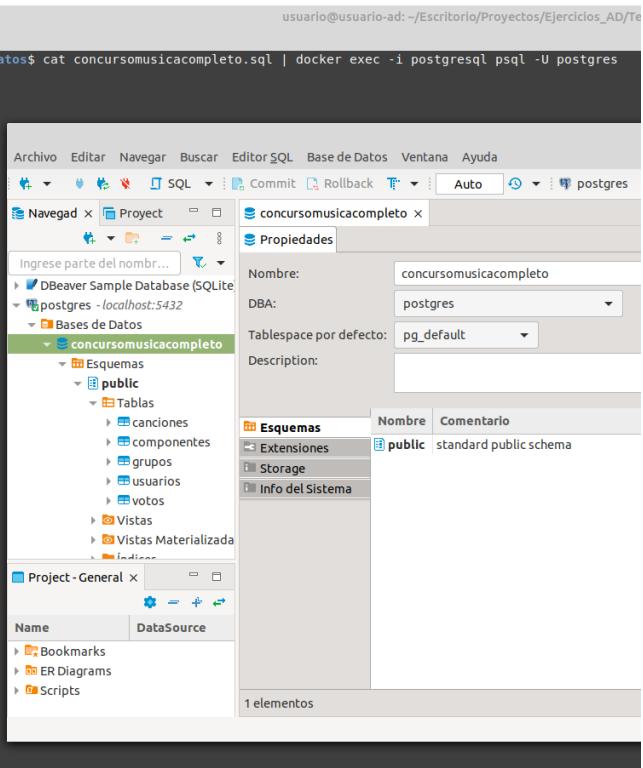
    while (true)
    try ( PreparedStatement consultaPreparada = conexion.prepareStatement(sql))
    {
        consultaPreparada.setInt(1, Teclado.introInt("Numero Cancion:"));
        String titulo = Teclado.introString("Titulo Cancion:");
        if (titulo.length() == 0)
            break;
        consultaPreparada.setString(2, titulo);
        consultaPreparada.setInt(3, Teclado.introInt("Duracion (seg):"));
        consultaPreparada.setInt(4, numeroGrupo);

        consultaPreparada.executeUpdate();
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
}
```

DB SQL Hoja 4

En los ejercicios de esta actividad tienes que usar la base de datos PostgreSQL **concursumusicacompleto**.

Descarga el script SQL e importalo en el contenedor de Docker donde tienes la base de datos. Para ello usa el comando: **cat concursomusicacompleto.sql | docker exec -i postgresql psql -U postgres**



The screenshot shows the DBeaver interface with the 'concursumusicacompleto' database selected. The left pane displays the database structure:

- Bases de Datos
 - concursumusicacompleto
 - Esquemas
 - public
 - Tablas
 - canciones
 - componentes
 - grupos
 - usuarios
 - votos
 - Vistas
 - Vistas Materializada

The right pane shows the properties for the 'concursumusicacompleto' database, including:

 - Nombre: concursomusicacompleto
 - DBA: postgres
 - Tablespace por defecto: pg_default

Below the properties pane, there is a table titled 'Esquemas' with one row:

Nombre	Comentario
public	standard public schema

At the bottom of the interface, there is a status bar with icons for connection, workspace, and other tools.

