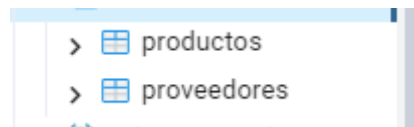
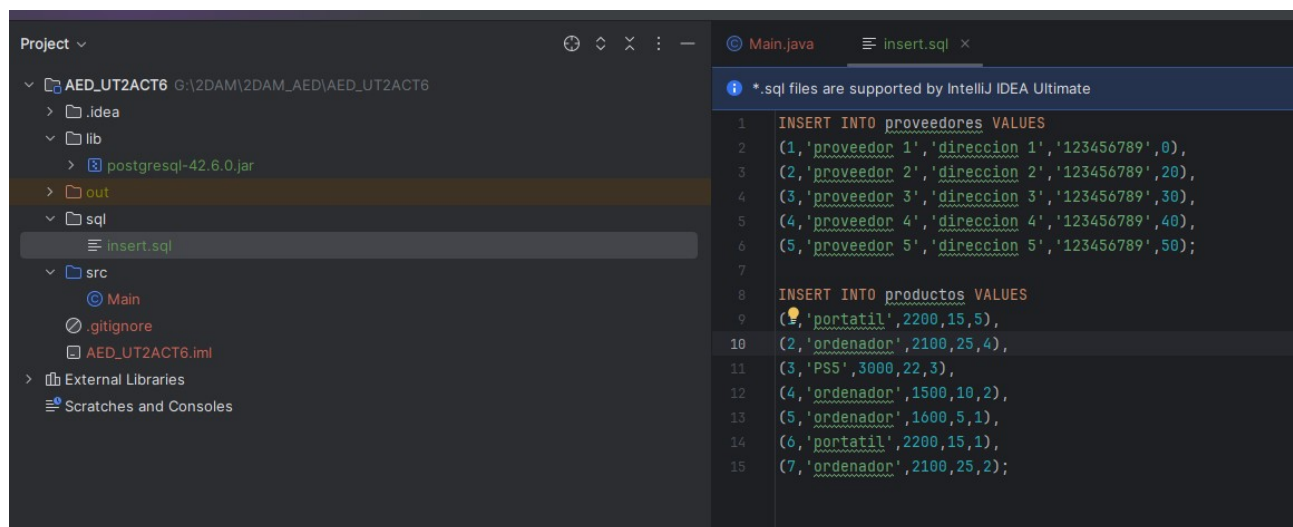


1. Creacion de tablas

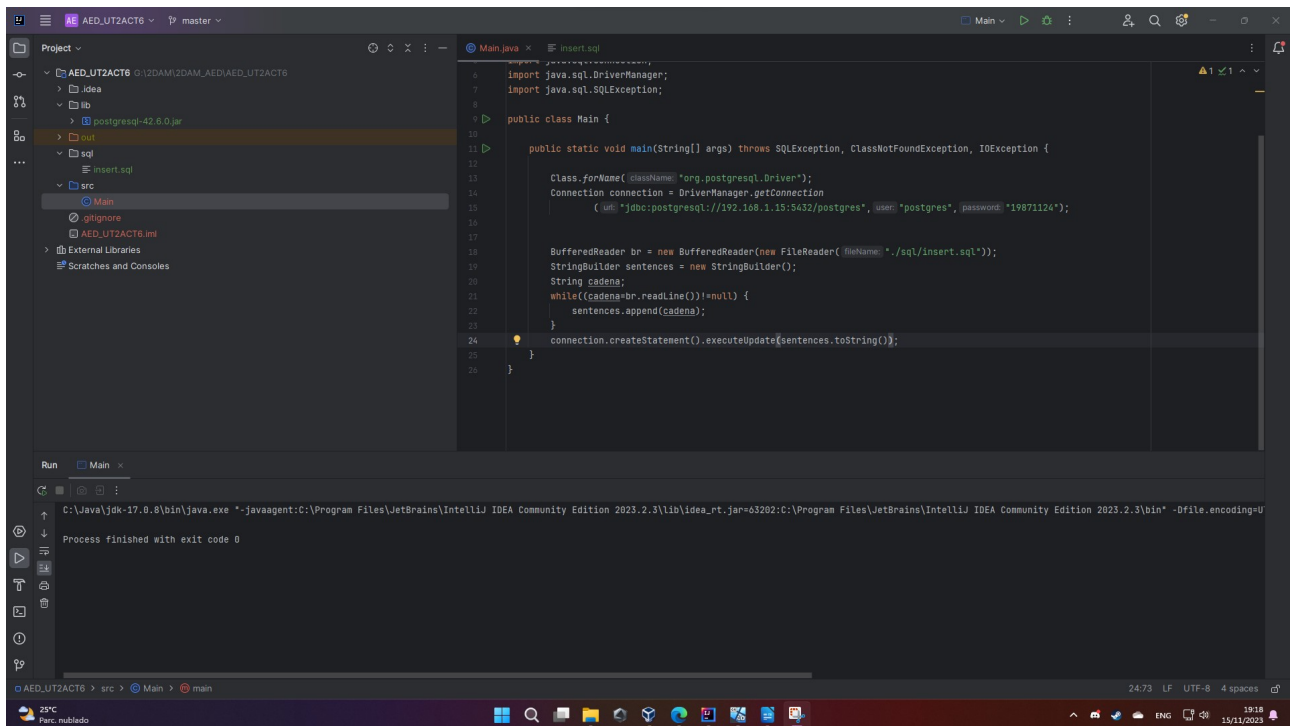
```
postgres=# CREATE TABLE proveedores (  
postgres=#   cod_prov CHAR(4) PRIMARY KEY NOT NULL,  
postgres=#   nombre_prov CHAR(30) NOT NULL,  
postgres=#   direccion CHAR(30) NOT NULL,  
postgres=#   telefono CHAR(9) NOT NULL,  
postgres=#   bonifica INT NOT NULL  
postgres=# );  
CREATE TABLE  
postgres=# CREATE TABLE productos (  
postgres=#   cod_prod CHAR(5) PRIMARY KEY NOT NULL,  
postgres=#   nombre_prod CHAR(30) NOT NULL,  
postgres=#   precio DECIMAL(10,2) NOT NULL,  
postgres=#   stock INT NOT NULL,  
postgres=#   cod_prov CHAR(4) NOT NULL,  
postgres=#   CONSTRAINT prod_codProv FOREIGN KEY (cod_prov) REFERENCES proveedores (cod_prov)  
postgres=# );  
CREATE TABLE  
postgres=#
```



