1. Instalar PostgreSQL

1. Instalación
   La descarga de postgresql para todos los sistemas operativos está en:
   https://www.postgresql.org/download/

   Al realizar la instalación, dejar configurado:
   puerto: 5432
   password: abc123.
   Y al finalizar la instalación, seleccionar el dirver pgJDBC

   Podéis ver un vídeo de la instalación en Windows. https://youtu.be/cHGaDfzJyY4

2. Crearemos BBDD 'test'
3. Crearemos un usuario accesodatos con password 'abc123.', al que daremos privilegios totales en test.

```
postgres=# create user accesodatos WITH PASSWORD 'abc123.';
CREATE ROLE
postgres=# grant all privileges on database test to accesodatos;
GRANT
postgres=#
```

2. Conexión a una base de datos PostgreSQL

1. Añadir las dependencias en el archivo pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.18</version>
    </dependency>

</dependencies>
```

2. Conectarnos a la base de datos.
   Para conectarnos a la base de datos necesitamos (como en la mayoría de conexiones a la base de datos) la **URL**, el **nombre de la base de datos** y las **credenciales**.

```java
String url ="localhost";
String db = "test";

//Indicamos las propiedades de la conexión
Properties props = new Properties();
props.setProperty("user", "accesodatos");
props.setProperty("password", "abc123.");

//Dirección de conexión a la base de datos
String postgres = "jdbc:postgresql://"+url+"/"+db;

//Conectamos a la base de datos
try {
    Connection miConexion = DriverManager.getConnection(postgres,props);

    //Cerramos la conexión con la base de datos
    if(miConexion!=null) miConexion.close();
} catch (SQLException ex) {
    System.err.println("Error: " + ex.toString());
}
```

3. Tabla con archivo binario.

Creación de tabla con archivo binario

El tipo de datos a usar será **bytea**. En este caso guardaremos una imagen, aunque podría ser cualquier otro fichero binario.

```
//Creamos la tabla que contendra las imagenes
String sqlTableCreation =
    "CREATE TABLE IF NOT EXISTS imagenes (nombreimagen text, imagen bytea);";

//Ejecutamos la sentencia SQL anterior
CallableStatement createFunction = conn.prepareCall(sqlTableCreation);
createFunction.execute();
```

Inserción de una imagen

Para hacer un insert en la tabla necesitaremos leer los bytes de un fichero (visto en la unidad 2) y posteriormente insertarla en la base de datos.

```
    //tendremos una imagen en el proyecto
    File file = new File("pajaro.jpg");
    FileInputStream fis = null;

    fis = new FileInputStream(file);

    //Creamos la consulta que inserta la imagen en la base de datos
    PreparedStatement ps = null;
    String sqlInsert = "INSERT INTO imagenes VALUES (?, ?);";
    ps = miConexion.prepareStatement(sqlInsert);

    //pasamos los parámetros (1 - nombre del fichero, 2 - archivo y longitud
    ps.setString(1, file.getName());
    ps.setBinaryStream(2, fis, (int)file.length());

    //Ejecutamos la consulta
    ps.executeUpdate();

    //Cerrramos la consulta y el archivo abierto
    ps.close();
    fis.close();
```

Recuperar una imagen

Ahora vamos a recuperar la imagen de la BBDD y guardarla en un fichero cuyo nombre será “pajaro_copia.jpg”.

```
//Creamos la consulta para recuperar la imagen
String sqlSelect = "SELECT imagen FROM imagenes WHERE nombreimagen = ?;";
PreparedStatement ps2 = null;

ps2 = miConexion.prepareStatement(sqlSelect);
//Se le pasa el nombre como parámetro
ps2.setString(1, nombreFichero);
//Ejecutamos la consulta
ResultSet rs = ps2.executeQuery();

//Vamos recuperando todos los bytes de las imágenes
byte[] imgBytes = null;
while (rs.next())
{
   imgBytes = rs.getBytes(1);
}

//Cerramos a consulta
rs.close();
ps2.close();

//Creamos el flujo de datos para guardar el fichero recuperado
String ficheroSaida = new String("pajaro_copia.jpg");
File fileOut = new File(ficheroSaida);
FileOutputStream flujoDatos = new FileOutputStream(fileOut);

//Guardamos el fichero recuperado
if(imgBytes != null){
   flujoDatos.write(imgBytes);
}

//cerramos el flujo de datos de salida
flujoDatos.close();
```

4. Tabla con arrays.

PostgreSQL permite el uso de arrays. La información sobre este tipo de datos se encuentra en: https://www.postgresql.org/docs/current/arrays.html

Definición de una tabla en un array.

PostgreSQL permite crear un array dentro de una tabla, en este caso, vamos a ver un array de teléfonos:

```
//Creamos la tabla que contendrá el array
String createSQL =
       "CREATE TABLE IF NOT EXISTS telefonoArrays (" +
                   "id SERIAL PRIMARY KEY, " +
                   "nombre text, " +
                   "telefono INTEGER[]);";

//Ejecutamos la sentencia SQL anterior
CallableStatement createFunction = miConexion.prepareCall(createSQL);
createFunction = createFunction.execute();
createFunction.close();
```

Para realizar un insert de un array

Para poder añadir un array de objetos de JAVA hay que transformar ese array con el siguiente método:
Array createArrayOf(String typeName, Object[] elements) throws SQLException

```
String insertSQL = "INSERT INTO telefonoArrays (nombre,telefono) VALUES (?,?);";

Integer[] arrayTelefonos = {981123456, 982123456,986123456,988123456};

PreparedStatement prepInsert = miConexion.prepareStatement(insertSQL);
prepInsert.setString(1, "Jose");
prepInsert.setArray(2, miConexion.createArrayOf("INTEGER", arrayTelefonos));
prepInsert.execute();
prepInsert.close();
```

5. Crear tablas con tipos definidos por el usuario

https://www.postgresql.org/docs/13/rowtypes.html

Crear tablas con tipos de datos definidos por el usuario

Crearemos un tipo de dato, tipoDireccion y desde la tabla persona tendremos un campo dirección que será de tipo tipoDireccion.

```
//Creamos el tipo de datos direccion
String createTipo = "CREATE TYPE tipoDireccion AS (" +
        "    tipoVia            text," +
        "    nombreCalle        text," +
        "    numero     integer" +
        ");";
//Creamos la tabla que contendrá el tipo definido por nosotros
String createSQL = "CREATE TABLE IF NOT EXISTS persona (dni text, direccion
tipoDireccion);";

CallableStatement callstmt = null;

//Ejecutamos la sentencia crear Tipo
callstmt = miConexion.prepareCall(createTipo);
callstmt.execute();

//Ejecutamos la sentencia create Tabla
callstmt = miConexion.prepareCall(createSQL);
callstmt.execute();

//cerramos el CallableStatement
callstmt.close();
```

Añadir datos

```
// Definimos la query para añadir valores
String insertSQL = "INSERT INTO persona (dni,direccion) VALUES (?,ROW(?,?,?))";

// preparamos la query y sus parámetros
PreparedStatement prepInsert = miConexion.prepareStatement(insertSQL);
prepInsert.setString(1, dni);
prepInsert.setString(2, tipoVia);
prepInsert.setString(3, calle);
prepInsert.setInt(4, numero);

//se ejecuta y cierra el preparedStatement
prepInsert.execute();
prepInsert.close();
```

Consultar datos

Los datos del tipo definido por nosotros, deben de ir precedidos del nombre del campo, de la tabla. (nombreCampoTabla.nombreTipo)

```
//Define la query de consulta
String consultaSQL =
"SELECT (direccion).nombrecalle,dni FROM persona WHERE (direccion).numero = ?";

// preparamos la query y sus parámetros.
// Seleccionamos los que viven en el numero 12.
PreparedStatement prepInsert = miConexion.prepareStatement(consultaSQL);
prepInsert.setInt(1, 12);

//se ejecuta y cierra el preparedStatement
prepInsert.execute();
prepInsert.close();
```

6. Declaración y llamada de funciones

Declaración de funciones

El lenguaje empleado para definir funciones es PLPSQL. Vamos a ver un ejemplo de cómo se define una función, que lo que va a realizar es incrementar 1 el valor que se le pasa como parámetro.

```
//Creamos la sentencia SQL para crear unha función
String funcionSQL = "CREATE OR REPLACE FUNCTION incrementa(val integer) RETURNS integer
AS $$ " +
        "BEGIN " +
        "RETURN val + 1; " +
        "END;" +
        "$$ LANGUAGE PLPGSQL;";

//Ejecutamos la sentencia SQL anterior
CallableStatement createFunction = miConexion.prepareCall(funcionSQL);
createFunction.execute();
createFunction.close();
```

Los símbolos $$ son para que no finalice la sentencia con los ; que van dentro de la función.

Llamada a funciones

```
//Creamos la llamada a la función
String sqlCallFunction = "{? = call incrementa( ? ) }";
CallableStatement callFunction = miConexion.prepareCall(sqlCallFunction);

//El primer parámetro indica el tipo de datos que devuelve
callFunction.registerOutParameter(1, Types.INTEGER);

//El segundo parámetro indica el valor que le pasamos a la función
callFunction.setInt(2,5);

//Ejecutamos la función, devolverá el número siguiente a 5.
callFunction.execute();

//Obtenemos el valor resultante de la función
int valorDevuelto = callFunction.getInt(1);
callFunction.close();

