



UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TEORÍA DE BASES DE DATOS

SEMESTRE 2021-1

PROYECTO FINAL

LOPERENA ALCÁNTARA
MARISOL VIRIDIANA

ROJAS MÉNDEZ GABRIEL

25 DE ENERO DE 2020

GRUPO 1

OBJETIVO GENERAL:

El alumno analizará una serie de requerimientos y propondrá una solución que atienda a los mismos, aplicando los conceptos vistos en el curso.

OBJETIVO PARTICULARES:

- Aplicar cada uno de los conceptos aprendidos en el curso.
- Obtener nuevos conocimientos para poder tener una interfaz gráfica de la base de datos.

INTRODUCCIÓN:

Se solicita diseñar una base de datos para una cadena de papelerías que se encuentra en busca de innovar la manera en almacenar su información. De acuerdo al requerimiento para los proveedores se requiere almacenar los siguientes datos:

- Id proveedor, el cual es un número único que permite identificar a cada uno de los proveedores.
- Razón Social
- Domicilio, está compuesto por calle, número, C.P. y estado.
- Nombre, está compuesto por nombre de pila, apellido paterno, considerando que el proveedor puede o no contar con apellido materno.
- Teléfonos, en donde se debe permitir almacenar más de un número telefónico.

Para los clientes se está solicitando almacenar los siguientes datos:

- Razón social
- Nombre, está compuesto por nombre de pila, apellido paterno y considerando que el cliente puede o no contar con el apellido materno.
- Domicilio, se compone por calle, número, C.P. y estado.
- Email, este tipo de dato permite almacenar más de un correo electrónico.

Para productos se está solicitando almacenar los siguientes datos:

- Código de barras, se compone de una serie numérica única, con el que se identificara a los productos.
- Precio en el que se compró el producto, se solicita este dato para poder llevar un historial del precio del producto.
- Fecha de compra, se solicita este dato para poder tener un control de venta de los productos.
- Cantidad, es un dato que permite conocer cuantitativamente lo que se tiene en almacén.
- Marca, permite conocer las marcas que se tienen de los productos.
- Descripción, almacenas características específicas del producto.
- Precio de regalos, almacenar el precio de regalos para poder consultarlos.
- Artículo de papelería, conocer con qué artículos se cuenta respecto a papelería.
- Impresiones, almacena los servicios de impresiones
- Recargas, almacenar cuantitativamente el servicio de recargas

Para venta se solicita almacenar los siguientes datos:

- Número de venta, identificador único de cada venta.
- Fecha de venta, almacena fecha en que se realizó la venta
- Total de compra, para conocer el monto total de la compra realizada.

Para cumplir con el punto dos de nuestro requerimiento, el cual solicita crear una interfaz gráfica vía app móvil o web, se requirió de investigación y propios conocimientos para determinar cómo se realizaría este punto. Así que se propuso utilizar Django, dado que es un innovador software que permite la creación de sitios web, brindándonos la conexión con nuestra base de datos, además de ser recomendada para el manejador de PostgreSQL y soporta el lenguaje Python, con el cual además se desarrolló este software y nosotros en materias anteriores como "Estructura de Datos y Algoritmos II" utilizamos este lenguaje.

PLAN DE TRABAJO:

Una vez que el profesor nos compartió los requerimientos para el proyecto de la materia de bases de datos.

Se prosiguió a tener una primera reunión para acordar los días en que se trabajaría en el proyecto.

De este modo las reuniones se agendaron para miercoles, viernes y sabado, para dar a conocer la planeación de la semana, analizar el requerimiento y comentar las posibles soluciones.

Aunque se puede observar que sábado se marca como día para trabajar en el proyecto, a lo largo de la semana también se trabajo en el proyecto pero fue de acuerdo a los tiempos que se tuvieron dada la carga de trabajo de las diferentes materias y laboratorios.