```
Archivo Editar Ver Buscar Terminal Ayuda
usuario@usuario-ad:~/Escritorio/Proyectos/Ejercicios_AD/Tema_4/Bases_de_Datos$ cat concursomusicacompleto.sql | docker exec -i postgresql psql -U postgres
SET
SET
SET
SET
DROP DATABASE
ERROR:  database "concursumusicacompleto" does not exist
ERROR:  current user cannot be dropped
ERROR:  role "postgres" already exists
ALTER ROLE
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
SET
UPDATE 1
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
SET
COMMENT
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
```

En esta base de datos se gestiona un concurso de música que se realiza en una página web. En esa página web los usuarios registrados pueden votar canciones de grupos de música (las canciones registradas en la base de datos). De cada grupo puede haber varias canciones.

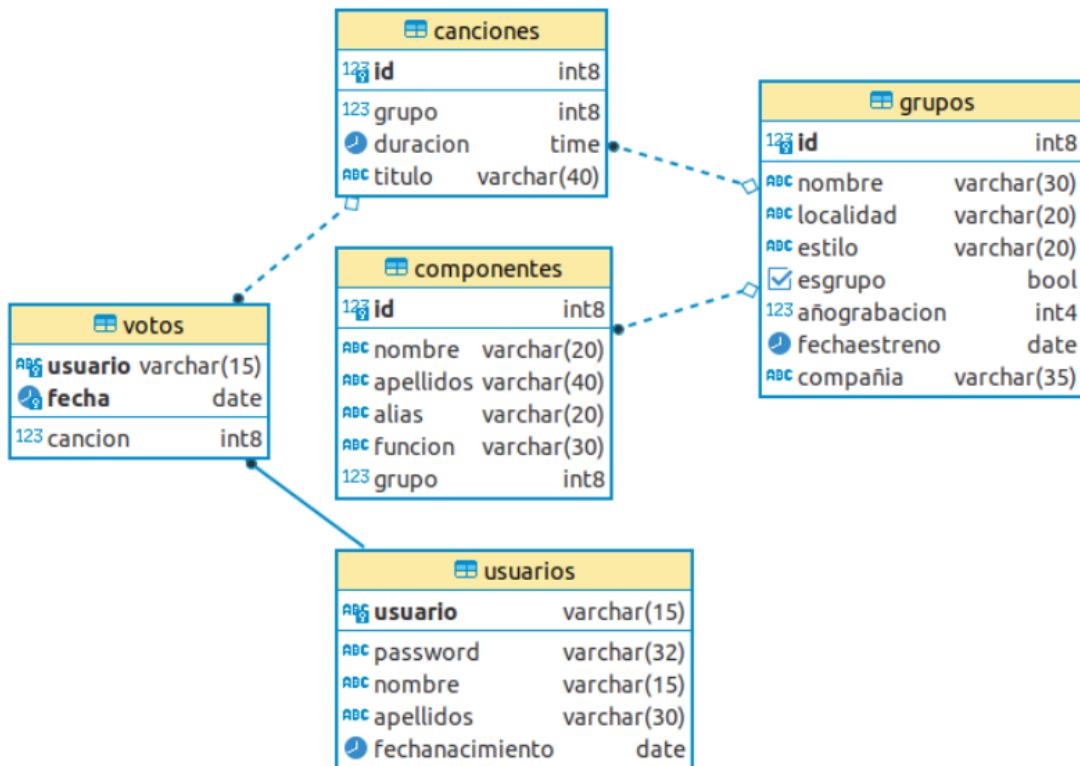
La tabla **grupos** contiene información sobre todos los grupos y artistas en solitario que participan con alguna canción en el concurso (pueden participar con varias). Cuando se trata de un artista en solitario, indica con false o 0 en la columna **esgrupo**. Por defecto en esta columna se carga true o 1. En **fechaestreno** se registra la fecha de la primera actuación en directo del grupo y en **añograbacion** el año en que grabaron el primer disco. El identificador del grupo **id** es una columna numérica y autoincrementada.

De cada grupo, se tiene en la tabla **componentes** información sobre los componentes del grupo. En la columna **función** se tienen valores como **batería, vocalista, voz y guitarra, etc.**

En la tabla **canciones** se tienen todas las canciones que se pueden votar. El identificador de cada canción es el número de canción (**id que es autoincrementada**). La duración es una columna de tipo **TIME**, por tanto, se representan sus datos en formato **HH:MM:SS**. La columna grupo es clave ajena y contiene el identificador del grupo al que pertenece la canción.

Los usuarios de la web se almacenan en la tabla **usuarios**. Cada usuario elige su identificador (**usuario**) y su contraseña. La contraseña debe almacenarse con encriptación **MD5**.

En la tabla **votos** se inserta una fila cada vez que un usuario vota una canción. En esta tabla es **PK fecha+usuario**, por lo que un usuario no pueda dar más de un voto en un mismo día. En la columna **cancion** se registra el número o identificador de canción votada en cada voto.



concursumusicacompleto canciones componentes usuarios votos

Propiedades Datos Diagrama ER

postgres Bases de Datos concursumusicacompleto Esquemas public

canciones Enter a SQL expression to filter results (use Ctrl + Space)

Grilla Texto Record

numcancion	grupo	duracion	titulo	total votos
38	38	14	00:02:27 Cigarettes	3
39	39	14	00:05:08 Just Like A Wall	1
40	40	18	00:03:12 Autocrítica	1
41	41	18	00:03:22 Salvese quien Pueda	1
42	42	18	00:04:01 Copenhagen	3
43	43	18	00:02:36 Al Respirar	3
44	44	12	00:04:23 El hombre bolígrafo	2
45	45	12	00:04:36 Polaroid	3
46	46	12	00:03:52 Aspiradora Espacial	0
47	47	11	00:02:46 La chica Vampira	1
48	48	17	00:04:40 Deli	1
49	49	17	00:03:50 Real Love	0
50	50	17	00:04:27 Stay Close	2
51	51	17	00:04:27 Seasun	1
52	52	13	00:05:12 Evolucion	1
53	53	13	00:04:01 El Tiempo Pasará	3
54	54	16	00:02:23 Bailando	3
55	55	16	00:03:41 Todo nos parece una mier	1
56	56	19	00:04:17 Luz Artificial	4
57	57	19	00:06:33 Melodrama	2
58	58	19	00:03:11 Que Vienes	0
59	59	20	00:04:18 How to Turn Rivers into All	1
60	60	20	00:04:01 8:24	0
61	61	20	00:05:23 Clouds Factory	4
62	62	21	00:03:15 Hallo	2
63	63	21	00:02:35 Con mi voz	2
64	64	22	00:03:42 El mundo se va a acabar	1
65	65	22	00:03:21 Titubeas	0
66	66	23	00:02:54 Las flores	2

Save Cancel Script Record

Propiedades Datos Diagrama ER

postgres Bases de Datos concursumusicacompleto Esquemas public Tablas

componentes Enter a SQL expression to filter results (use Ctrl + Space)

Grilla Texto Record

idcomp	nombre	apellido	alias	funcion	grupo	
1	Javi	Fernandez	[NULL]	Voz, piano, guitarra	4	
2	2	Jordi	Navarro	[NULL]	Guitarra y electrónica	4
3	3	Gülle	Mostaza	[NULL]	Voz y guitarra	6
4	4	Santi	Capote	[NULL]	Bajo	6
5	5	Gorka	Dresbaj	[NULL]	Guitarra y electrónica	4
6	6	José	Tejedor	[NULL]	Bajo	4
7	7	Sel	Lee	[NULL]	Batería	4
8	8	Javier	Valencia	[NULL]	Vocalista	3
9	9	Jesús	Gutiérrez	[NULL]	Bajo	3
10	10	Jaime	Gutiérrez	[NULL]	Batería	3
11	11	Manu	Jurado	[NULL]	Teclado	3
12	12	Enrique	Sánchez	Buby	Guitarra	3
13	13	Fran	Gómez	[NULL]	Vocalista	9
14	33	Alejandro	Méndez	[NULL]	Guitarra	13
15	39	Sean	Frutos	[NULL]	Vocalista	10
16	40	Jorge	Guirao	[NULL]	Guitarra	10
17	41	Javier	Vox	Javi	Guitarra y teclado	10
18	42	Fernando	Roble	Nando	Bajo	10
19	43	Francisco	Guirao	Fran	Batería	10
20	44	Oscar	Ferrer	[NULL]	Vocalista	2
21	45	Aarón	Sáez	[NULL]	Guitarra y teclados	2
22	46	Vicente	Llescas	[NULL]	Guitarra	2
23	52	Antonio	Lomas	[NULL]	Percusión	13
24	53	Miguel	López	[NULL]	Bajo Eléctrico	13
25	54	Alfredo	Núñez	[NULL]	Batería	13
26	55	Mikel	Izal	[NULL]	Vocalista	5
27	56	Alejandro	Jordá	[NULL]	Batería	5
28	57	Emanuel	Perez	[NULL]	Bajo eléctrico	5
29	58	Alberto	Perez	[NULL]	Guitarra	5

Save Cancel Script Record

Propiedades Datos Diagrama ER

postgres Bases de Datos concursumusicacompleto Esquemas public Tablas usuarios

usuarios Enter a SQL expression to filter results (use Ctrl + Space)

Grilla Texto Record

user	contraseña	nombre	apellidos	Fechanac	numvotos	
1	f547f0213884e47059f0532cfb7487a	Ana	García Herrero	1999-03-06	13	
2	02Elsa	843b632a1f159a265a2bc4432f5fb4da8a	Elsa	Frutos Núñez	1989-02-23	11
3	02Eva	4a8ae230f049a2a265a232d4ab7735e5f	Eva	Alvarez Martínez	1991-08-04	3
4	02Pedro	39ff4c11837a042e9a9e643b32efffff	Pedro	Pancrvo Hidalgo	1996-03-03	0
5	03ez	cd4ee254fb7b83270a4b9d685a98b721	Francisco	Freire Alonso	1993-07-25	2
6	03lemon	7352f5b4fc51fb68a6396441dc5b7f	Arturo	Abriñes salas	1993-07-25	2
7	03quezy	44ff4704b7b03270a4b9d685a98b721	Sean	Matthews	1993-07-25	2
8	03squeozy	f818a704b3d520ea4aa5f0c6595b4c4	Sonia	Rey Ortega	1993-07-25	1
9	04Carlos1a	cc01164c660a136cc638004d8ad2e1b	Carlos	Labra Mora	1996-02-10	0
10	05User2	841bddd4b26732c507e1b74b6c38fa	Carla	Antón Martinez	1996-12-09	2
11	05User3	b54ec2d573b6bdada5ca61b35fc8e640	Jonathan	León Suárez	1996-03-05	1
12	05User4	294f236f3e522a9908c525377a2c0d6830	Alfredo	Marco Roura	1993-08-30	0
13	10galas	13fcf2d525485e878b24213b26ek4ck7a	Agata	Cortés Valverde	1997-01-08	0
14	10asdfgh	b546f7e156ee923cda42842d5e40178	Asunta	Serra Montaner	1992-12-12	0
15	10messy10	b914b2921d6b3821a75457a658a49236	Laura	Gómez Cuenca	1990-01-15	0
16	10qwert	63aea4ca6e20fe38f80853d944148f	Esteban	Rodríguez Santiago	1990-12-12	0
17	11epami	48eb4a24cbdb9bb614086ba37cafb	Maria	Labrador Revuelta	2016-01-12	4
18	11Eustaquio	4b72f2362338c195bab0576b4c92d3c1	Eustaquio	García García	1990-01-08	3
19	11Fernando	ff54f1f60f352d05a4f2ca8a8fbaf3	Fernando	Fernández Huidobro	1997-01-09	0
20	11Jorge	acf7abc161a46c774fa27863509db	Jorge	Diez Gil	1996-01-07	0
21	13Antonio	e8cb202890f5be03b23d3e3d8eef	Antonio	Fernández Montes	1992-05-05	0
22	13Carlota	5947a0cad25041544dfe391c3e209	Carlota	Revuelta Herrero	1994-05-05	1
23	13Concepcion	6fc8c1aa897fbc04ad30c8d12571cb	Concepcion	Cano Higuera	1980-05-18	2
24	13LourHerna	d9f7c1a911eaabbcc00542db55ab4	Lourdes	Hernández González	1995-11-20	3
25	13Lucia	3c565b9550a33254b827172a47c333b	Lucía	Calderón García	1993-05-25	3
26	16Ana	e044ca35f4d3382df7a8e0a4b193a6c	Ana Patricia	Fernández González	1999-05-12	1
27	16Carlos	0938aa921713474782108ab9c06ef214	Carlos	Labrador Amieva	1996-07-08	3
28	16Victor	575a5fb9d90fbca38fbc53c5d8e029	Victor	Baldo Suárez	1996-01-09	2
29	16Yeray	5c0f1d3calcd51c238bee7cb7b0583f	Yeray	Peña Sisniega	1996-09-11	0

Save Cancel Script Record

Propiedades Datos Diagrama ER

postgres Bases de Datos concursumusicacompleto Esquemas public Tablas votos

votos Enter a SQL expression to filter results (use Ctrl + Space)

Grilla Texto Record

usuario	fecha	cancion
1	2018-10-21	[NULL]
2	2018-10-21	[NULL]
3	2018-10-22	[NULL]
4	2018-06-07	[NULL]
5	2018-05-28	[NULL]
6	2018-05-27	[NULL]
7	2018-08-14	[NULL]
8	2018-10-13	[NULL]
9	2018-06-05	[NULL]
10	2018-10-21	[NULL]
11	2018-10-21	[NULL]
12	2018-06-07	[NULL]
13	2018-05-28	[NULL]
14	2018-06-03	[NULL]
15	2018-05-28	[NULL]
16	2018-06-06	[NULL]
17	2018-05-27	[NULL]
18	2018-10-16	[NULL]
19	2018-10-16	[NULL]
20	2018-05-29	[NULL]
21	2018-10-13	[NULL]
22	2018-10-22	[NULL]
23	2018-06-06	[NULL]
24	2018-06-07	[NULL]
25	2018-06-07	[NULL]
26	2018-06-07	[NULL]

Save Cancel Script Record

Propiedades Datos Diagrama ER

postgres Bases de Datos concursumusicacompleto Esquemas public Tablas votos

votos Enter a SQL expression to filter results (use Ctrl + Space)

Grilla Texto Record

usuario	fecha	cancion
1	2018-10-21	[NULL]
2	2018-10-21	[NULL]
3	2018-10-22	[NULL]
4	2018-06-07	[NULL]
5	2018-05-28	[NULL]
6	2018-05-27	[NULL]
7	2018-08-14	[NULL]
8	2018-10-13	[NULL]
9	2018-06-05	[NULL]
10	2018-10-21	[NULL]
11	2018-10-21	[NULL]
12	2018-06-07	[NULL]
13	2018-05-28	[NULL]
14	2018-06-03	[NULL]
15	2018-05-28	[NULL]
16	2018-06-06	[NULL]
17	2018-05-27	[NULL]
18	2018-10-16	[NULL]
19	2018-10-16	[NULL]
20	2018-05-29	[NULL]
21	2018-10-13	[NULL]
22	2018-10-22	[NULL]
23	2018-06-06	[NULL]
24	2018-06-07	[NULL]
25	2018-06-07	[NULL]
26	2018-06-07	[NULL]

Save Cancel Script Record

EJERCICIO 1

Realiza un programa que permite añadir un usuario a la tabla usuarios con los datos recogidos por teclado.

No es necesario validar los datos introducidos.

Si que hay que comprobar si al hacer la inserción se produce excepción por clave duplicada debido a que el usuario ya exista.

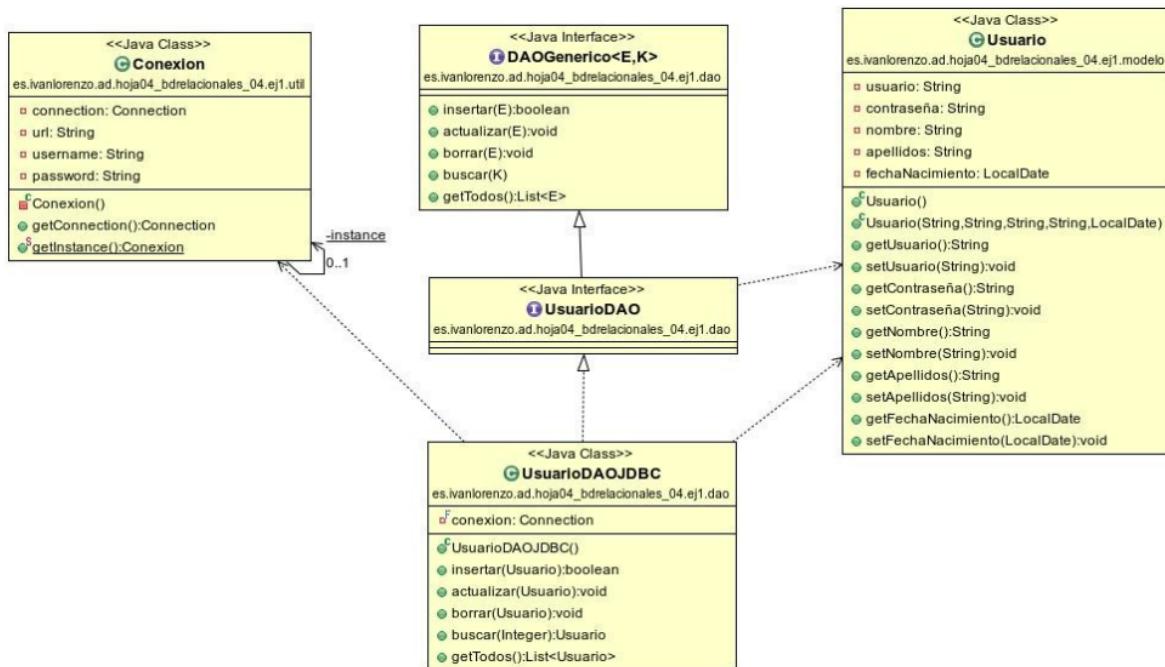
A partir de ahora en nuestras aplicaciones con bases de datos tendremos una clase Conexion que seguirá el patrón de diseño **Singleton**. Busca información acerca de él y pregunta al profesor en caso de dudas.

Además, utilizaremos el patrón DAO para los accesos a bases de datos. Para ello crearemos una interface genérica DAOGenerico<E,K> con las operaciones básicas.

UsuarioDAO será una interface que herede de DAOGenerico cuya entidad (E) sea Usuario y cuya clave (K) sea un String (su clave primaria).

Por último, realizaremos la clase UsuarioDAOImpl que implemente UsuarioDAO y en la que desarrollaremos los métodos que necesitemos (en este primer ejercicio únicamente el método insertar).

El diagrama de clases sería similar al siguiente:



Clase Main