//Mostramos el valor que devuelve la función
System.out.println("El número que va a continuación es: " +  valorDevuelto);
```

7. Notificaciones en PostgreSQL

https://www.postgresql.org/docs/12/sql-notify.html

PostgreSQL permite que el propio gestor de base de datos avise al cliente de que ocurrió un evento. En JAVA no es un lenguaje asíncrono, con lo cuál no se le saca todo el partido, por lo tanto, tenemos que chequear cada cierto tiempo que hay notificaciones.

El proceso que vamos a realizar en este ejemplo es:

Lanzaremos una notificación cada vez que se añada una persona a la tabla 'persona'.
Para ello se necesitan varios pasos:

1. Crear trigger y función asociado, lo estamos haciendo desde java, pero lo normal es que esto ya esté generado en la BBDD
2. Nos suscribimos al canal 'nuevaPersona' para recibir notificaciones.
3. Cada cierto tiempo comprobamos si hay notificaciones.

Crear la función

https://www.postgresql.org/docs/9.1/plpgsql-statements.html (perform)
https://www.postgresql.org/docs/12/sql-notify.html (pg_notify)
https://www.postgresql.org/docs/9.2/plpgsql-trigger.html (trigger procedure)

No vamos a entrar en la definición de la función. Simplemente la función pg_notify, que lo que hace es notificación, cuyo canal será 'nuevapersona', y el payload el dni de la persona.

```
//Creamos la función asociada al trigger. Notificará cuando se añada una persona
String sqlCreateFunction = "CREATE OR REPLACE FUNCTION notificar_persona() " +
       "RETURNS trigger AS $$ " +
       "BEGIN " +
       "PERFORM pg_notify('nuevapersona',NEW.dni); " +
        "RETURN NEW; " +
       "END; " +
       "$$ LANGUAGE plpgsql; ";
CallableStatement createFunction = miConexion.prepareCall(sqlCreateFunction);
createFunction.execute();
createFunction.close();
```

Crear trigger

Lanzamos un trigger cada vez que se realiza un INSERT sobre la tabla 'persona', este lanzará la funcion "notificar_persona", que genera una notificación.

```
//Creamos el trigger que se ejecuta cuando se añade una persona
String sqlCreateTrigger = "DROP TRIGGER IF EXISTS trigger_persona ON persona; " +
       "CREATE TRIGGER trigger_persona " +
       "AFTER INSERT ON persona FOR EACH ROW " +
       "EXECUTE PROCEDURE notificar_persona(); ";
CallableStatement createTrigger = miConexion.prepareCall(sqlCreateTrigger);
createTrigger.execute();
createTrigger.close();
```

Suscribirse al canal

Cuando se haga un insert se creará una notificación en el canal “nuevaPersona”, con las siguientes líneas, estaremos a la escucha de esas notificaciones.

```
//Configuramos para estar a la escucha
pgconn = miConexion.unwrap(PGConnection.class);
Statement stmt = miConexion.createStatement();
stmt.execute("LISTEN nuevapersona;");
stmt.close();
System.out.println("Esperando cambios...");
```

Recoger notificaciones

Para cada notificación que nos llegue, realizaremos una consulta para recuperar sus datos y los mostraremos por pantalla

```
//Recogemos todas las notificaciones
PGNotification[] notifications = pgconn.getNotifications();

//Si hay notificaciones, las recorremos e imprimimos
if(notifications != null){
   for(int i=0;i < notifications.length;i++){

        //La notificacion nos da como parámetro el dni de la persona
        String dni = notifications[i].getParameter();

        //Hacemos una consulta a la BBDD y recuperamos los teléfonos asociados
        PreparedStatement sqlMensaxe = miConexion.prepareStatement(
                "SELECT dni, direccion FROM persona WHERE dni=?;");

        sqlMensaxe.setString(1, dni);

        ResultSet rs = sqlMensaxe.executeQuery();
        rs.next();
        System.out.println(rs.getString(1) + ":" + rs.getString(2));
        rs.close();
   }
}
```

Repetir el proceso

El proceso estará a la espera durante un minuto, y estará comprobando cada segundo las notificaciones.

```
//Tiempo en minutos que estara pendiente de las notificaciones
int tiempo = 1;

//Tiempo que espera para cada chequeo de notificacion (en milisegundos)
int espera = 1000;

//Variables para controlar el tempo de espera
boolean flag=true;

…
…
…
…
…
```

```
long finishAt = new java.util.Date().getTime() + (tiempo * 60000);

//Bucle para ir leyendo los mensajes
while(flag){
   recogeNotificacion(pgconn);

   //Esperamos entre comprobaciones
   Thread.sleep(espera);

   //Se revisa si llegamos al final del tiempo
   if(new Date().getTime() > finishAt) flag=false;
}
```

Dejar de escuchar el canal

```
//dejamos de escuchar el canal
Statement stmt2 = miConexion.createStatement();
stmt2.execute("UNLISTEN nuevapersona;");
stmt2.close();
```