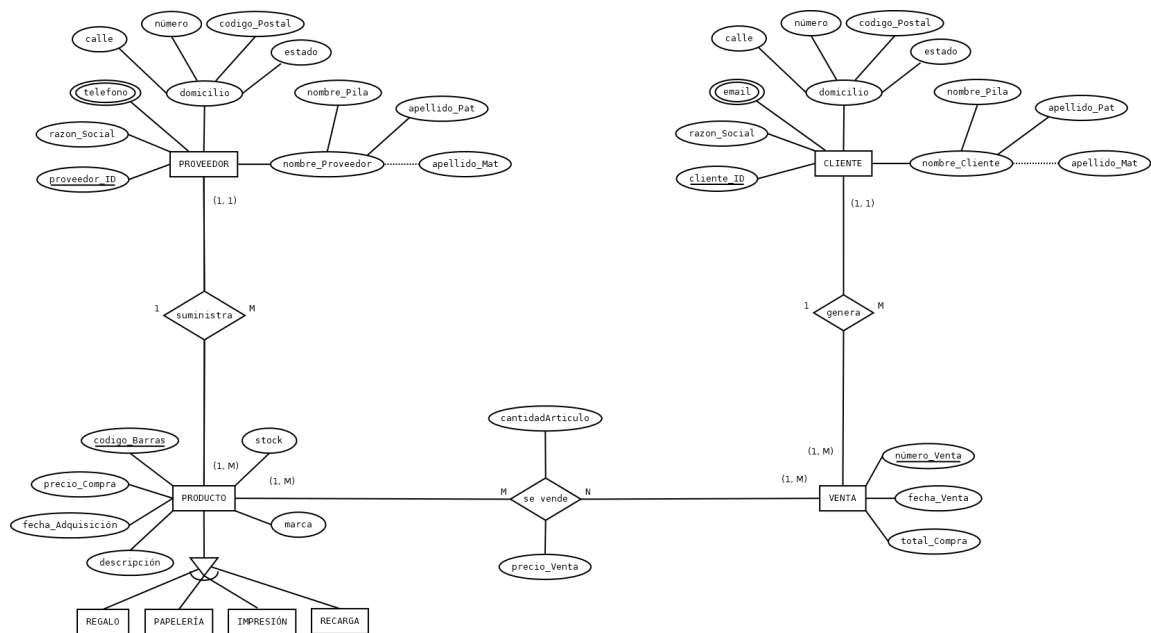
HORA	LUNES	MARTES	MIERCOLES	JUEVES	VIERNES	SABADO	DOMINGO
7 am a 9 am	Clase BD		Clase BD		Clase BD		
9 am a 11 am							
11 am a 1 pm						Tareas, cursos, proyectos, investigación	
1 pm a 3 pm	Clases y tareas	Clases y tareas	Clases y tareas	Clases y tareas	Clases y tareas	REUNIÓN	Entrega de prácticas
5 pm a 7 pm							
7 pm a 9 pm			REUNIÓN		REUNIÓN	Tareas, cursos, proyectos, investigación	

A cada uno de los integrantes le correspondía investigar de acuerdo a las necesidades que surgían en el proyecto, para que en la reunión correspondiente, se estuviera al mismo nivel de conocimientos y de esta manera se implementara en el proyecto. Se usó el software Draw.io, para en conjunto realizar los diagramas de modelo entidad - relación y modelo relacional, esto es debido a que durante el curso se utilizó como herramienta de apoyo y por la facilidad de poder colaborar ambos.

DISEÑO:

Para comenzar a diseñar la base de datos, se recurrió al modelo entidad **Entidad Relación**, el cual después de hacer leído detenidamente el caso de estudio, se fueron extrayendo las entidades correspondientes y sus respectivos atributos.

Posteriormente, mediante las reglas de negocio se fueron determinando la relaciones que había entre cada una de las entidades, y así, poder ir dándole formato al diseño.



Una vez que el diseño entidad relación fue adecuado, se continuó con el segundo paso, el cual fue la representación intermedia, la cual nos permitiría adecuar el resultado obtenido al modelo relacional.