```
public class NewMain
{
    public static void main(String[] args)
    {
        // TODO code application logic here
        UsuarioDAO dao = new UsuarioDAOJDBC();
        Usuario usuario = recogerDatos();
        dao.insertar(usuario);
    }

    public static Usuario recogerDatos(){
        String user = Teclado.introString("User:");
        String pasw = Teclado.introString("Password:");
        String name = Teclado.introString("Nombre:");
        String surn = Teclado.introString("Apellidos:");
        LocalDate date = Teclado.introFecha("Fecha de nacimiento");

        return new Usuario(user, pasw, name, surn, date);
    }
}
```

Interfaz DAO Genérico

```
public interface DAOGenerico<E, K>
{
    public boolean insertar(E entity);
    public void actualizar(E entity);
    public void borrar(E entity);
    public void buscar(E entity);
    public List<E> getTodos();
}
```

Interfaz Usuario DAO

```
public interface UsuarioDAO extends DAOGenerico<Usuario, String>
{
    public List<Usuario> getUsuariosMayoresEdad();
}
```

Clase Usuario DAO JDBC

```
public class UsuarioDAOJDBC implements UsuarioDAO
{
    private final Connection conexion;

    public UsuarioDAOJDBC()
    {
        this.conexion = Conexion.getInstance().getConnection();
    }

    @Override
    public boolean insertar(Usuario usuario)
    {
        boolean insertado = false;
        PreparedStatement consultaPreparada = null;
        String sql = "INSERT INTO usuarios (nombre, apellidos, \"user\", contraseña, fechanac)"
                    + "VALUES(?, ?, ?, md5(?), ?)";
        try
        {
            consultaPreparada = conexion.prepareStatement(sql);

            consultaPreparada.setString(1, usuario.getNombre());
            consultaPreparada.setString(2, usuario.getApellidos());
            consultaPreparada.setString(3, usuario.getUsuario());
            consultaPreparada.setString(4, usuario.getContraseña());
            consultaPreparada.setObject(5, usuario.getFechaNacimiento());

            int n = consultaPreparada.executeUpdate();
            if(n == 1){
                System.out.println("Insertado Correctamente");
                insertado = true;
            }
        } catch (SQLException ex)
        {
            System.out.println("Error del PreparedStatement");
            System.out.println(ex.toString());
        }
        return insertado;
    }
}
```

```

@Override
public void actualizar(Usuario entity)
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public void borrar(Usuario entity)
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public void buscar(Usuario entity)
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public List<Usuario> getTodos()
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public List<Usuario> getUsuariosMayoresEdad()
{
    throw new UnsupportedOperationException("Not supported yet.");
}

}

```

EJERCICIO 2

Realiza un que presenta el siguiente menú (en este caso no es obligatorio utilizar el patrón DAO ya que incrementaría la complejidad del ejercicio):

- 1.- Listado de grupos
- 2.- Listado de canciones
- 3.- Número de canciones por grupo
- 4.- Canciones de un grupo
- 5.- Las 5 canciones más votadas
- 6.- Grupos sin canciones
- 7.- Los últimos 5 votos
- 8.- Eliminar canciones de un grupo
- 9.- Modificar datos de grupo

Y escribe en pantalla los resultados de la opción seleccionada.

```
public class Grupo
{
    private BigDecimal codgrupo;
    private String nombre;
    private String localidad;
    private String estilo;
    private boolean esgrupo;
    private int annograb;
    private Date fechaestreno;
    private String compania;
    private List<Cancion> canciones;

    public Grupo()
    {
        canciones = new ArrayList();
    }

    public void addCancion(Cancion cancion)
    {
        this.canciones.add(cancion);
    }

    public List<Cancion> getCanciones()
    {
        return canciones;
    }

    public void setCanciones(List<Cancion> canciones)
    {
        this.canciones = canciones;
    }

    public BigDecimal getCodgrupo()
    {
        return codgrupo;
    }

    public void setCodgrupo(BigDecimal codgrupo)
    {
        this.codgrupo = codgrupo;
    }
}



---


    public String getNombre()
    {
        return nombre;
    }

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public String getLocalidad()
    {
        return localidad;
    }

    public void setLocalidad(String localidad)
    {
        this.localidad = localidad;
    }

    public String getEstilo()
    {
        return estilo;
    }

    public void setEstilo(String estilo)
    {
        this.estilo = estilo;
    }

    public boolean isEsgrupo()
    {
        return esgrupo;
    }

    public void setEsgrupo(boolean esgrupo)
    {
        this.esgrupo = esgrupo;
    }

    public int getAnnograb()
    {
        return annograb;
    }
}
```

```
public void setAnnograb(int annograb)
{
    this.annograb = annograb;
}

public Date getFechaestreno()
{
    return fechaestreno;
}

public void setFechaestreno(Date fechaestreno)
{
    this.fechaestreno = fechaestreno;
}

public String getCompania()
{
    return compania;
}

public void setCompania(String compania)
{
    this.compania = compania;
}

}
```

Clase Grupo para almacenar los resultados de las consultas de la tabla grupos.

Los atributos se corresponden con las columnas de la tabla grupos, además de una lista de canciones por la relación 1-N.

Los métodos son los getter y setter de los atributos y un método para llenar la lista de canciones.

```

public class Cancion {
    private int numcancion;
    private int grupo;
    //Desde el principio con objeto grupo
    private Time duracion;
    private String titulo;
    private int total_votos;
    private Grupo grupoObjeto;
    private List<Voto> votos;

    public Cancion()
    {}

    public List<Voto> getVotos()
    {
        return votos;
    }

    public void setVotos(List<Voto> votos)
    {
        this.votos = votos;
    }

    public Grupo getGrupoObjeto()
    {
        return grupoObjeto;
    }

    public void setGrupoObjeto(Grupo grupoObjeto)
    {
        this.grupoObjeto = grupoObjeto;
    }

    public int getNumcancion()
    {
        return numcancion;
    }

    public void setNumcancion(int numcancion)
    {
        this.numcancion = numcancion;
    }
}

public int getGrupo()
{
    return grupo;
}

public void setGrupo(int grupo)
{
    this.grupo = grupo;
}

public Time getDuracion()
{
    return duracion;
}

public void setDuracion(Time duracion)
{
    this.duracion = duracion;
}

public String getTitulo()
{
    return titulo;
}

public void setTitulo(String titulo)
{
    this.titulo = titulo;
}

public int getTotal_votos()
{
    return total_votos;
}

public void setTotal_votos(int total_votos)
{
    this.total_votos = total_votos;
}
}

```

Clase Canción para almacenar los resultados de las consultas de la tabla canciones.

Los atributos se corresponden con las columnas de la tabla canciones, además de un objeto grupo por la relación 1-N y una lista de votos por la relación 1-N.

Los métodos son los getter y setter de los atributos.

```
public class Voto {  
    private String usuario;  
    private Date fecha;  
    private Cancion cancion;  
  
    public Voto()  
    {  
    }  
  
    public String getUsuario()  
    {  
        return usuario;  
    }  
  
    public void setUsuario(String usuario)  
    {  
        this.usuario = usuario;  
    }  
  
    public Date getFecha()  
    {  
        return fecha;  
    }  
  
    public void setFecha(Date fecha)  
    {  
        this.fecha = fecha;  
    }  
  
    public Cancion getCancion()  
    {  
        return cancion;  
    }  
  
    public void setCancion(Cancion cancion)  
    {  
        this.cancion = cancion;  
    }  
}
```

Clase Voto para almacenar los resultados de las consultas de la tabla votos.

Los atributos se corresponden con las columnas de la tabla votos, además de un objeto canción por la relación 1-N.

Los métodos son los getter y setter de los atributos.

```

public class GestorDB
{
    private Connection conexion;
    public GestorDB()
    {
        try{
            Class.forName("org.postgresql.Driver");
            conexion = DriverManager.getConnection(
                "jdbc:postgresql://127.0.0.1/concursomusicacompleto",
                "postgres", "PassWd!10");
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Opción 1: El listado de grupos mostrará el código del grupo, el nombre, la localidad y el estilo y debe estar ordenado por el id.

```

public List<Grupo> listadoGrupos()
{
    List<Grupo> grupos = new ArrayList();
    String sql = "SELECT * "
        + "FROM grupos ORDER BY codgrupo ASC";

    try
    {
        Statement st = conexion.createStatement();
        ResultSet result = st.executeQuery(sql);

        while (result.next())
        {
            Grupo grupo = new Grupo();

            grupo.setCodgrupo(result.getBigDecimal("codgrupo"));
            grupo.setNombre(result.getString("nombre"));
            grupo.setLocalidad(result.getString("localidad"));
            grupo.setEstilo(result.getString("estilo"));
            grupo.setAnnograb(result.getInt("annograb"));
            grupo.setCompania(result.getString("compania"));
            grupo.setEsgrupo(result.getBoolean("esgrupo"));
            grupo.setFechaestreno(result.getDate("fechaestreno"));

            grupos.add(grupo);
        }
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
    return grupos;
}

```

Opción 2: El listado de canciones debe estar ordenado por nombre de grupo y debe tener este formato:

```
public List<Grupo> listadoGruposCanciones() {
    List<Grupo> grupos = new ArrayList();
    try {
        String sql_grupo = "SELECT * FROM grupos ORDER BY nombre ASC";
        Statement st = conexion.createStatement();
        ResultSet resultGrupo = st.executeQuery(sql_grupo);
        while (resultGrupo.next()) {
            int codGrupo = resultGrupo.getInt("codgrupo");

            Grupo grupo = new Grupo();

            grupo.setCodgrupo(resultGrupo.getBigDecimal("codgrupo"));
            grupo.setNombre(resultGrupo.getString("nombre"));
            grupo.setLocalidad(resultGrupo.getString("localidad"));
            grupo.setEstilo(resultGrupo.getString("estilo"));
            grupo.setAnnograb(resultGrupo.getInt("annograb"));
            grupo.setCompania(resultGrupo.getString("compania"));
            grupo.setEsgrupo(resultGrupo.getBoolean("esgrupo"));
            grupo.setFechaestreno(resultGrupo.getDate("fechaestreno"));

            String sql_cancion = "SELECT * FROM canciones "
                + "WHERE grupo = " + codGrupo + ";";
            Statement st2 = conexion.createStatement();
            ResultSet resultCancion = st2.executeQuery(sql_cancion);

            while (resultCancion.next()) {
                Cancion cancion = new Cancion();

                cancion.setDuracion(resultCancion.getTime("duracion"));
                cancion.setGrupo(resultCancion.getInt("grupo"));
                cancion.setNumcancion(resultCancion.getInt("numcancion"));
                cancion.setTitulo(resultCancion.getString("titulo"));
                cancion.setTotal_votos(resultCancion.getInt("total_votos"));

                grupo.addCancion(cancion);
            }
            grupos.add(grupo);
        }
    } catch (SQLException ex) {
        System.out.println(ex.toString());
    }
    return grupos;
}
```

Opción 3: Deben mostrarse todos los grupos y, para cada grupo, el número de canciones que tiene.