2. Sentencia de insercion guardado en fichero



Realizar la conexión al base de datos y insertar los datos (Hice con maquina virtual)



The screenshot shows the IntelliJ IDEA IDE with a project named 'AED_UT2ACT6'. The 'src' directory contains a file named 'insert.sql'. The 'Main.java' file is open, showing a Java program that connects to a PostgreSQL database and inserts data from a file.

```
import java.sql.DriverManager;
import java.sql.SQLException;
import java.io.*;

public class Main {

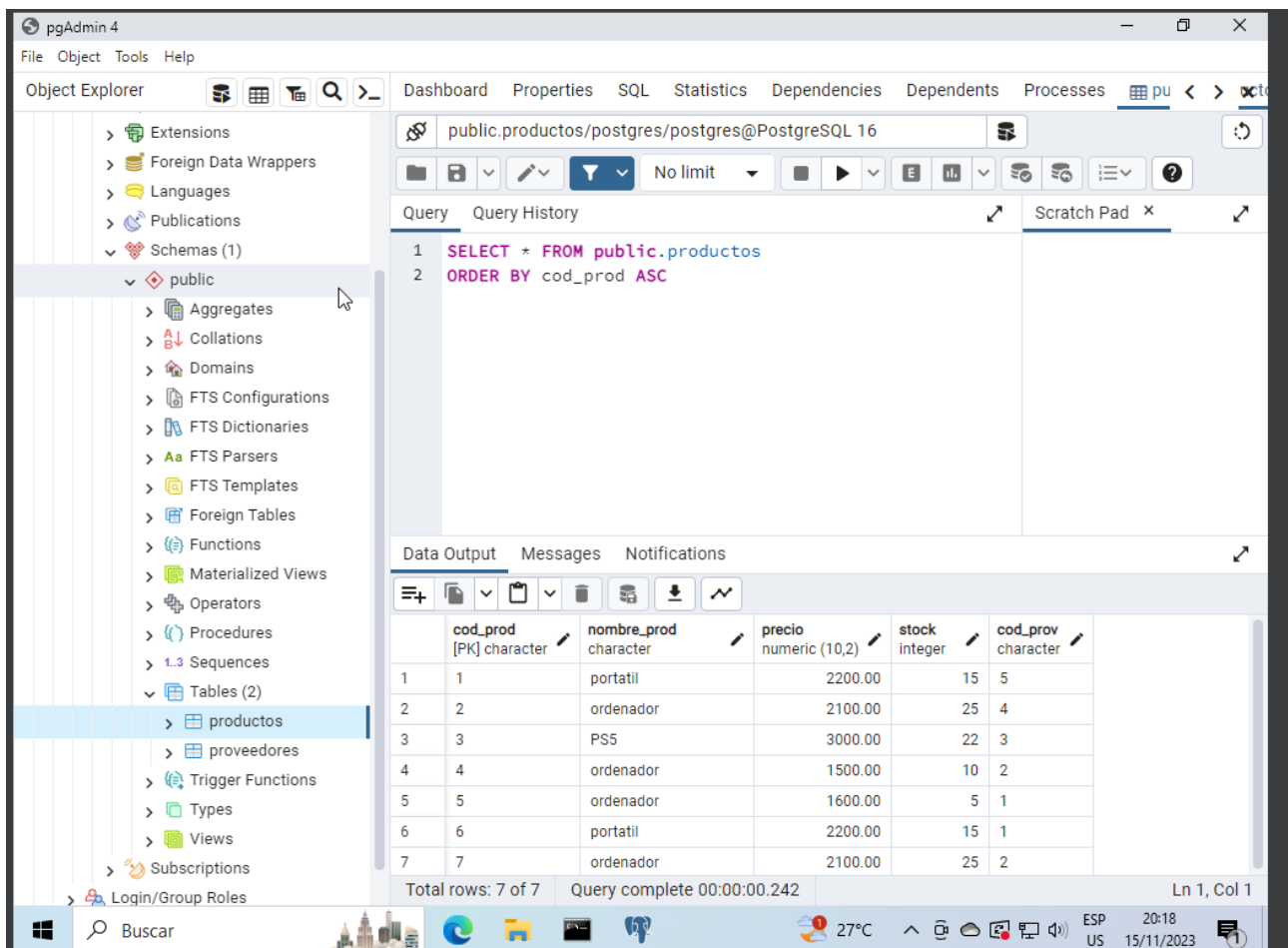
    public static void main(String[] args) throws SQLException, ClassNotFoundException, IOException {

        Class.forName("org.postgresql.Driver");
        Connection connection = DriverManager.getConnection(
            "jdbc:postgresql://192.168.1.15:5432/postgres", "postgres", "19871124");

        BufferedReader br = new BufferedReader(new FileReader("src/insert.sql"));
        StringBuilder sentences = new StringBuilder();
        String cadena;
        while((cadena=br.readLine())!=null) {
            sentences.append(cadena);
        }
        connection.createStatement().executeUpdate(sentences.toString());
    }
}
```