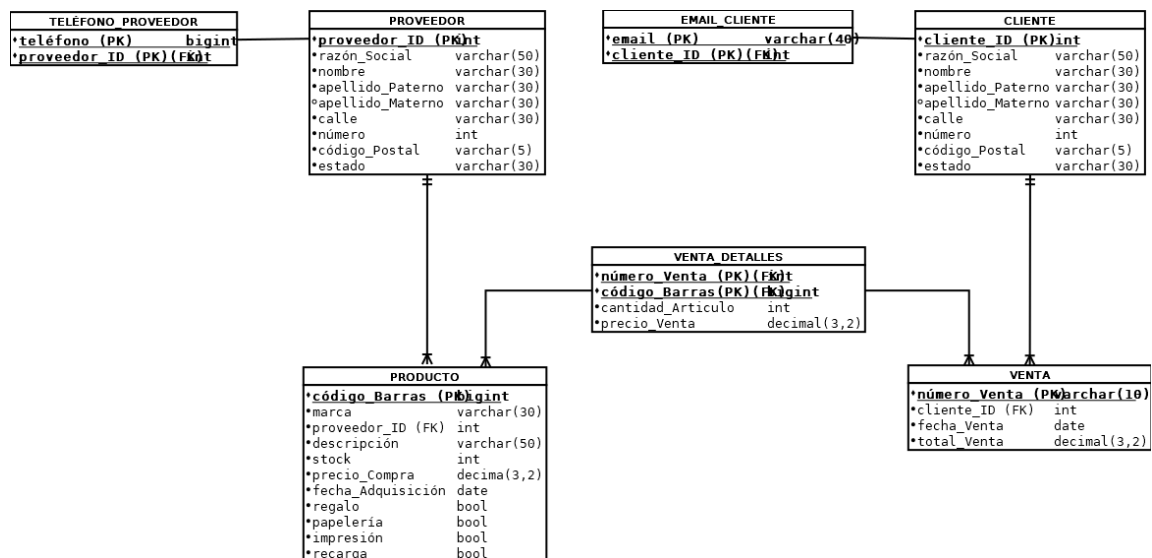
Las reglas de transformación fueron las siguientes:

- **PROVEEDOR:** proveedorID int (PK), razónSocial varchar(50), nombre varchar(30), apellidoPaterno varchar(30), apellidoMaterno (N) varchar(30), calle varchar(30), número int, códigoPostal varchar(5), estado varchar(30)
- **TELEFONO-PROVEEDOR:** (teléfono bigint, proveedorID (FK))[PK]
- **CLIENTE:** clienteID int (PK), razónSocial varchar(50), nombre varchar(30), apellidoPaterno varchar(30), apellidoMaterno (N) varchar(30), calle varchar(30), número int, códigoPostal varchar(5), estado varchar(30)
- **EMAIL-CLIENTE:** (email, clienteID (FK))[PK]
- **PRODUCTO:** códigoBarras bigint (PK), marca varchar(30), proveedorID int (FK), descripción varchar(50), stock int,

precioCompra decimal(4,2), fechaAdquisición date, regalo (N) bool, papelería (N) bool, impresión (N) bool, recarga (N) bool

- **VENTA:** númeroVenta varchar(10) (PK), clienteID int (FK), fechaVenta date, totalVenta decimal(4,2)
- **VENTA DETALLES:** [númeroVenta varchar(10) (FK), códigoBarras bigint (FK)](PK), cantidadArtículo int, precioVenta decimal(4,2)

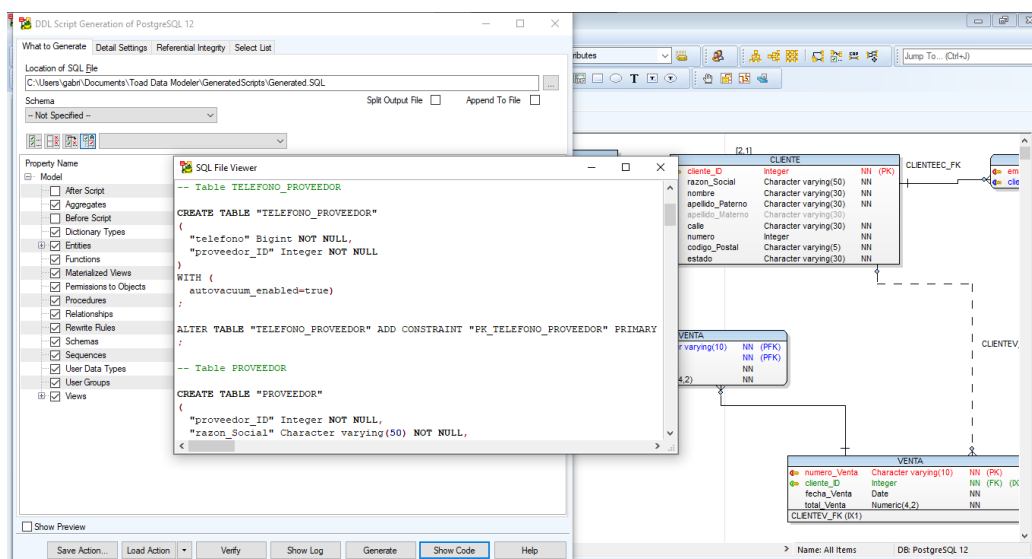
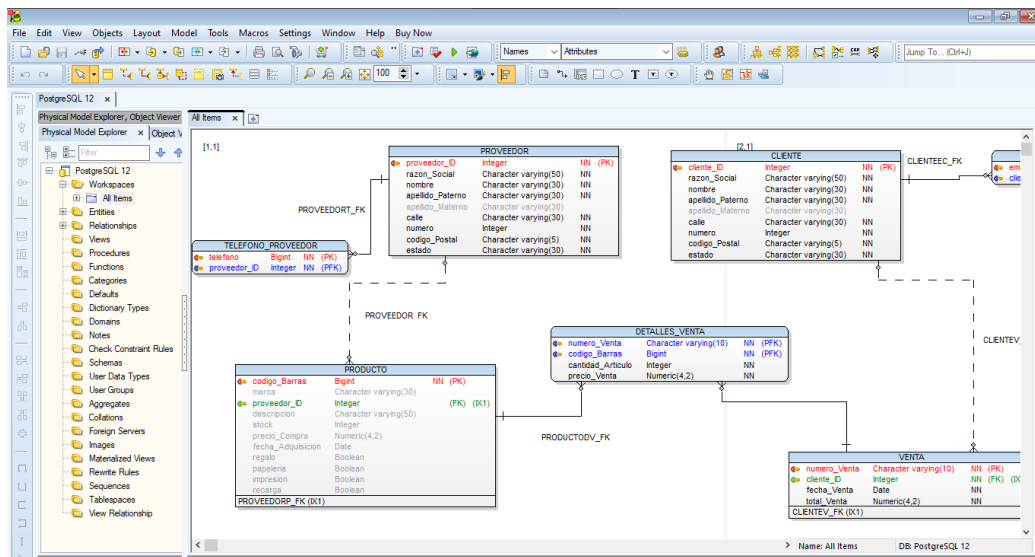
Una vez realizada la representación intermedia fue más simple poder hacer la conversión para el modelo relacional, en el cual se aplicó una de las técnicas aprendidas en el curso, la cual fue la especialización exclusiva, dando como resultado el siguiente modelo.