Lo hacemos desde el main con el método listadoGruposCanciones() haciendo un grupo.getCancelones().size.

Opción 4: Se pide un nombre de grupo y muestran los datos de las canciones del grupo (id, título y duración).

```
public List<Cancion> cancionesUnGrupo(String grupo)
{
    List<Cancion> canciones = new ArrayList();
    try
    {
        String sql = "SELECT numcancion, duracion, titulo FROM canciones "
                    + "INNER JOIN grupos ON canciones.grupo = grupos.codgrupo "
                    + "WHERE grupos.nombre = '" + grupo + "'";

        Statement st = conexion.createStatement();
        ResultSet result = st.executeQuery(sql);

        while (result.next())
        {
            int numCancion = result.getInt("numcancion");
            Time duracion = result.getTime("duracion");
            String titulo = result.getString("titulo");

            Cancion cancion = new Cancion();

            cancion.setNumcancion(numCancion);
            cancion.setDuracion(duracion);
            cancion.setTitulo(titulo);

            canciones.add(cancion);
        }
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
    return canciones;
}
```

Opción 5: Se deben mostrar, para las canciones con más votos, títulos de canciones y nombres de los grupos.

```
public List<Cancion> cancionesMasVotadas()
{
    List<Cancion> canciones = new ArrayList();
    try
    {
        String sql = "SELECT canciones.total_votos, canciones.titulo, grupos.nombre FROM canciones "
                    + "INNER JOIN grupos ON canciones.grupo = grupos.codgrupo "
                    + "ORDER BY total_votos DESC";

        Statement st = conexion.createStatement();
        ResultSet result = st.executeQuery(sql);

        while (result.next())
        {
            Cancion cancion = new Cancion();
            Grupo grupo = new Grupo();

            cancion.setTotal_votos(result.getInt("total_votos"));
            cancion.setTitulo(result.getString("titulo"));
            grupo.setNombre(result.getString("nombre"));
            cancion.setGrupoObjeto(grupo);

            canciones.add(cancion);
        }
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
    return canciones;
}
```

Opción 6: Se muestra un listado de nombres de grupos sin canciones en la tabla canciones

Se usa el método listado GruposCanciones() en el main y se filtra por el size de la lista de canciones de cada grupo.

Opción 7: Se muestran datos de los 5 votos más recientes. Hay que mostrar el título de la canción votada, el grupo al que pertenece y la fecha del voto.

```
public List<Voto> votosMasRecientes()
{
    List<Voto> votos = new ArrayList();
    try
    {
        String sql = "SELECT canciones.titulo, votos.fecha, grupos.nombre FROM canciones "
                    + "INNER JOIN votos ON canciones.numcancion = votos.cancion "
                    + "INNER JOIN grupos ON canciones.grupo = grupos.codgrupo "
                    + "ORDER BY votos.fecha DESC LIMIT 5";
        Statement st = conexion.createStatement();
        ResultSet result = st.executeQuery(sql);

        while (result.next())
        {
            String nombreGrupo = result.getString("nombre");
            String titulo = result.getString("titulo");
            Date fecha = result.getDate("fecha");

            Cancion cancion = new Cancion();
            Grupo grupo = new Grupo();
            Voto voto = new Voto();

            voto.setFecha(fecha);
            cancion.setTitulo(titulo);
            grupo.setNombre(nombreGrupo);

            cancion.setGrupoObjeto(grupo);
            voto.setCancion(cancion);
            //Voto[Cancion(Grupo)]

            votos.add(voto);
        }
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
    return votos;
}
```

Opción 8: Se pide por teclado el nombre de un grupo y, si existe, se eliminan todas las canciones del grupo. Como está restringido el borrado de canciones si tienen votos, se han de eliminar primero los votos de las canciones del grupo. *NOTA: Buscad primero cómo realizar un DELETE con Postgresql.*

```
public int eliminarCancionesGrupo(String nombreGrupo)
{
    int resultCanciones = 0;
    try
    {
        String sqlVotos = "DELETE from votos "
            + "USING canciones, grupos "
            + "WHERE votos.cancion = canciones.numcancion "
            + "AND canciones.grupo = grupos.codgrupo "
            + "AND grupos.nombre = '" + nombreGrupo + "' ;";

        String sqlCanciones = "DELETE FROM canciones "
            + "WHERE grupo IN "
            + "(SELECT codgrupo FROM grupos WHERE nombre = '" + nombreGrupo + "' );";

        //Primero hay que borrar los votos de las canciones que queremos borrar
        //Porque el DELETE no está en cascada

        Statement stVotos = conexion.createStatement();
        int resultVotos = stVotos.executeUpdate(sqlVotos);
        //Result es el numero de filas afectadas por Update

        //Una vez borrados los votos ya podemos borrar las canciones.

        Statement stCanciones = conexion.createStatement();
        resultCanciones = stCanciones.executeUpdate(sqlCanciones);
        //Result es el numero de filas afectadas por Update
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
    return resultCanciones;
}
```

Opción 9: Se pide por teclado el nombre de un grupo y, si existe, se escribe en pantalla, el estilo, el año de primera grabación de un disco, la fecha de la primera actuación, la localidad y la compañía. A continuación, se escribe un menú preguntando cual de los anteriores datos se quiere modificar, incluido el nombre del grupo, y se modifica ese dato con el nuevo valor que se introduzca por teclado.

```
public void modificarGrupo(String nombreGrupo)
{
    try
    {
        String sql = "SELECT * FROM grupos WHERE nombre = '" + nombreGrupo + "'";

        Statement st = conexion.createStatement();
        ResultSet result = st.executeQuery(sql);
        result.next();

        int codgrupo = result.getInt("codgrupo");
        String nombre = result.getString("nombre");
        String localidad = result.getString("localidad");
        String estilo = result.getString("estilo");
        boolean esgrupo = result.getBoolean("esgrupo");
        int annograb = result.getInt("annograb");
        Date fechaestreno = result.getDate("fechaestreno");
        String compania = result.getString("compania");

        System.out.println("codgrupo: " + codgrupo);
        System.out.println("nombre: " + nombre);
        System.out.println("localidad: " + localidad);
        System.out.println("estilo: " + estilo);
        System.out.println("esgrupo: " + esgrupo);
        System.out.println("annograb: " + annograb);
        System.out.println("fechaestreno: " + fechaestreno);
        System.out.println("compania: " + compania);

        String sqlUpdate = "";

        Scanner teclado = new Scanner(System.in);
        int select;

        System.out.println("|-----MENU-----|");
        System.out.println("| 1.Modificar codgrupo |");
        System.out.println("| 2.Modificar nombre  |");
        System.out.println("| 3.Modificar localidad|");
        System.out.println("| 4.Modificar estilo   |");
        System.out.println("| 5.Modificar esgrupo  |");
        System.out.println("| 6.Modificar annograb |");
        System.out.println("| 7.Modificar fechaestren|");
        System.out.println("| 8.Modificar compania |");
        System.out.println("|-----|");
        System.out.println("| Selecciona opcion:   |");
        select = teclado.nextInt();
        System.out.println("|-----|");
    }
}
```

```

switch (select)
{
    case 1:
        int nuevoCodgrupo = Teclado.introInt("nuevo codigo de grupo: ");
        sqlUpdate = "UPDATE grupos SET codgrupo = " + nuevoCodgrupo + " "
                    + "WHERE codgrupo = " + codgrupo + ";";
        break;
    case 2:
        String nuevoNombre = Teclado.introString("nuevo nombre del grupo");
        sqlUpdate = "UPDATE grupos SET nombre = '" + nuevoNombre + "' "
                    + "WHERE nombre = '" + nombre + "';";
        break;
    case 3:
        String nuevaLocalidad = Teclado.introString("nueva localidad del grupo:");
        sqlUpdate = "UPDATE grupos SET localidad = '" + nuevaLocalidad + "' "
                    + "WHERE localidad = '" + localidad + "';";
        break;
    case 4:
        String nuevoEstilo = Teclado.introString("nuevo estilo del grupo:");
        sqlUpdate = "UPDATE grupos SET estilo = '" + nuevoEstilo + "' "
                    + "WHERE estilo = '" + estilo + "';";
        break;
    case 5:
        boolean nuevoEsgrupo = Teclado.introBoolean("Es grupo?");
        sqlUpdate = "UPDATE grupos SET esgrupo = " + nuevoEsgrupo + " "
                    + "WHERE esgrupo = " + esgrupo + ";";
        break;
    case 6:
        int nuevoAnnograb = Teclado.introInt("nuevo annograb del grupo: ");
        sqlUpdate = "UPDATE grupos SET annograb = " + nuevoAnnograb + " "
                    + "WHERE annograb = " + annograb + ";";
        break;
    case 7:
        String nuevaFechaestreno = Teclado.introString("nueva fechaestreno del grupo:");
        sqlUpdate = "UPDATE grupos SET fechaestreno = '" + nuevaFechaestreno + "' "
                    + "WHERE fechaestreno = '" + fechaestreno + "';";
        break;
    case 8:
        String nuevaCompania = Teclado.introString("nueva compania del grupo:");
        sqlUpdate = "UPDATE grupos SET compania = '" + nuevaCompania + "' "
                    + "WHERE compania = '" + compania + "';";
        break;

    default:
        System.out.println("Opcion no valida");
}

Statement stUpdate = conexion.createStatement();
int resultUpdate = stUpdate.executeUpdate(sqlUpdate);
//Result es el numero de filas afectadas por Update

System.out.println("Se modificó " + resultUpdate + " datos");
} catch (SQLException ex)
{
    System.out.println(ex.toString());
}
}
}

