**M06\_U2\_E1 (PostgreSQL):**

**BD Ejemplo:**

CREATE TYPE creador as

(

nick character varying(10),

nomUsuari character varying(40),

majorEdat boolean,

residencia character varying(25)

);

CREATE TABLE Canal

(

id integer NOT NULL,

nom character varying(30),

tema character varying(40),

numVideos integer,

propietari creador,

subscriptors character varying(40)[],

CONSTRAINT clauPrimariaCanal PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

ALTER TABLE Canal

OWNER TO ioc;

**Conexión**:

**//Variables de clase con los datos de la conexión**

private Connection **connexio**;

private final String URL="jdbc:postgresql://localhost:5432/eac2\_2021s2";

private final String USER="ioc";

private final String PSW="ioc";

**//Método para abrir la conexión**

private void obrir() throws SQLException{

**connexio** = DriverManager.getConnection(URL, USER, PSW);

**connexio**.setAutoCommit(true); //

gestor = new GestorCanal(connexio); //Crea una instancia de la clase GestorCanal

}

**//Método para cerrar la conexión**

private void tancar() throws SQLException{

connexio.close();

}

**Nota:** PostgreSQL permite utilizar lo que se denomina tipos (TYPE), que es como una tabla virtual que se enlaza con una tabla real como si fuera un campo. Esta tabla virtual contiene una serie de campos que pueden ser de diferentes tipos, tal cual como sería una tabla normal.

También PostgreSQL, nos permite definir campos de tipo array. Estos campos han de ser de un tipo u otro, es decir, VARCHAR, INT, BOOLEAN, etc.., todos han de ser del mismo tipo.

Los comandos son los básicos de tipo MySql (SELECT, INSERT, DELETE, etc…), con la particularidad de que los campos de TIPO (TYPE) se han de especificar entre paréntesis.

**Insertar:**

Para insertar datos se realiza en JAVA se utiliza los Objetos **PreparedStatement** (para el paso de parámetros) y **ResultSet** (para el caso de query’s)

Realimos un insert en un String en la clase de ejemplo:

***String consulta = "INSERT INTO canal VALUES(?,?,?,?,(?,?,?,?),?)";***🡪 Como podemos ver, los datos entre paréntesis hacen referencia al campo TYPE, en este caso, ese campo se llama “propietari”.

Ahora instanciamos el objeto **PreparedStatement** ayudándonos del objeto “conexio” pasándole como parámetro la consulta (String):

***PreparedStatement p = connexio.******prepareStatement(consulta);***

Ahora debemos especificar a que corresponde cada valor de la consulta (cada ‘?’):

***p.setInt(1, can.getId());*** 🡪 “can” es un objeto de tipo “Canal (clase Canal pasada por parámetro)”. Es el primer parámetro que corresponde al campo “id” que es de tipo “int”.

Los mismo hacemos con los siguientes campos (son un total de 6 (id, nom, tema, numVideos, propietari (es de tipo TYPE), subscriptors (es un array de VARCHARS)):

***p.setString(2, can.getNom());***

***p.setString(3, can.getTema());***

***p.setInt(4, can.getNumVideos());***

//propietari 🡪 Es el campo compuesto

***p.setString(5, can.getNick());***

***p.setString(6, can.getNomUsuari());***

***p.setBoolean(7, can.isMajorEdat());***

***p.setString(8, can.getResidencia());***

//Ahora los subscriptores que es un array de VARCHAR

***Array obj = conexio.createArrayOf("VARCHAR", can.getSubscriptors().toArray());***

***p.setArray(9, obj);***

Para el último campo (subscriptores), instanciamos un objeto “Array” al que denominamos “obj”, y con ayuda del objeto “conexio” y el método “***createArrayOf(“TIPO\_DATO”, can.getSubscriptors().toArray();***”, capturamos el campo completo como un array para que posteriormente, lo asignemos al último ‘?’ que es el número ‘9’ 🡪 ***p.setArray(9, obj);***

Por último, con el objeto “p” creado con “**PreparedStatement**”, ejecutamos la consulta:

***p.executeUpdate();***

**Obtención de datos (QUERY):**

El proceso es similar, salvo por algunos matices. La última sentencia, en vez de ser “executeUpdate()”, es “executeQuery()” que lo almacenará el objeto “ResultSet”. Vemos el proceso.

Si queremos obtener un objeto a través de una “query”, en nuestro caso es un objeto de tipo Canal debemos seguir el siguiente proceso:

Lo primero es instanciar los objetos que utilizaremos:

**Canal canal = null;** 🡪 Objeto de la clase Canal

**PreparedStatement p = null;** 🡪 Objeto para el paso de parámetros.

**ResultSet result = null;** 🡪 Almacén del resultado de la query.

Creamos la consulta y la almacenamos en un String:

**String can = "SELECT id, nom, tema, numVideos, (propietari).\*, subscriptors FROM canal WHERE id=?";**

Instanciamos el objeto “PreparedStatement” y le pasamos la consulta por parámetro:

**p = conn.prepareStatement(can);**

Capturamos el parámetro y pasamos la query al objeto **Result**:

**p.setInt(1, canId);**

**result = p.executeQuery();**

Ahora con el objeto “Result”, y el método “next()” miramos si hay algo:

**if(result.next()) {**

**Array arraySubscriptores = result.getArray("subscriptors");**

**String[] subscriptores = (String[])arraySubscriptores.getArray();**

**canal = new Canal(**

**result.getInt("id"),**

**result.getString("nom"),**

**result.getString("tema"),**

**result.getInt("numvideos"),**

**subscriptores,**

**result.getString("nick"),**

**result.getString("nomUsuari"),**

**result.getBoolean("majorEdat"),**

**result.getString("residencia"));**

**}**

Cosas a tener en cuenta:

* **Array arraySubscriptores = result.getArray("subscriptors");** 🡪 Creamos un objeto “Array” y le asignamos el resultado del “result” haciendo uso de su método “getArray(“campoTabla”).
* **String[] subscriptores = (String[])arraySubscriptores.getArray();** 🡪 Creamos un array de String y le asignamos el valor del objeto “**arraySubscriptores**” haciendo un “casting” con el tipo de dato que se espera. Para este caso es de tipo String. Si fuera otro tipo de dato, tendríamos que crear y castear el array con el tipo correspondiente.

**NOTA:** En el caso visto anteriormente, se esperaba obtener un solo objeto, si la consulta se quiere obtener varios objetos, lo primero es que en vez de un objeto de tipo Canal (Canal canal), tendría que ser una lista de Canales (List<Canal> canales = new ArrayList();). Otra cosa a tener en cuenta, es que si esperamos más de un objeto, en vez expresar << **if(result.next())** >>, tendríamos que pasarlo por un bucle “while” (**while(result.next())**).