Finalmente, así quedo concluida la parte de diseño, y posteriormente mediante el modelo relacional, se pasó a la etapa de codificación e implementación.

IMPLEMENTACIÓN:

Para poder codificar adecuadamente la base de datos, se empleó un software de modelado de datos, cual es *Toad Data Modeler*, así mediante el modelo relacional obtenido, se extrapoló a este software, en donde se eligió el manejador de bases de datos y la versión, para así posteriormente dejar que el software realizará su trabajo.



Ya con el respectivo código SQL generado, se pasó a cargar ese script en el manejador de bases de datos POSTGRES, para así pasar a crear la base de datos, y posteriormente, empezar a cargar datos para así poder realizar las primeras pruebas correspondientes.

```
Database.sql x Funciones.sql x Triggers.sql x INDICES.sql x VIEWS.sql x Registros.sql x JOIN.sql x
1 -- Table TELEFONO_PROVEEDOR
2 CREATE TABLE TELEFONO_PROVEEDOR (
3     telefono Bigint NOT NULL,
4     proveedor_ID Integer NOT NULL
5 )
6 WITH (
7     autovacuum_enabled=true);
8
9 ALTER TABLE TELEFONO_PROVEEDOR ADD CONSTRAINT PK_TELEFONO_PROVEEDOR PRIMARY KEY (telefono,proveedor_ID);
10
11 -- Table PROVEEDOR
12 CREATE TABLE PROVEEDOR (
13     proveedor_ID Integer NOT NULL,
14     razon_Social Character varying(50) NOT NULL,
15     nombre Character varying(30) NOT NULL,
16     apellido_Paterno Character varying(30) NOT NULL,
17     apellido_Materno Character varying(30),
18     calle Character varying(30) NOT NULL,
19     numero Integer NOT NULL,
20     codigo_Postal Character varying(5) NOT NULL,
21     estado Character varying(30) NOT NULL
22 )
23 WITH (
24     autovacuum_enabled=true);
25
26 ALTER TABLE PROVEEDOR ADD CONSTRAINT PK_PROVEEDOR PRIMARY KEY (proveedor_ID);
27
28 -- Table CLIENTE
29 CREATE TABLE CLIENTE(
30     cliente_ID Integer NOT NULL,
31     razon_Social Character varying(50) NOT NULL,
32     nombre Character varying(30) NOT NULL,
33     apellido_Paterno Character varying(30) NOT NULL,
34     apellido_Materno Character varying(30),
35     calle Character varying(30) NOT NULL,
```

```
Database.sql x Funciones.sql x Triggers.sql x INDICES.sql x VIEWS.sql x Registros.sql x JOIN.sql x