```

```
public class Main
{
    public static void main(String[] args)
    {
        GestorDB gestor = new GestorDB();
        List<Grupo> grupos;
        List<Cancion> canciones;
        List<Voto> votos;

        Scanner teclado = new Scanner(System.in);
        int select;
        do
        {
            System.out.println("-----MENU-----");
            System.out.println(" 1.Listado de grupos");
            System.out.println(" 2.Listado de canciones");
            System.out.println(" 3.Número de canciones por grupo");
            System.out.println(" 4.Canciones de un grupo");
            System.out.println(" 5.Las 5 canciones más votadas");
            System.out.println(" 6.Grupos sin canciones");
            System.out.println(" 7.Los últimos 5 votos");
            System.out.println(" 8.Eliminar canciones de un grupo");
            System.out.println(" 9.Modificar datos de grupo");
            System.out.println(" 0.Salir");
            System.out.println("-----");
            System.out.println(" Selecciona opcion:");
            select = teclado.nextInt();
            System.out.println("-----");
            switch (select)
            {
                case 1:
                    grupos = gestor.listadoGrupos();
                    for (Grupo grupo : grupos)
                    {
                        System.out.println(grupo.getCodgrupo() + " - "
                            + grupo.getNombre() + " - " + grupo.getLocalidad()
                            + " - " + grupo.getEstilo());
                    }
                    break;
            }
        } while (select != 0);
    }
}
```

```

case 2:
    grupos = gestor.listadoGruposCanciones();
    for (Grupo grupo : grupos)
    {
        System.out.println(grupo.getNombre());
        for (Cancion cancion : grupo.getCanciones())
        {
            System.out.println("\t" + cancion.getTitulo());
        }
    }
    break;
case 3:
    grupos = gestor.listadoGruposCanciones();
    for (Grupo grupo : grupos)
    {
        System.out.println(grupo.getNombre()
                            + " tiene " + grupo.getCanciones().size() + " canciones");
    }
    break;

case 4:
    String nombreGrupo = Teclado.introString("Grupo buscado: ");
    canciones = gestor.cancionesUnGrupo(nombreGrupo);
    System.out.println("Las canciones de " + nombreGrupo + " son:");
    for(Cancion cancion: canciones){
        System.out.println("Id: " + cancion.getNumcancion() +
                           " Titulo: " + cancion.getTitulo() +
                           " Duracion: " + cancion.getDuracion());
    }
    break;
case 5:
    canciones = gestor.cancionesMasVotadas();
    for(Cancion cancion: canciones){
        System.out.println("Votos: " + cancion.getTotal_votos() +
                           " Titulo: " + cancion.getTitulo() +
                           " Grupo: " + cancion.getGrupoObjeto().getNombre());
    }
    break;
case 6:
    grupos = gestor.listadoGruposCanciones();
    for(Grupo grupo: grupos){
        if(grupo.getCanciones().size() == 0){
            System.out.println(grupo.getNombre() + " no tiene cancion");
        }
    }
    break;
case 7:
    votos = gestor.votosMasRecientes();
    for(Voto voto: votos){ //Voto[Cancion(Grupo)]
        System.out.println("Titulo: " + voto.getCancion().getTitulo() +
                           " Grupo: " + voto.getCancion().getGrupoObjeto().getNombre() +
                           " Fecha: " + voto.getFecha());
    }
    break;

```

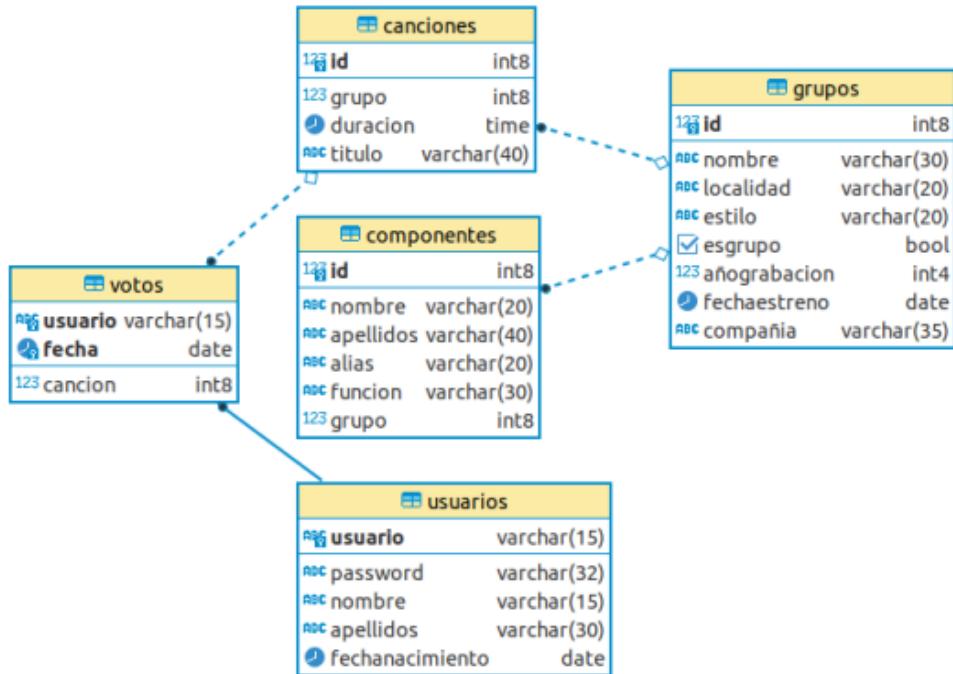
```
        case 8:
            String nombreGrupoBorrar = Teclado.introString("Nombre del grupo:");
            int cancionesBorradas = gestor.eliminarCancionesGrupo(nombreGrupoBorrar);
            System.out.println("Se eliminaron " + cancionesBorradas + " canciones");
            break;
        case 9:
            String nombreGrupoModificar = Teclado.introString("Nombre del Grupo:");
            gestor.modificarGrupo(nombreGrupoModificar);
            break;
        case 0:
            break;
        default:
            System.out.println("Opcion no valida");
    }
} while (select != 0);
}
▶}
```

DB SQL Hoja 5

En los ejercicios de esta actividad tienes que usar la base de datos PostgreSQL **concursemusicacompleto**.

Los ejercicios de esta tarea deben realizarse con métodos de actualización a través de los objetos leídos en un *ResultSet*.

El esquema relacional correspondiente a esta base de datos era:



EJERCICIO 1

Realiza un programa que pide por teclado el id de un grupo y a continuación va mostrando en pantalla los datos de los componentes del grupo.

De cada componente muestra sus datos y pregunta si se modificar su función. Si se responde que si, se pide un nuevo dato y se modifica.

Lo más adecuado es usar un *ResultSet* editable.

```

public class Componente {
    private BigDecimal idcomp;
    private String nombre;
    private String apellido;
    private String alias;
    private String funcion;
    private Grupo grupo;

    public Componente()
    {}

    public BigDecimal getIdcomp()
    {
        return idcomp;
    }

    public void setIdcomp(BigDecimal idcomp)
    {
        this.idcomp = idcomp;
    }

    public String getNombre()
    {
        return nombre;
    }

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public String getApellido()
    {
        return apellido;
    }

    public void setApellido(String apellido)
    {
        this.apellido = apellido;
    }
}

public String getAlias()
{
    return alias;
}

public void setAlias(String alias)
{
    this.alias = alias;
}

public String getFuncion()
{
    return funcion;
}

public void setFuncion(String funcion)
{
    this.funcion = funcion;
}

public Grupo getGrupo()
{
    return grupo;
}

public void setGrupo(Grupo grupo)
{
    this.grupo = grupo;
}

```

La clase grupo es la misma que el ejercicio anterior con una lista de componentes

```
public class GestorDB
{
    private Connection conexion;

    public GestorDB()
    {
        try
        {
            Class.forName("org.postgresql.Driver");
            conexion = DriverManager.getConnection(
                "jdbc:postgresql://127.0.0.1/concursomusicacompleto",
                "postgres", "PassWd!10");
        } catch (ClassNotFoundException ex)
        {
            System.out.println(ex.toString());
        } catch (SQLException ex)
        {
            System.out.println(ex.toString());
        }
    }

    public List<Componente> listadoComponentes(int idGrupo)
    {
        List<Componente> componentes = new ArrayList();
        String sql = "SELECT * FROM componentes WHERE grupo = '" + idGrupo + "' ORDER BY idcomp;";

        try
        {
            Statement st = conexion.createStatement();
            ResultSet result = st.executeQuery(sql);

            while (result.next())
            {
                Componente componente = new Componente();

                componente.setIdcomp(result.getBigDecimal("idcomp"));
                componente.setNombre(result.getString("nombre"));
                componente.setApellido(result.getString("apellido"));
                componente.setAlias(result.getString("alias"));

                Grupo grupo = new Grupo();
                grupo.setCodgrupo(BigDecimal.valueOf(result.getInt("grupo")));
                componente.setGrupo(grupo);

                componentes.add(componente);
            }
        } catch (SQLException ex)
        {
            System.out.println(ex.toString());
        }
        return componentes;
    }
}
```

```
public void ModificarAliasComponentes(int idGrupo)
{
    String sql = "SELECT * FROM componentes WHERE grupo = '" + idGrupo + "'";

    try
    {
        Statement st = conexion.createStatement(
            ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
        ResultSet result = st.executeQuery(sql);

        while (result.next())
        {
            System.out.println(result.getBigDecimal("idcomp") + " - "
                + result.getString("nombre") + " - "
                + result.getString("apellido") + " - "
                + result.getString("alias") + " - "
                + result.getInt("grupo"));

            boolean modificar = Teclado.introBoolean("Modificar Alias?");

            if (modificar == true)
            {
                String nuevoAlias = Teclado.introString("Nuevo Alias:");
                result.updateString("alias", nuevoAlias);
                result.updateRow();
            }
        }
    } catch (SQLException ex)
    {
        System.out.println(ex.toString());
    }
}
```

```
public class Main
{
    public static void main(String[] args)
    {
        GestorDB gestor = new GestorDB();

        List<Componente> componentes;

        int codigo = Teclado.introInt("Codigo del grupo: ");

        componentes = gestor.listadoComponentes(codigo);

        for (Componente componente : componentes)
        {
            System.out.println(componente.getIdcomp() + " - "
                + componente.getNombre() + " - "
                + componente.getApellido() + " - "
                + componente.getAlias() + " - "
                + componente.getGrupo().getCodgrupo());
        }

        System.out.println("-----");

        gestor.ModificarAliasComponentes(codigo);

        System.out.println("-----");

        componentes = gestor.listadoComponentes(codigo);

        for (Componente componente : componentes)
        {
            System.out.println(componente.getIdcomp() + " - "
                + componente.getNombre() + " - "
                + componente.getApellido() + " - "
                + componente.getAlias() + " - "
                + componente.getGrupo().getCodgrupo());
        }
    }
}
```

BD SQL Hoja 6

Antes de nada debemos crear un contenedor docker con **MariaDB**. Para ello podemos usar el siguiente docker-compose.yml

```
version: '3'

services:
  mariadb:
    image: mariadb
    container_name: mariadb
    volumes:
      - mariadb_data_container:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: root
    ports:
      - "3310:3306"
    networks:
      - mariadb-network

networks:
  mariadb-network:

volumes:
  mariadb_data_container:
```

En los ejercicios de esta actividad tienes que usar la base de datos MariaDB **preguntastest**. Para ello crearemos la base de datos con ese nombre desde Dbeaver, por ejemplo.