The Run console shows the command executed: `C:\Java\jdk-17.0.8\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.3\lib\idea_rt.jar=63202:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.3\bin -Dfile.encoding=UTF-8`. The process finished with exit code 0.

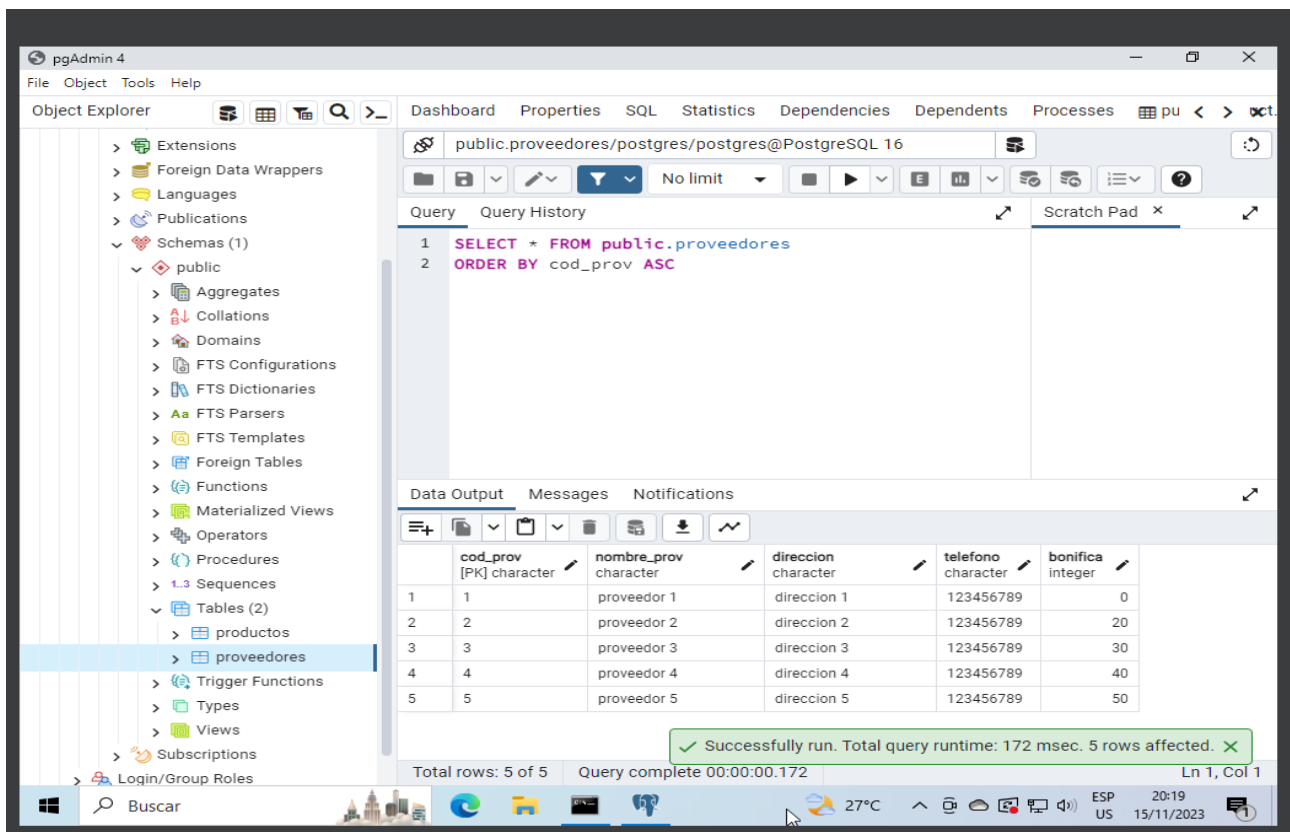
Datos guardados



The screenshot shows the pgAdmin 4 interface. The 'Object Explorer' on the left shows the 'public' schema. The 'Query' tab is selected, and the query is: `SELECT * FROM public.productos ORDER BY cod_prod ASC`. The 'Data Output' tab shows the results of the query.

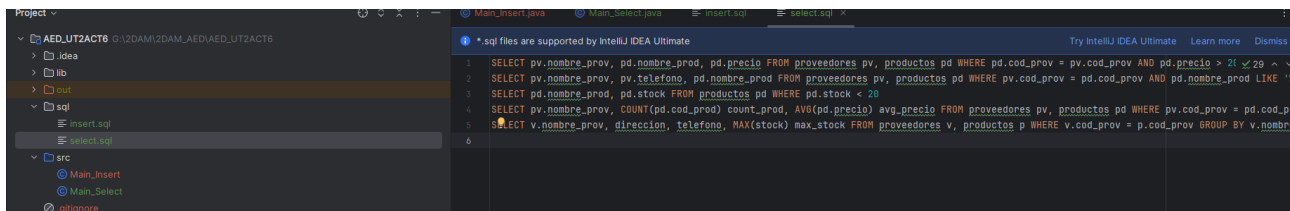
	cod_prod [PK] character	nombre_prod character	precio numeric (10,2)	stock integer	cod_prov character
1	1	portatil	2200.00	15	5
2	2	ordenador	2100.00	25	4
3	3	PS5	3000.00	22	3
4	4	ordenador	1500.00	10	2
5	5	ordenador	1600.00	5	1
6	6	portatil	2200.00	15	1
7	7	ordenador	2100.00	25	2

Total rows: 7 of 7 Query complete 00:00:00.242 Ln 1, Col 1



3. Consultar al base de datos desde java

Sentencia guardado en un fichero



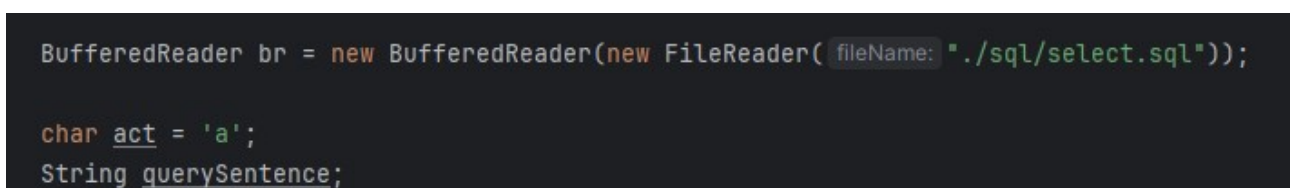
Conectar al base de datos



BufferedReader para recorrer cada linea de fichero, cada linea es una sentencia de select

Variable act para registrar de que parte de ejercicio estamos