36     numero Integer NOT NULL,
37     codigo_Postal Character varying(5) NOT NULL,
38     estado Character varying(30) NOT NULL
39 )
40 WITH (
41     autovacuum_enabled=true);
42
43 ALTER TABLE CLIENTE ADD CONSTRAINT PK_CLIENTE PRIMARY KEY (cliente_ID);
44
45 -- Table EMAIL_CLIENTE
46 CREATE TABLE EMAIL_CLIENTE (
47     email Character varying(40) NOT NULL,
48     cliente_ID Integer NOT NULL
49 )
50 WITH (
51     autovacuum_enabled=true);
52
53 ALTER TABLE EMAIL_CLIENTE ADD CONSTRAINT PK_EMAIL_CLIENTE PRIMARY KEY (email,cliente_ID);
54
55 -- Table VENTA
56 CREATE TABLE VENTA (
57     numero_Venta Character varying(10) NOT NULL,
58     cliente_ID Integer NOT NULL,
59     fecha_Venta Date NOT NULL,
60     total_Venta Numeric(6,2) NOT NULL
61 )
62 WITH (
63     autovacuum_enabled=true);
64
65 CREATE INDEX CLIENTEV_FK ON VENTA (cliente_ID);
66
67 ALTER TABLE VENTA ADD CONSTRAINT PK_VENTA PRIMARY KEY (numero_Venta);
68
69 -- Table DETALLES_VENTA
70 CREATE TABLE DETALLES_VENTA (
```



```

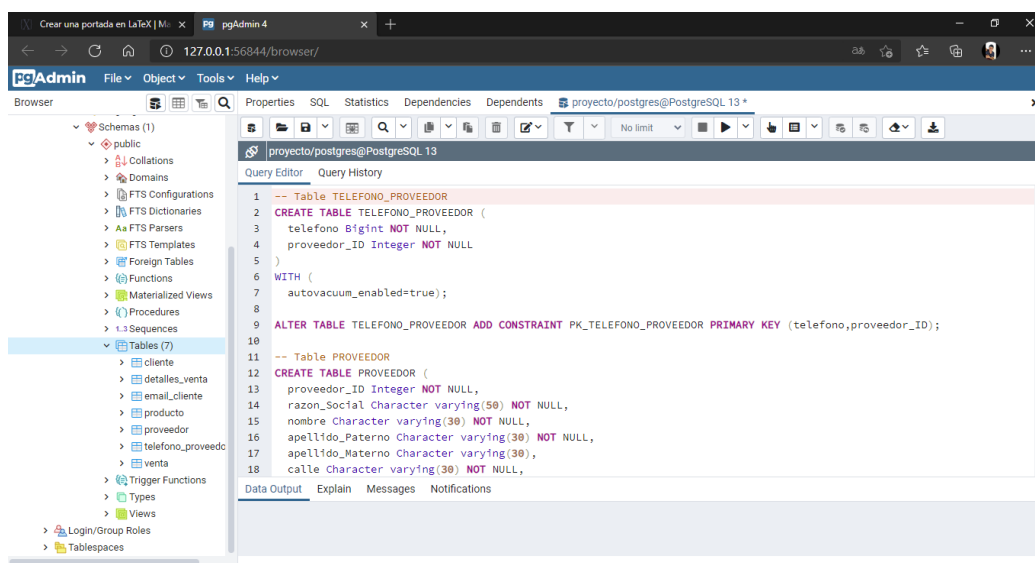
71 numero_venta Character varying(10) NOT NULL,
72 codigo_barras Bigint NOT NULL,
73 cantidad_articulo Integer NOT NULL,
74 precio_venta Numeric(6,2) NOT NULL
75 )
76 WITH (
77     autovacuum_enabled=true);
78
79 ALTER TABLE DETALLES_VENTA ADD CONSTRAINT PK_DETALLES_VENTA PRIMARY KEY (codigo_barras,numero_venta);
80
81 -- Table PRODUCTO
82 CREATE TABLE PRODUCTO (
83     codigo_barras Bigint NOT NULL,
84     marca Character varying(30) NOT NULL,
85     proveedor_id Integer NOT NULL,
86     descripcion Character varying(50) NOT NULL,
87     stock Integer NOT NULL,
88     precio_compra Numeric(6,2) NOT NULL,
89     fecha_adquisicion Date NOT NULL,
90     regalo Boolean,
91     papeleria Boolean,
92     impresion Boolean,
93     recarga Boolean
94 )
95 WITH (
96     autovacuum_enabled=true);
97
98 CREATE INDEX PROVEEDORP_FK ON PRODUCTO (proveedor_id);
99
100 ALTER TABLE PRODUCTO ADD CONSTRAINT PK_PRODUCTO PRIMARY KEY (codigo_barras);
101
102 -- Create foreign keys (relationships) section -----
103
104 ALTER TABLE TELEFONO_PROVEEDOR ADD CONSTRAINT PROVEEDORT_FK
105 FOREIGN KEY (proveedor_id) REFERENCES PROVEEDOR (proveedor_id)