Luego descargaremos el script preguntastest.sql y ejecutaremos este comando desde la carpeta donde lo tengamos:

```
docker exec -i mariadb mysql -uroot -proot preguntastest < preguntastest.sql
```

```

usuario@usuario-ad:~$ mkdir MariaDB
usuario@usuario-ad:~$ sudo nano docker-compose.yml
[sudo] contraseña para usuario:
usuario@usuario-ad:~$ cd MariaDB
usuario@usuario-ad:~/MariaDB$ sudo nano docker-compose.yml
[sudo] contraseña para usuario:
usuario@usuario-ad:~/MariaDB$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
              NAMES
d59f1021eab3   postgres:latest "docker-entrypoint.s..."  4 weeks ago   Up 7 minutes   0.0.0.0:5432->5432
/tcp, :::5432->5432/tcp      postgresql
a3f376450193   mongo:latest   "docker-entrypoint.s..."  8 weeks ago   Up 7 minutes   0.0.0.0:27017->270
17/tcp, :::27017->27017/tcp    mongo-server
usuario@usuario-ad:~/MariaDB$ docker-compose up -d
Creating network "mariadb_mariadb-network" with the default driver
Creating volume "mariadb_mariadb_data_container" with default driver
Pulling mariadb (mariadb:)... 
latest: Pulling from library/mariadb
ea362f368469: Pull complete
adb9a1b1379d: Pull complete
ac5c95406850: Pull complete
fa48d8b47ec1: Pull complete
bcf1feb44ac3: Pull complete
8a5de7784a0f: Pull complete
b8724b8a281a: Pull complete
a8a7c3f612d6: Pull complete
39b09b59e889: Pull complete
14bc3a6b0a94: Pull complete
Digest: sha256:5a37e65a6414d78f60d523c4ddcf93d715854337beb46f8beeb1a23d83262184
Status: Downloaded newer image for mariadb:latest
Creating mariadb ... done
usuario@usuario-ad:~/MariaDB$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
              NAMES
c9699f2df0f5   mariadb       "docker-entrypoint.s..."  16 seconds ago Up 14 seconds  0.0.0.0:3310->
3306/tcp, :::3310->3306/tcp      mariadb
d59f1021eab3   postgres:latest "docker-entrypoint.s..."  4 weeks ago   Up 9 minutes   0.0.0.0:5432->
5432/tcp, :::5432->5432/tcp      postgresql
a3f376450193   mongo:latest   "docker-entrypoint.s..."  8 weeks ago   Up 9 minutes   0.0.0.0:27017-
>27017/tcp, :::27017->27017/tcp    mongo-server
usuario@usuario-ad:~/MariaDB$ 

```

Crear DB como la anterior pero con MariaDB y cambiando los parámetros según el compose del ejercicio -> Puerto 3310 usuario root contraseña root.

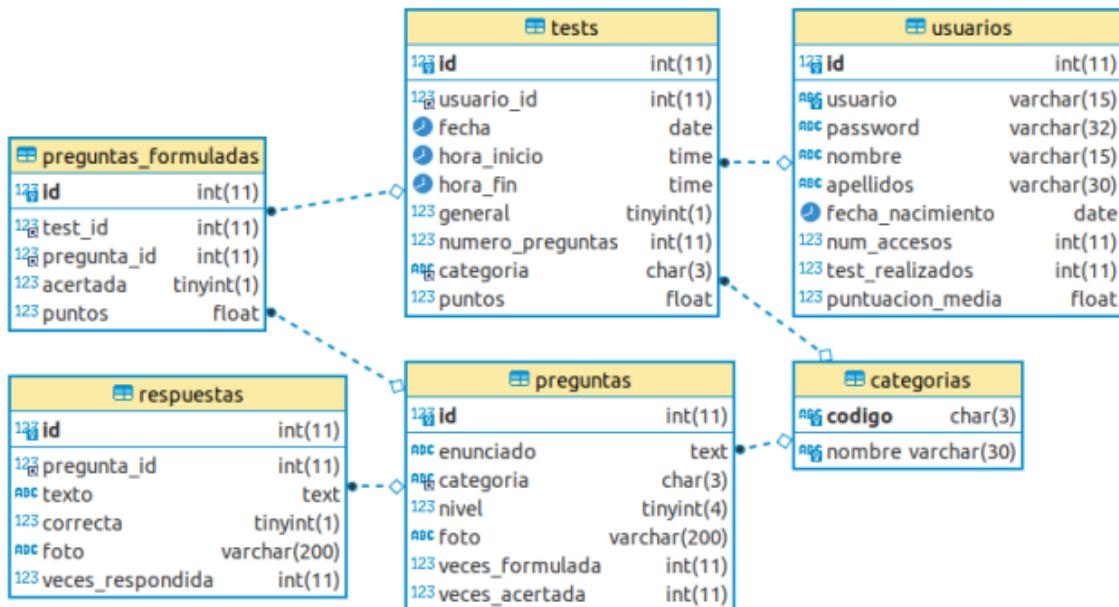
Crear la database preguntastest e importar el script de la base.

```

Mint-AD [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
usuario@usuario-ad:~/Escritorio/Ejercicios_AD/Tema_4/Bases_de_Datos
Archivo Editar Ver Buscar Terminal Ayuda
usuario@usuario-ad:~/Escritorio/Ejercicios_AD/Tema_4/Bases_de_Datos$ docker exec -i mariadb mysql -uroot -proot < preguntastest.sql

```

Una vez importada la base de datos podemos observar su esquema relacional:



La tabla **categorías** tiene información sobre los temas posibles de las preguntas: ART-Arte y cultura, CIE-Ciencias, DEP-Deportes, ECO-Economía, FIL-Filosofía, etc.

La tabla **preguntas** contiene datos sobre una batería de preguntas de test.

- El id es autoincrementado por lo que nunca se le debe asignar un valor.
- El nivel es 1, 2 o 3 (de momento todas las preguntas tendrán nivel 1, así que ese campo no lo usaremos)
- La columna **ruta_foto** contiene la ruta de un posible archivo de imagen que se usaría en la pregunta.

La tabla **usuarios** contiene información de los usuarios que se registran para realizar test. La columna **id** es autoincrementada.

La tabla **respuestas** contiene las respuestas que se mostrarán en el test para cada pregunta. En la columna **correcta** se almacena un uno cuando es una respuesta correcta y un cero cuando no lo es.

```
public class Usuario {
    //ATRIBUTOS-----
    private int id;
    private String usuario;
    private String password;
    private String nombre;
    private String apellidos;
    private Date fecha_nacimiento;
    private int num_accesos;
    private int test_realizados;
    private float puntuacion_media;
    //CONSTRUCTOR-----
    public Usuario()
    {
    }
    //GETTER_&_SETTER-----
    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getUsuario()
    {
        return usuario;
    }

    public void setUsuario(String usuario)
    {
        this.usuario = usuario;
    }

    public String getPassword()
    {
        return password;
    }

    public void setPassword(String password)
    {
        this.password = password;
    }
}
```

```
public class Pregunta
{
    //ATRIBUTOS-----
    private int id;
    private String enunciado;
    private String categoria;
    private int nivel;
    private String foto;
    private int veces_formulada;
    private int veces_acertada;
    private List<Respuesta> respuestas;
    //CONSTRUCTOR-----
    public Pregunta()
    {
    }
    //GETTER_&_SETTER-----
    public List<Respuesta> getRespuestas()
    {
        return respuestas;
    }

    public void setRespuestas(List<Respuesta> respuestas)
    {
        this.respuestas = respuestas;
    }

    public void addRespuesta(Respuesta respuesta)
    {
        this.respuestas.add(respuesta);
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getEnunciado()
    {
    }
}
```

```
public class Respuesta {
    //ATRIBUTOS-----
    private int id;
    private Pregunta pregunta;
    private String texto;
    private boolean correcta;
    private String foto;
    private int veces_respondida;
    //CONSTRUCTOR-----
    public Respuesta()
    {
    }
    //GETTER_&_SETTER-----
    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public Pregunta getPregunta()
    {
        return pregunta;
    }

    public void setPregunta(Pregunta pregunta)
    {
        this.pregunta = pregunta;
    }

    public String getTexto()
    {
        return texto;
    }

    public void setTexto(String texto)
    {
        this.texto = texto;
    }
}
```

```
public class Categoria {
    //ATRIBUTOS-----
    private int codigo;
    private String nombre;
    //CONSTRUCTOR-----
    public Categoria()
    {
    }
    //GETTER_&_SETTER-----
    public int getCodigo()
    {
        return codigo;
    }

    public void setCodigo(int codigo)
    {
        this.codigo = codigo;
    }

    public String getNombre()
    {
        return nombre;
    }

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }
}
```

EJERCICIO 1

Realizar un programa para registrarte como usuario en **preguntastest**. Sólo debe pedir el nombre, los apellidos, la fecha de nacimiento, el usuario de acceso y la contraseña.

Debes usar una clase **Conexion** singleton.

Para gestionar los datos de un usuario, debes desarrollar una clase POJO **Usuario** con los datos miembro correspondientes a las columnas de la tabla **usuarios**.

La contraseña hay que guardarla con la función **MD5** de MySQL.

```
public class Conexion {  
    //ATRIBUTOS-----  
    private static Conexion instance;  
    private Connection connection;  
    private String url = "jdbc:mariadb://localhost:3310/preguntastest?user=root&password=root";  
    private String username = "root";  
    private String password = "root";  
    //CONSTRUCTOR SINGLETON-----  
    private Conexion() {  
        try {  
            Class.forName("org.mariadb.jdbc.Driver");  
            this.connection = DriverManager.getConnection(url, username, password);  
        } catch (ClassNotFoundException ex) {  
            System.out.println(ex.toString());  
        } catch (SQLException ex) {  
            System.out.println(ex.toString());  
        }  
    }  
  
    public Connection getConnection() {  
        return connection;  
    }  
  
    public static Conexion getInstance() {  
        try {  
            if (instance == null) {  
                instance = new Conexion();  
            } else if (instance.getConnection().isClosed()) {  
                instance = new Conexion();  
            }  
        } catch (SQLException ex) {  
            System.out.println(ex.toString());  
        }  
        return instance;  
    }  
}
```

```

public class Main {
    public static void main(String[] args) {
        GestorDB gestor = new GestorDB();
        GestorFicheros gestorFicheros = new GestorFicheros();
        int opcion = 0;

        do {
            Scanner teclado = new Scanner(System.in);
            System.out.println("");
            System.out.println("[=====] ");
            System.out.println("[" + " QUE COMIENCE EL JUEGO " + "]");
            System.out.println("[=====] ");
            System.out.println("[" + " ¿Que quieres hacer " + "]");
            System.out.println("[" + " ]");
            System.out.println("[" + " 1 - Añadir un nuevo usuario " + "]");
            System.out.println("[" + " 2 - Cargar fichero " + "]");
            System.out.println("[" + " 3 - Ver preguntas por categoría " + "]");
            System.out.println("[" + " 4 - Jugar " + "]");
            System.out.println("[" + " 5 - Salir " + "]");
            System.out.println("[=====] ");
            opcion = teclado.nextInt();
            teclado.nextLine();
            System.out.println("");
            switch (opcion) {
                case 1: //Ejercicio_1: Registrar un usuario nuevo en la base de datos
                    boolean respuesta;
                    Usuario usuario = pedirDatos();
                    respuesta = gestor.registrarUsuario(usuario);

                    System.out.println("");
                    if(respuesta == true) System.out.println("Usuario registrado correctamente");
                    else System.out.println("Error al registrar usuario");
                    System.out.println("");
                    break;
                =====
                private static Usuario pedirDatos() {
                    String nombre = Teclado.introString("NOMBRE:");
                    String apellidos = Teclado.introString("APELLIDOS:");
                    Date fecha_nacimiento = Date.valueOf(Teclado.introFecha("FECHA DE NACIMIENTO:"));
                    String usuario = Teclado.introString("USUARIO:");
                    String password = Teclado.introString("PASSWORD:");

                    Usuario user = new Usuario();

                    user.setNombre(nombre);
                    user.setApellidos(apellidos);
                    user.setFecha_nacimiento(fecha_nacimiento);
                    user.setUsuario(usuario);
                    user.setPassword(password);

                    return user;
                }
            }
        }
    }
}