Variable querySentence para coger cada linea de sentencias



Un bucle while para recorrer todas las sentencias select registrado en el fichero

Cada vez realizar una sentencia de select se va a volver asignar un nuevo resultado de consulta.

Utilizar getMetaData() para obtener cuantas columna tiene y su nombre.

Utilizar un bucle for para recorrer los nombres de columna de esta vez de consulta y meterlo al lista

```
while((querySentence = br.readLine())!=null) {

    ResultSet resultSet = connection.createStatement().executeQuery(querySentence);
    ResultSetMetaData rsm = resultSet.getMetaData();
    ArrayList<String[]> al = new ArrayList<>();

    System.out.println("Ejercicio select " + act + ":");

    String[] columnName = new String[rsm.getColumnCount()];
    for (int i = 1; i <= rsm.getColumnCount(); i++) {
        columnName[i-1] = rsm.getColumnName(i).trim();
    }
    al.add(columnName);
}
```

Luego a coger cada row de datos de esta vez de consulta y meterlo a una vector, cuando termina de coger una linea de row, dale este row a la lista (es una Matriz) y mueve al siguiente row con next().

Una vez terminada de una sentencia de consulta, se va a recorrer todos los resultados guardados en la ArrayList, primera for para recorrer rows y dentro de otro for para recorrer columna.