```

```

106 ON DELETE CASCADE ON UPDATE CASCADE;
107
108 ALTER TABLE PRODUCTO ADD CONSTRAINT PROVEEDOR_FK
109 FOREIGN KEY (proveedor_id) REFERENCES PROVEEDOR (proveedor_id)
110 ON DELETE CASCADE ON UPDATE CASCADE;
111
112 ALTER TABLE DETALLES_VENTA ADD CONSTRAINT PRODUCTOV_FK
113 FOREIGN KEY (codigo_barras) REFERENCES PRODUCTO (codigo_barras)
114 ON DELETE CASCADE ON UPDATE CASCADE;
115
116 ALTER TABLE DETALLES_VENTA ADD CONSTRAINT VENTADV_FK
117 FOREIGN KEY (numero_venta) REFERENCES VENTA (numero_venta)
118 ON DELETE CASCADE ON UPDATE CASCADE;
119
120 ALTER TABLE EMAIL_CLIENTE ADD CONSTRAINT CLIENTEEC_FK
121 FOREIGN KEY (cliente_id) REFERENCES CLIENTE (cliente_id)
122 ON DELETE CASCADE ON UPDATE CASCADE;
123
124 ALTER TABLE VENTA ADD CONSTRAINT CLIENTEV_FK
125 FOREIGN KEY (cliente_id) REFERENCES CLIENTE (cliente_id)
126 ON DELETE CASCADE ON UPDATE CASCADE;

```



The screenshot shows the pgAdmin 4 web interface. On the left, the 'public' schema is expanded, showing a list of tables: cliente, detalles_venta, email_cliente, producto, proveedor, telefono_proveedor, and venta. The 'telefono_proveedor' table is selected. On the right, the 'Query Editor' is open, displaying the SQL code for creating the 'telefono_proveedor' and 'proveedor' tables. The code includes primary key constraints and foreign key relationships.

Con la base creada , se comenzó a realizar la carga de datos, para así también poder comprobar que cumplía con la estructura requerida y además de que la relación que había entre las distintas tablas contenidas fuera adecuada.

```

4  INSERT INTO PROVEEDOR
5  VALUES(2, 'DIXON', 'Katherine', 'Ibarra', 'Mata', 'Guanabana', 318, '02820', 'Ciudad de México');
6  INSERT INTO PROVEEDOR
7  VALUES(3, 'CASIO', 'Angel', 'Martinez', 'Gutierrez', 'Av. Insurgentes', 1457, '03840', 'Ciudad de México');
8  INSERT INTO PROVEEDOR
9  VALUES(4, 'STEREN', 'Alheli', 'Mejia', 'Angel', 'Republica del Salvador', 35, '06000', 'Ciudad de México');
10 INSERT INTO PROVEEDOR
11 VALUES(5, 'TELCEL', 'Marco', 'Guerra', 'Arce', 'Calle 6', 110, '08100', 'Estado de México');
12 INSERT INTO PROVEEDOR
13 VALUES(6, 'HP', 'Ricardo', 'Alvarado', 'Rodriguez', 'Av. Canal de Tezontle', 1512, '09020', 'Ciudad de México');
14 INSERT INTO PROVEEDOR
15 VALUES(7, 'Scribe', 'Andrea', 'Mejia', 'Godinez', 'Av. Ejercito Nacional', 559, '11520', 'Ciudad de México');
16 INSERT INTO PROVEEDOR
17 VALUES(8, 'Nike', 'Jonathan', 'Boni', 'Jimenez', 'Av. Rio san Joaquin', 406, '11529', 'Ciudad de México');
18 INSERT INTO PROVEEDOR
19 VALUES(9, 'Morne', 'Javier', 'Blas', 'Perales', 'Lago Chalco', 81, '11320', 'Ciudad de México');
20 INSERT INTO PROVEEDOR
21 VALUES(10, 'Papel S.A. de C.V.', 'Alexis', 'Mayoral', 'Montejano', 'Simon Bolivar', 95, '06080', 'Ciudad de México');
22
23 --TABLA TELEFONO_PROVEEDOR--
24 INSERT INTO TELEFONO_PROVEEDOR
25 VALUES(5571382456, 1);
26 INSERT INTO TELEFONO_PROVEEDOR
27 VALUES(5513554430, 2);
28 INSERT INTO TELEFONO_PROVEEDOR
29 VALUES(5582067875, 3);
30 INSERT INTO TELEFONO_PROVEEDOR
31 VALUES(5514617194, 4);
32 INSERT INTO TELEFONO_PROVEEDOR
33 VALUES(5588147350, 5);
34 INSERT INTO TELEFONO_PROVEEDOR
35 VALUES(5539597982, 6);
36 INSERT INTO TELEFONO_PROVEEDOR
37 VALUES(5561844905, 7);
38 INSERT INTO TELEFONO_PROVEEDOR