```

```
public class GestorDB {  
    //Ejercicio_1: Registrar un usuario nuevo en la base de datos-----  
    public boolean registrarUsusario (Usuario usuario) {  
        boolean respuesta = false;  
  
        Connection conexion = Conexion.getInstance().getConnection();  
        String sql = "INSERT INTO usuarios (nombre, apellidos, fecha_nacimiento, usuario, password)"  
            + " VALUES (?,?,?,?,md5(?));"  
  
        try( PreparedStatement consulta = conexion.prepareStatement(sql) ){  
            consulta.setString(1, usuario.getNombre());  
            consulta.setString(2, usuario.getApellidos());  
            consulta.setDate(3, usuario.getFecha_nacimiento());  
            consulta.setString(4, usuario.getUsuario());  
            consulta.setString(5, usuario.getPassword());  
  
            int resul = consulta.executeUpdate();  
            if(resul != 0) respuesta = true;  
            //El metodo retorna true si se registra correctamente o false si no se registra.  
        } catch (SQLException ex) {  
            System.out.println(ex.toString());  
        }  
        return respuesta;  
    }  
}
```

EJERCICIO 2 (en el mismo proyecto que el 1)

Antes de desarrollar el programa crear un fichero CSV con datos de preguntas y respuestas. Los datos de cada pregunta están separados por el carácter punto y coma (;). Las preguntas se identificarán con el carácter P y las respuestas con la R.

De cada pregunta se indicará el enunciado, la categoría y el nivel.

De cada respuesta se indicará el texto y si es correcta o no.

Un ejemplo de una pregunta es el siguiente:

```
P;¿Cuál era el nombre original de la ciudad de Nueva York?;HIS;1
R;Nueva Ámsterdam;1
R;La gran manzana;0
R;Empire State;0
R;Gotham;0
P;¿En qué país se celebraron los primeros Juegos Olímpicos?;DEP;1
R;Italia;0
R;Grecia;1
R;Japón;0
R;Francia;0
```

Se debe crear una clase Pregunta con los atributos necesarios. Además, cada pregunta tendrá una colección de objetos de tipo Respuesta.

Se puede crear una paquete util que contenga una clase GestorFicheros que contenga un método leerPreguntasCSV que devuelva una colección de preguntas.

Después, esas preguntas y respuestas serán cargadas en la base de datos.

En la clase Principal haz un menú con las dos opciones que tenemos hasta ahora:

1. Insertar usuario
2. Cargar fichero CSV

En esta opción 2 deberemos solicitar la ruta del fichero CSV a cargar

```
public class GestorFicheros {
    public List<Pregunta> leerPreguntasCSV (File fichero) {
        List<Pregunta> preguntas = new ArrayList();
        BufferedReader br = null;

        try{
            br = new BufferedReader(new FileReader(fichero));
            String line = br.readLine();

            while(line != null){
                String[] fields = line.split(";");
                //Cadena de String's separados por ";"

                if(fields[0].equalsIgnoreCase("P")){ //Es una pregunta
                    Pregunta p = new Pregunta();

                    p.setEnunciado(fields[1]);
                    p.setCategoria(fields[2]);
                    p.setNivel(Integer.parseInt(fields[3]));

                    preguntas.add(p);
                }else if(fields[0].equalsIgnoreCase("R")) { //Es una respuesta
                    Respuesta r = new Respuesta();

                    r.setTexto(fields[1]);
                    if(fields[2].equalsIgnoreCase("1")) r.setCorrecta(true);
                    else r.setCorrecta(false);
                    r.setPregunta(preguntas.get(preguntas.size()-1));

                    preguntas.get(preguntas.size()-1).addRespuesta(r);
                }
                line = br.readLine();
            }
        } catch (FileNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (IOException ex) {
            System.out.println(ex.toString());
        }
        return preguntas;
    }
}
```

```

public class GestorDB {

    ...
    //Ejercicio_2: Anadir una lista de preguntas leida de un CSV a la base de datos-----
    public int aniadirPreguntas(List<Pregunta> preguntas) {
        int contadorCorrectos = 0;

        Connection conexion = Conexion.getInstance().getConnection();
        String sql = "INSERT INTO preguntas (enunciado, categoria, nivel) VALUES (?,?,?)";

        for(Pregunta pregunta: preguntas){
            try(PreparedStatement consulta = conexion.prepareStatement(sql)){
                consulta.setString(1, pregunta.getEnunciado());
                consulta.setString(2, pregunta.getCategoría());
                consulta.setInt(3, pregunta.getNivel());

                int resul = consulta.executeUpdate();
                contadorCorrectos = contadorCorrectos + resul;
            } catch (SQLException ex) {
                System.out.println(ex.toString());
            }
        }
        return contadorCorrectos;
    }

    public class Main {
        public static void main(String[] args) {

            ...
            case 2://Ejercicio_2: Add una lista de preguntas leida de un CSV a la base de datos
            File fichero = new File("fichero.csv");
            List<Pregunta> preguntas = gestorFicheros.leerPreguntasCSV(fichero);

            int preguntasAniadidas = gestor.aniadirPreguntas(preguntas);
            System.out.println(preguntasAniadidas + " preguntas añadidas correctamente");
            break;
        }
    }
}

```

EJERCICIO 3 (en el mismo proyecto que los anteriores)

Realizar un programa que muestra el texto de 5 preguntas de la categoría indicada por el usuario (el código) y, a continuación de cada pregunta, los textos de cada respuesta.

Si la categoría indicada tiene menos de 5 preguntas las mostrará todas. Si la categoría indicada tiene más de 5 preguntas elegirá aleatoriamente 5.

Por cada pregunta mostrada habrá que pulsar ENTER para pasar a ver la siguiente.

Añadir al menú la opción 3. Visualizar preguntas de una categoría

```

public class GestorDB {

    ...

    //Ejercicio_3: Visualizar preguntas de cada categoria-----
    public void preguntasPorCategorias(String categoria) {
        Connection conexion = Conexion.getInstance().getConnection();
        String sqlCategoria = "SELECT id, enunciado FROM preguntas WHERE categoria=?";
        String sqlCategoriaAleatoria
            = "SELECT id, enunciado FROM preguntas WHERE categoria=? ORDER BY RAND() LIMIT 5";
        String sqlRespuesta = "SELECT texto FROM respuestas WHERE pregunta_id=?";

        try ( PreparedStatement consultaCategoria = conexion.prepareStatement(sqlCategoria)) {
            consultaCategoria.setString(1, categoria);
            ResultSet result = consultaCategoria.executeQuery();

            result.last(); //Obtenemos el ultimo registro
            int cantidadRegistros = result.getRow(); //y comprobamos la cantidad de preguntas.
            result.beforeFirst(); //Volvemos a la primera posicion.

            if (cantidadRegistros > 5) {
                PreparedStatement consultaCategoriaAleatoria =
                    conexion.prepareStatement(sqlCategoriaAleatoria);
                consultaCategoriaAleatoria.setString(1, categoria);
                result = consultaCategoriaAleatoria.executeQuery();
            }
            while (result.next()) {
                int id = result.getInt("id");
                String enunciado = result.getString("enunciado");

                System.out.printf("%15s %-15s\n", id, enunciado);
                //Recorremos un nuevo resultSet para obtener las respuestas a partir de una pregunta
                PreparedStatement consultaRespuestas = conexion.prepareStatement(sqlRespuesta);
                consultaRespuestas.setInt(1, id);
                ResultSet resultRespuesta = consultaRespuestas.executeQuery();

                while (resultRespuesta.next()) {
                    String texto = resultRespuesta.getString("texto");
                    System.out.println(texto);
                }

                System.out.println("[ ===== ]");
                System.out.println("");
            }
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
    }
}

public class Main {
    public static void main(String[] args) {

    ...

        case 3:
            String categoria = Teclado.introString("Categoria para mostrar preguntas: ");
            gestor.preguntasPorCategorias(categoria);
            break;
    }
}

```

Mejor con Objetos.

```
//Ejercicio_3: Visualizar preguntas de cada categoria-----
public List<Pregunta> preguntasPorCategorias2(String categoria) {
    Connection conexion = Conexion.getInstance().getConnection();
    String sqlCategoria = "SELECT * FROM preguntas WHERE categoria=?";
    String sqlCategoriaAleatoria
        = "SELECT * FROM preguntas WHERE categoria=? ORDER BY RAND() LIMIT 5";
    String sqlRespuesta = "SELECT * FROM respuestas WHERE pregunta_id=?";

    List<Pregunta> preguntas = new ArrayList();

    try ( PreparedStatement consultaCategoria = conexion.prepareStatement(sqlCategoria)) {
        consultaCategoria.setString(1, categoria);
        ResultSet result = consultaCategoria.executeQuery();

        result.last(); //Obtenemos el ultimo registro
        int cantidadRegistros = result.getRow(); //y comprobamos la cantidad de preguntas.
        result.beforeFirst(); //Volvemos a la primera posicion.

        if (cantidadRegistros > 5) {
            PreparedStatement consultaCategoriaAleatoria =
                conexion.prepareStatement(sqlCategoriaAleatoria);
            consultaCategoriaAleatoria.setString(1, categoria);
            result = consultaCategoriaAleatoria.executeQuery();
        }
        while (result.next()) {
            Pregunta pregunta = new Pregunta();

            pregunta.setId(result.getInt("id"));
            pregunta.setEnunciado(result.getString("enunciado"));
            pregunta.setCategoria(result.getString("categoria"));
            pregunta.setNivel(result.getInt("nivel"));
            pregunta.setFoto(result.getString("foto"));
            pregunta.setVeces_formulada(result.getInt("veces_formulada"));
            pregunta.setVeces_acertada(result.getInt("veces_acertada"));

            //Recorremos un nuevo resultset para obtener las respuestas a partir de una pregunta
            PreparedStatement consultaRespuestas = conexion.prepareStatement(sqlRespuesta);
            consultaRespuestas.setInt(1, pregunta.getId());
            ResultSet resultRespuesta = consultaRespuestas.executeQuery();
        }
    }
}
```

```

        while (resultRespuesta.next()) {
            Respuesta respuesta = new Respuesta();

            respuesta.setId(resultRespuesta.getInt("id"));
            respuesta.setPregunta(pregunta);
            respuesta.setTexto(resultRespuesta.getString("texto"));
            respuesta.setCorrecta(resultRespuesta.getBoolean("correcta"));
            respuesta.setFoto(resultRespuesta.getString("foto"));
            respuesta.setVeces_respondida(resultRespuesta.getInt("veces_respondida"));

            pregunta.addRespuesta(respuesta);
        }

        preguntas.add(pregunta);
    }
} catch (SQLException ex) {
    System.out.println(ex.toString());
}
return preguntas;
}

case 3:
String categoria = Teclado.introString("Categoria para mostrar preguntas: ");
// gestor.preguntasPorCategorias(categoria);

List<Pregunta> preguntasCategoria = new ArrayList();
preguntasCategoria = gestor.preguntasPorCategorias2(categoria);

for(Pregunta p: preguntasCategoria){
    System.out.println(p.getEnunciado());
    System.out.println("");

    for(Respuesta r: p.getRespuestas()){
        System.out.println(r.getTexto());
    }
    System.out.println("");
}
break;

```

EJERCICIO 4 (en el mismo proyecto que los anteriores)

Realiza un programa que pide un nombre usuario y contraseña para acceder a la base de datos **preguntastest**. Hay que tener en cuenta que la contraseña está guardada con MD5.