Y al ultimo se va a cambiar el valor de caracter que es para decir en que parte de ejercicio estamos (variable act('a') ASCII value = 97 + 1 = 98) encoding to 'b'

```
while(resultSet.next()) {
    String[] row = new String[rsm.getColumnCount()];
    for (int i = 1; i <= rsm.getColumnCount(); i++) {
        row[i-1] = resultSet.getString(rsm.getColumnName(i)).trim();
    }
    al.add(row);
}

for (String[] strings : al) {
    for (int j = 0; j < rsm.getColumnCount(); j++) {
        System.out.print(strings[j] + "    ");
    }
    System.out.println();
}
System.out.println();
act = (char) (act+1);
}
```

C:\Java\jdk-17.0.8\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Co

Ejercicio select a:

nombre_prov	nombre_prod	precio
proveedor 3	PS5	3000.00
proveedor 5	portatil	2200.00
proveedor 1	portatil	2200.00
proveedor 2	ordenador	2100.00
proveedor 4	ordenador	2100.00

Ejercicio select b:

nombre_prov	telefono	nombre_prod
proveedor 2	123456789	ordenador
proveedor 1	123456789	ordenador
proveedor 2	123456789	ordenador
proveedor 4	123456789	ordenador

Ejercicio select c:

nombre_prod	stock
portatil	15
ordenador	10
ordenador	5
portatil	15
ordenador	5

Ejercicio select d:

nombre_prov	count_prod	avg_precio
proveedor 2	2	1800.0000000000000000
proveedor 5	1	2200.0000000000000000
proveedor 4	1	2100.0000000000000000
proveedor 3	1	3000.0000000000000000
proveedor 1	2	1900.0000000000000000

Ejercicio select e:

nombre_prov	direccion	telefono	max_stock
proveedor 1	direccion 1	123456789	15
proveedor 4	direccion 4	123456789	5
proveedor 2	direccion 2	123456789	25
proveedor 3	direccion 3	123456789	22
proveedor 5	direccion 5	123456789	15

4. Creacion de procedimiento almacenado

(Hice el parametro con un decimal y multiplicarlo con bonifica es porque me da una problema de calculo en postgre es que hago "SET bonifica = bonifica * (1 + (porcentaje/100))" siempre me da igual los datos, y no se donde esta la razon)

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Schemas (1)' tree is expanded to 'public', and 'Procedures' is selected. The main query editor displays the following SQL code:

```
1 CREATE OR REPLACE PROCEDURE update_bonifica(codProv CHAR, per FLOAT)
2 LANGUAGE 'plpgsql'
3 AS $$
4 BEGIN
5     UPDATE proveedores SET bonifica = bonifica * per WHERE cod_prov = codProv;
6 END;
7 $$
8
```

Below the query editor, the 'Data Output' tab is active, showing the message 'CREATE PROCEDURE'.

Como eso, no se donde esta mal calculo, siempre me da mismo datos

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Schemas (1)' tree is expanded to 'public', and 'Trigger Functions' is selected. The main query editor displays the following SQL code:

```
1 UPDATE proveedores SET bonifica = bonifica * (1 + (10/100)) WHERE cod_prov = '2';
2 SELECT * FROM proveedores;
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The table has the following columns: cod_prov [PK] character, nombre_prov character, direccion character, telefono character, and bonifica integer.

	cod_prov [PK] character	nombre_prov character	direccion character	telefono character	bonifica integer
3	4	proveedor 4	direccion 4	123456789	40
4	5	proveedor 5	direccion 5	123456789	50
5	2	proveedor 2	direccion 2	123456789	20

Llamar el procedimiento almacenado, codigo de proveedor = 2, multiplicar bonifica por 0.1

```
CallableStatement call = connection.prepareCall( sql: "CALL update_bonifica(?,?)");
call.setString( parameterIndex: 1, x: "3");
call.setDouble( parameterIndex: 2, x: 0.10);
call.executeUpdate();
```

Ante bonifica es 30 y ahora es 3

	cod_prov [PK] character	nombre_prov character	direccion character	telefono character	bonifica integer
3	5	proveedor 5	direccion 5	123456789	50
4	2	proveedor 2	direccion 2	123456789	20
5	3	proveedor 3	direccion 3	123456789	3

5. Creacion de trigger

Crear tabla pedidos_pendientes para despues de update a insertar Log

The screenshot shows a database management tool interface. On the left, a tree view displays the database structure, including Schemas, public, Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures (1), Sequences, and Tables (2). The 'Tables (2)' folder is expanded, showing 'productos' and 'proveedores'. The main area displays a SQL query in the 'Query' tab:

```
1 CREATE TABLE pedidos_pendientes (
2   cod_prov CHAR(4) NOT NULL,
3   stock INT NOT NULL,
4   fecha_mod DATE NOT NULL
5 )
```

Below the query, the 'Data Output' tab shows the result of the query execution:

```
CREATE TABLE
Query returned successfully in 104 msec.
```


Crear una funcion primero para luego usar por procedimiento almacenado
Cuando el nuevo stock actualizado es menor de 10 se va a insertar un log al tabla pedido_pendiente

The screenshot shows the PostgreSQL interface with the 'public' schema selected. The 'Trigger Functions' folder is expanded, showing the 'insertlog()' function. The 'Query' tab is active, displaying the following SQL code:

```
1 CREATE FUNCTION insertLog() RETURNS TRIGGER
2 LANGUAGE 'plpgsql'
3 AS $$
4 BEGIN
5     IF new.stock < 10 THEN
6         INSERT INTO pedidos_pendientes VALUES (new.cod_prov, new.stock, CURRENT_DATE);
7     END IF;
8     RETURN NEW;
9 END;
10 $$;
11
12 CREATE TRIGGER stockPendientes
13 BEFORE UPDATE ON productos FOR EACH ROW
14 EXECUTE PROCEDURE insertLog();
```

The 'Data Output' tab is also visible, showing the message: 'Query returned successfully in 49 msec.'

Actualizar el stock de cod_producto = 2, y esta insertado al pedido_pendientes

The screenshot shows the PostgreSQL interface with the 'public' schema selected. The 'Query' tab is active, displaying the following SQL code:

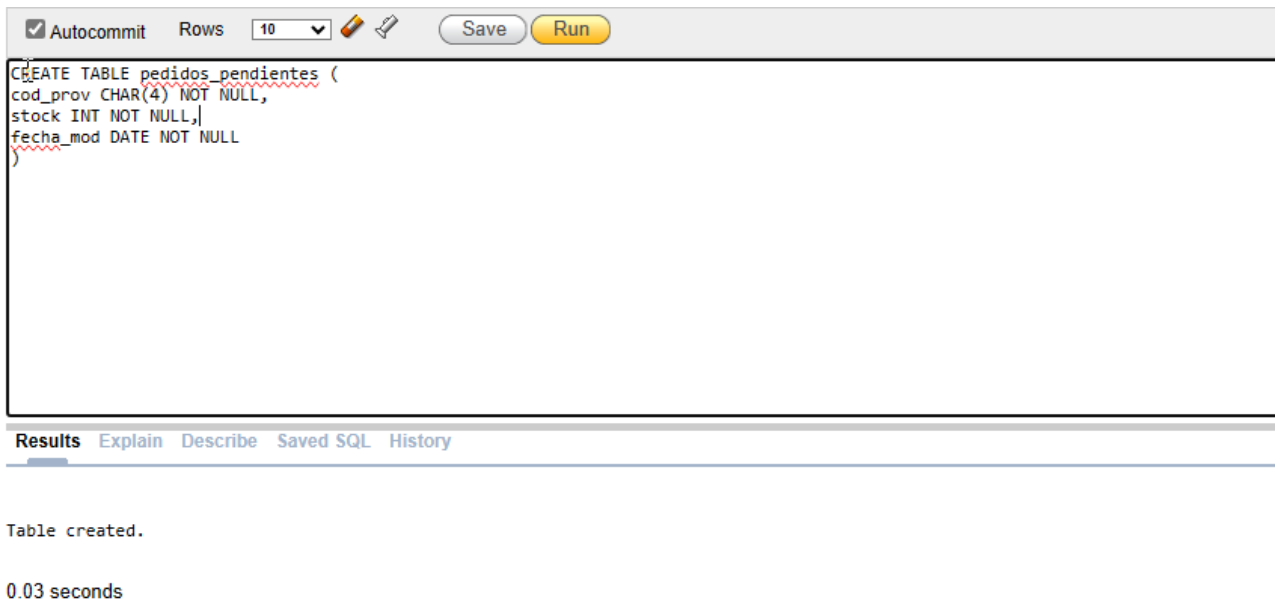
```
1 UPDATE productos SET stock = 5 WHERE cod_prod = '2';
2 SELECT * FROM pedidos_pendientes;
```

The 'Data Output' tab is also visible, showing the result of the query:

	cod_prov	stock	fecha_mod
1	4	5	2023-11-16

ORACLE

Creacion de tabla son iguales



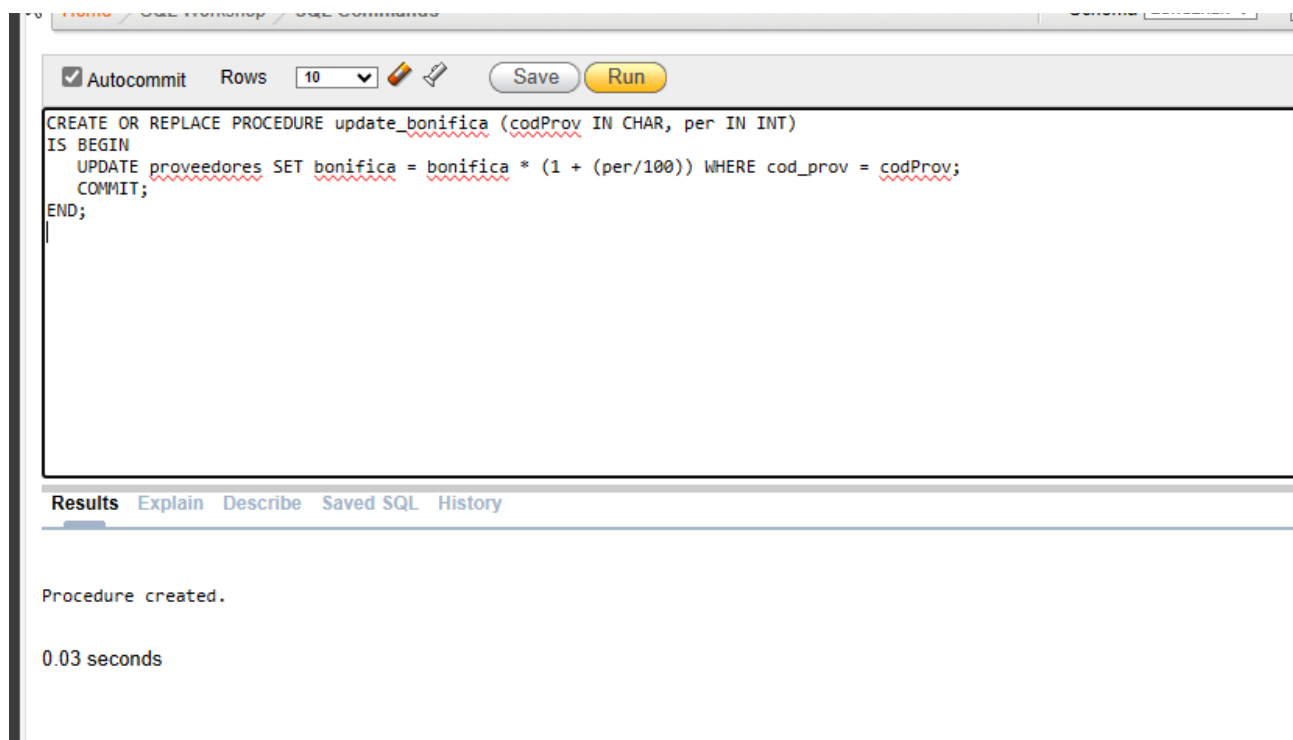
The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with a checked 'Autocommit' checkbox, a 'Rows' dropdown set to '10', and 'Save' and 'Run' buttons. The main text area contains the following SQL code:

```
CREATE TABLE pedidos_pendientes (  
  cod_prov CHAR(4) NOT NULL,  
  stock INT NOT NULL,  
  fecha_mod DATE NOT NULL  
)
```

Below the text area, there is a tabbed interface with 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History' tabs. The 'Results' tab is active and displays the message 'Table created.' followed by '0.03 seconds'.

Creacion de procedimiento almacenado la sintaxis es algo diferente

Y me da bien al hacer $\text{bonifica} * (1 + (\text{per}/100))$, anterior en postgre no me da bien



The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with a checked 'Autocommit' checkbox, a 'Rows' dropdown set to '10', and 'Save' and 'Run' buttons. The main text area contains the following SQL code:

```
CREATE OR REPLACE PROCEDURE update_bonifica (codProv IN CHAR, per IN INT)  
IS  
BEGIN  
  UPDATE proveedores SET bonifica = bonifica * (1 + (per/100)) WHERE cod_prov = codProv;  
  COMMIT;  
END;
```

Below the text area, there is a tabbed interface with 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History' tabs. The 'Results' tab is active and displays the message 'Procedure created.' followed by '0.03 seconds'.

Llamar procedimiento almacenado desde java

```
CallableStatement call = connection.prepareCall(sql: "CALL update_bonifica(?,?)");
call.setString(parameterIndex: 1, x: "3");
call.setDouble(parameterIndex: 2, x: 10);
call.executeUpdate();
```

Resultado tambien esta bien, ante es 30 y ahora es 33 ($30 * (1 + (10/100))$)

EDIT	COD_PROV	NOMBRE_PROV	DIRECCION	TELEFONO	BONIFICA
	1	proveedor 1	direccion 1	123456789	0
	2	proveedor 2	direccion 2	123456789	20
	3	proveedor 3	direccion 3	123456789	33
	4	proveedor 4	direccion 4	123456789	40
	5	proveedor 5	direccion 5	123456789	50
row(s) 1 - 5 of 5					

Creacion de trigger, en oracle no tiene que crear la funcion

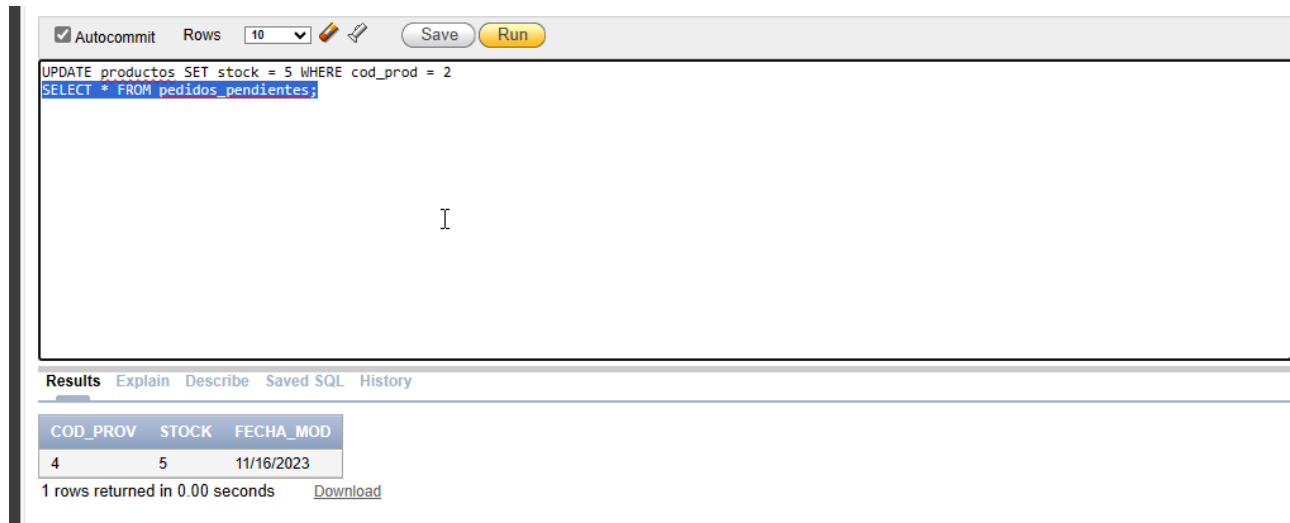
```
CREATE OR REPLACE TRIGGER insertStockLog
BEFORE UPDATE ON productos
FOR EACH ROW
BEGIN
  IF :new.stock < 10 THEN
    INSERT INTO pedidos_pendientes VALUES (:new.cod_prov, :new.stock, SYSDATE);
  END IF;
END;
```

Results Explain Describe Saved SQL History

Trigger created.

0.03 seconds

Cuando despues de actualizar y el stock tiene un menor de 10 se va a insertar al esta tabla (cod_prod 2 su cod_prov es 4)



The screenshot shows a SQL IDE interface. At the top, there's a toolbar with 'Autocommit' checked, 'Rows' set to 10, and 'Save' and 'Run' buttons. The main text area contains the following SQL code:

```
UPDATE productos SET stock = 5 WHERE cod_prod = 2  
SELECT * FROM pedidos_pendientes;
```

Below the code area, there's a 'Results' tab. It shows a table with the following data:

COD_PROV	STOCK	FECHA_MOD
4	5	11/16/2023

At the bottom, it says '1 rows returned in 0.00 seconds' and has a 'Download' link.

Conexion

```
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.97:1521:XE", user: "ZONGZHEN", password: "19871124");
```

Sintaxis de insertar tambien tiene diferencia con postgre

```
INSERT ALL  
  INTO proveedores (cod_prov, nombre_prov, direccion, telefono, bonifica) VALUES ('1','proveedor 1','direccion 1','123456789',0)  
  INTO proveedores (cod_prov, nombre_prov, direccion, telefono, bonifica) VALUES ('2','proveedor 2','direccion 2','123456789',20)  
  INTO proveedores (cod_prov, nombre_prov, direccion, telefono, bonifica) VALUES ('3','proveedor 3','direccion 3','123456789',30)  
  INTO proveedores (cod_prov, nombre_prov, direccion, telefono, bonifica) VALUES ('4','proveedor 4','direccion 4','123456789',40)  
  INTO proveedores (cod_prov, nombre_prov, direccion, telefono, bonifica) VALUES ('5','proveedor 5','direccion 5','123456789',50)  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('1','portatil',2200,15,'5')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('2','ordenador',2100,25,'4')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('3','PS5',3000,22,'3')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('4','ordenador',1500,10,'2')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('5','ordenador',1600,5,'1')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('6','portatil',2200,15,'1')  
  INTO productos (cod_prod, nombre_prod, precio, stock, cod_prov) VALUES ('7','ordenador',2100,25,'2')  
SELECT 1 FROM dual
```

Sintaxis de consultar son iguales

Apartado de update lo pongo en la programa

```
connection.createStatement().executeUpdate  
  ("sql: *UPDATE productos pd SET pd.precio = (pd.precio * 0.95) WHERE pd.cod_prov IN (SELECT cod_prov FROM proveedores WHERE bonifica = 0)");
```