```

pgAdmin 4

127.0.0.1:56844/browser/

pgAdmin File Object Tools Help

Browser

- Schemas (1)
 - public
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Procedures
 - Sequences
 - Tables (7)
 - cliente
 - detalles_venta
 - email_cliente
 - producto
 - proveedor
 - telefono_proveedor
 - venta
 - Trigger Functions
 - Types
 - Views
 - Login/Group Roles
 - Tablespaces

Properties SQL Statistics Dependencies Dependents

projecto/postgres@PostgreSQL 13 *

Query Editor Query History

```

1  SELECT *
2  FROM PRODUCTO;
3
4  SELECT *
5  FROM CLIENTE;
6
7  SELECT *
8  FROM PROVEEDOR;

```

	codigo_barras [PK] bigint	marca character varying (30)	proveedor_id integer	descripcion character varying (50)	stock integer	precio_compra numeric (6,2)	fecha_adquisicion date	regalo boolean
1	650240032974	HP		Paquete de 500 hojas blanc...	25	75.00	2020-01-23	[null]
2	7501065076625	TELCEL		Recarga de tiempo aire \$500	5	489.99	2020-01-23	[null]
3	40058082993551	STEREN		Paquete de LEDs	20	15.30	2020-01-23	[null]
4	7503006503016	DIXON		Lapiz HB para dibujo	50	2.50	2020-01-23	[null]
5	6920354822230	AFELPADOS		Peluche de felpa		1.00	2020-01-23	[null]
6	7502016305200	Scribe		Cuaderno profesional de				

Successfully run. Total query runtime: 263 msec. 20 rows affected.

Otra prueba que se realizó para checar que las relaciones fueran adecuadas,

fueron el uso de JOIN para así poder extraer información de todas las tablas relacionadas, lo cual arrojó un resultado positivo y así continuar con la siguiente parte de desarrollo del proyecto.

The screenshot shows the pgAdmin interface with a SQL query executed. The query is as follows:

```

1 SELECT PR.RAZON_SOCIAL, PR.NOMBRE, PR.APELLIDO_PATERNO, TP.TELEFONO
2 FROM PROVEEDOR AS PR
3 INNER JOIN TELEFONO_PROVEEDOR AS TP
4 ON PR.PROVEEDOR_ID = TP.PROVEEDOR_ID;
5
6 SELECT CL.NOMBRE, CL.APELLIDO_PATERNO, EC.EMAIL
7 FROM CLIENTE AS CL
8 INNER JOIN EMAIL_CLIENTE AS EC
9 ON CL.CLIENTE_ID = EC.CLIENTE_ID
10
11 SELECT PR.CODIGO_BARRAS, PRO.RAZON_SOCIAL, PR.DESCRIPCION
12 FROM PRODUCTO AS PR

```

The results are displayed in a table with the following columns: `razon_social`, `nombre`, `apellido_paterno`, and `telefono`. The data rows are:

razon_social	nombre	apellido_paterno	telefono
1 Ostitos Maravilla	Edwin	Santiago	5571382450
2 DIXON	Katherine	Ibarra	5513554430
3 CASIO	Angel	Martinez	5582067875
4 STEREN	Alheli	Mejia	5514617194
5 TELCEL	Marco	Guerra	5588147358
6 HP	Ricardo	Alvarado	5539597982