Si las credenciales son correctas, se muestra un mensaje indicando que se ha iniciado una sesión de usuario, el nombre y apellidos del usuario y:

- Se contabiliza un acceso más del usuario.
- Se formulan al usuario 4 preguntas aleatorias. Por cada pregunta se escriben sus respuestas.
- Por cada pregunta se pide al usuario que responda cuál es la correcta y se le responde si ha acertado o no.
- Por cada pregunta, se registra que ha sido formulada una vez más y, en su caso, que fue acertada la respuesta.
- Para la respuesta respondida se registra que ha sido respondida una vez más.
- Se contabiliza que el usuario ha realizado un test más.

Añadir al menú la opción 4. Jugar

```

public class GestorDB {

    ...

    //Ejercicio_4: Jugar-----
    public void jugar(String nombreUsuario, String password) {
        Connection conexion = Conexion.getInstance().getConnection();

        String sqlAcceso = "SELECT * FROM usuarios WHERE usuario=? AND password=md5(?);"

        try ( PreparedStatement consultaAcceso = conexion.prepareStatement(sqlAcceso)) {
            consultaAcceso.setString(1, nombreUsuario);
            consultaAcceso.setString(2, password);

            ResultSet resultAcceso = consultaAcceso.executeQuery();
            if (resultAcceso.next()) {
                String usuario = resultAcceso.getString("usuario");
                String nombre = resultAcceso.getString("nombre");
                String apellidos = resultAcceso.getString("apellidos");
                int num_accesos = resultAcceso.getInt("num_accesos");
                int test_realizados = resultAcceso.getInt("test_realizados");

                System.out.print("Se ha iniciado sesion con:");
                System.out.printf("%-10s Nombre: %-10s Apellidos: %-25s\n",
                    usuario, nombre, apellidos);
                System.out.println("");

                this.contabilizarAcceso((num_accesos + 1), usuario, conexion);
                this.aumentarTestRealizados((test_realizados + 1), usuario, conexion);
                this.preguntarUsuario(conexion);
            } else {
                System.out.println("ERROR AL INICIAR SESION");
            }
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
    }

    //Aumenta el numero de accesos de un usuario cada vez que inicia sesion
    private void contabilizarAcceso(int numeroAccesos, String usuario, Connection conexion) {
        try {
            String sql = "UPDATE usuarios SET num_accesos = ? WHERE usuario = ?";

            PreparedStatement update = conexion.prepareStatement(sql);
            update.setInt(1, numeroAccesos);
            update.setString(2, usuario);

            int numeroRegistrosModificados = update.executeUpdate();
            System.out.println("//contabilizarAcceso " + numeroRegistrosModificados +
                " registros modificados");
        } catch (SQLException ex) {
            System.out.println("ERROR contabilizarAcceso: " + ex.toString());
        }
    }
}

```

```

//Aumenta el numero de test realizados de un usuario cuando realiza un test
private void aumentarTestRealizados(int test_realizados, String usuario, Connection conexion) {
    try {
        String sql = "UPDATE usuarios SET test_realizados = ? WHERE usuario = ?";

        PreparedStatement update = conexion.prepareStatement(sql);
        update.setInt(1, test_realizados);
        update.setString(2, usuario);
        int numeroRegistrosModificados = update.executeUpdate();
        System.out.println("//aumentarTestRealizados " + numeroRegistrosModificados +
                           " registros modificados");
    } catch (SQLException ex) {
        System.out.println("ERROR aumentarTestRealizados: " + ex.toString());
    }
}

//Ofrece 4 preguntas aleatorias al usuario
private void preguntarUsuario(Connection conexion) {
    try {
        String sqlPreguntas = "SELECT * FROM preguntas ORDER BY RAND() LIMIT 4";

        PreparedStatement consultaPreguntas = conexion.prepareStatement(sqlPreguntas);
        ResultSet resultPreguntas = consultaPreguntas.executeQuery();
        System.out.println("\nQUE COMIENCE EL JUEGO !\n");
        while (resultPreguntas.next()) {
            Pregunta p = new Pregunta();
            p.setId(resultPreguntas.getInt("id"));
            p.setEnunciado(resultPreguntas.getString("enunciado"));
            p.setCategoria(resultPreguntas.getString("categoria"));
            p.setNivel(resultPreguntas.getInt("nivel"));
            p.setFoto(resultPreguntas.getString("foto"));
            p.setVeces_formulada(resultPreguntas.getInt("veces_formulada"));
            p.setVeces_acertada(resultPreguntas.getInt("veces_acertada"));

            System.out.printf("%-5s %-10s\n", p.getId(), p.getEnunciado());
            List<Respuesta> respuestas = ObtenerRespuestas(p, conexion);
            p.setRespuestas(respuestas);

            int numeradorRespuestas = 1;
            for(Respuesta respuesta: p.getRespuestas()){
                System.out.println(numeradorRespuestas + " - " + respuesta.getTexto());
                numeradorRespuestas++;
            }
            System.out.println("");
            //El usuario responde 1/2/3/4. Se comprueba si es correcto y se contabiliza.
            int respuestaUsuario = Teclado.introInt("Introduzca su respuesta: (1/2/3/4)");
            if(p.getRespuestas().get(respuestaUsuario - 1).isCorrecta()){
                System.out.println("Correcto!");
                this.aumentarPreguntaAcertada((p.getVeces_acertada() + 1), p.getId(), conexion);
            }else{
                System.out.println("Error!");
            }
            System.out.println("");
            this.aumentarRespuestaRespondida(
                p.getRespuestas().get(respuestaUsuario - 1).getVeces_respondida(),
                p.getRespuestas().get(respuestaUsuario - 1).getId(),
                conexion);
            this.aumentarPreguntaFormulada((p.getVeces_formulada() + 1), p.getId(), conexion);
            System.out.println("");
        }
    } catch (SQLException ex) {
        System.out.println("ERROR preguntarUsuario: " + ex.toString());
    }
}

```

```

//Saca las respuestas de una pregunta
private List<Respuesta> ObtenerRespuestas(Pregunta p, Connection conexion) {
    List<Respuesta> respuestas = new ArrayList();
    try {
        String sqlRespuestas = "SELECT * FROM respuestas WHERE pregunta_id = ?";
        PreparedStatement consultaRespuestas = conexion.prepareStatement(sqlRespuestas);
        consultaRespuestas.setInt(1, p.getId());
        ResultSet resultRespuestas = consultaRespuestas.executeQuery();

        while (resultRespuestas.next()) {
            Respuesta r = new Respuesta();
            r.setId(resultRespuestas.getInt("id"));
            r.setPregunta(p);
            r.setTexto(resultRespuestas.getString("texto"));
            r.setCorrecta(resultRespuestas.getBoolean("correcta"));
            r.setFoto(resultRespuestas.getString("foto"));
            r.setVeces_respondida(resultRespuestas.getInt("veces_respondida"));
            respuestas.add(r);
        }
    } catch (SQLException ex) {
        System.out.println("ERROR ObtenerRespuestas: " + ex.toString());
    }
    return respuestas;
}

//Aumenta el numero de veces que una pregunta es formulada
private void aumentarPreguntaFormulada(int veces_formulada, int id, Connection conexion){
    try {
        String sql = "UPDATE preguntas SET veces_formulada = ? WHERE id = ?";
        PreparedStatement update = conexion.prepareStatement(sql);
        update.setInt(1, veces_formulada);
        update.setInt(2, id);
        int numeroRegistrosModificados = update.executeUpdate();
        System.out.println("//aumentarPreguntaFormulada " + numeroRegistrosModificados +
                           " registros modificados");
    } catch (SQLException ex) {
        System.out.println("ERROR aumentarPreguntaFormulada: " + ex.toString());
    }
}

//Aumenta el numero de veces que una pregunta es acertada
private void aumentarPreguntaAcertada(int veces_acertada, int id, Connection conexion){
    try {
        String sql = "UPDATE preguntas SET veces_acertada = ? WHERE id = ?";
        PreparedStatement update = conexion.prepareStatement(sql);
        update.setInt(1, veces_acertada);
        update.setInt(2, id);
        int numeroRegistrosModificados = update.executeUpdate();
        System.out.println("//aumentarPreguntaAcertada " + numeroRegistrosModificados +
                           " registros modificados");
    } catch (SQLException ex) {
        System.out.println("ERROR aumentarPreguntaAcertada: " + ex.toString());
    }
}

```

```
//Aumenta el numero de veces que se responde una respuesta
public void aumentarRespuestaRespondida(int veces_respondida, int id, Connection conexion) {
    try {
        String sql = "UPDATE respuestas SET veces_respondida=? WHERE id=?";
        PreparedStatement update = conexion.prepareStatement(sql);
        update.setInt(1, veces_respondida + 1);
        update.setInt(2, id);
        int numeroRegistrosModificados = update.executeUpdate();
        System.out.println("//aumentarRespuestaRespondida " + numeroRegistrosModificados +
                           " registros modificados");
    } catch (SQLException ex) {
        System.out.println("ERROR aumentarRespuestaRespondida: " + ex.toString());
    }
}

=====
public class Main {
    public static void main(String[] args) {

    ...
    case 4:
        String introUsuario = Teclado.introString("USUARIO: ");
        String introPassword = Teclado.introString("PASSWORD: ");
        gestor.jugar(introUsuario, introPassword);
        break;
    }
}
```