The screenshot shows the pgAdmin interface with a SQL query executed. The query is as follows:

```

1 SELECT PR.RAZON_SOCIAL, PR.NOMBRE, PR.APELLIDO_PATERNO, TP.TELEFONO
2 FROM PROVEEDOR AS PR
3 INNER JOIN TELEFONO_PROVEEDOR AS TP
4 ON PR.PROVEEDOR_ID = TP.PROVEEDOR_ID;
5
6 SELECT CL.NOMBRE, CL.APELLIDO_PATERNO, EC.EMAIL
7 FROM CLIENTE AS CL
8 INNER JOIN EMAIL_CLIENTE AS EC
9 ON CL.CLIENTE_ID = EC.CLIENTE_ID
10
11 SELECT PR.CODIGO_BARRAS, PRO.RAZON_SOCIAL, PR.DESCRIPCION
12 FROM PRODUCTO AS PR

```

The results are displayed in a table with the following columns: `nombre`, `apellido_paterno`, and `email`. The data rows are:

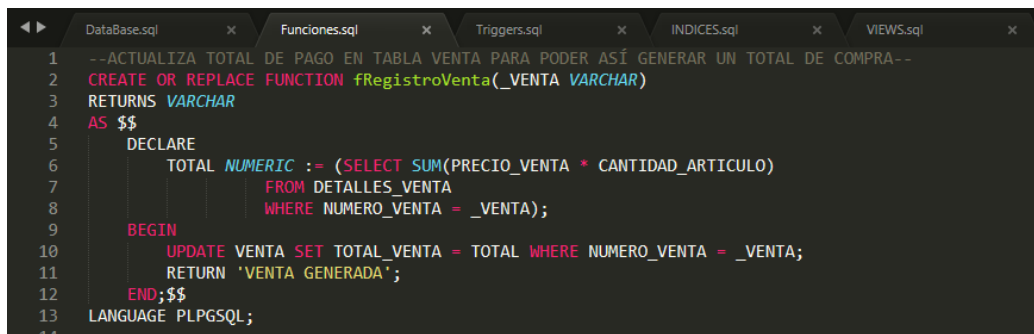
nombre	apellido_paterno	email
1 Daniel	Rojas	danyrojas@gmail.com
2 Gabriel	Rojas	gabrielrojasmedez@gmail.com
3 Carmen	Méndez	carmen610807@gmail.com
4 Lourdes	Rojas	emiline0190@gmail.com
5 Rauli	Rojas	raulrojas1166@gmail.com
6 Jose Julian	Rojas	julianTR@gmail.com

A green message box at the bottom right states: "Successfully run. Total query runtime: 119 msec. 10 rows affected."

El proyecto tenía requerimientos específicos, los cuales constaban de crear algunas funciones, procedimientos, triggers e incluso vistas, de esta manera se ponía en práctica los temas sobre estas funciones de agregación y el lenguaje PLPGSQL que nos ofrece POSTGRES.

A continuación se mostrará las distintas funciones creadas para así satisfacer las necesidades de dichos requerimientos.

- **Función fRegistroVenta:** esta función permite actualizar el campo *totalVenta*, pues cada que se realiza una compra de un producto el costo de estos se va sumando y así se almacena en dicho campo.



```
1  --ACTUALIZA TOTAL DE PAGO EN TABLA VENTA PARA PODER ASÍ GENERAR UN TOTAL DE COMPRA--
2  CREATE OR REPLACE FUNCTION fRegistroVenta(_VENTA VARCHAR)
3  RETURNS VARCHAR
4  AS $$
5      DECLARE
6          TOTAL NUMERIC := (SELECT SUM(PRECIO_VENTA * CANTIDAD_ARTICULO)
7                               FROM DETALLES_VENTA
8                               WHERE NUMERO_VENTA = _VENTA);
9      BEGIN
10         UPDATE VENTA SET TOTAL_VENTA = TOTAL WHERE NUMERO_VENTA = _VENTA;
11         RETURN 'VENTA GENERADA';
12     END;$$
13 LANGUAGE PLPGSQL;
```

- **fGeneradorFactura:** Esta función permite que a partir de cada número de venta se pueda mostrar una información que se asemeja a los datos contenidos en una factura real.

```

15 --FUNCIÓN QUE GENERA UNA VISTA LA CUAL SE ASEMEJA A UNA FACTURA--
16 CREATE OR REPLACE FUNCTION fGeradorFactura(_VENTA VARCHAR)
17 RETURNS TABLE(VENTA CHARACTER VARYING(10),
18               CLIENTE INTEGER,
19               NOMBRE CHARACTER VARYING(30),
20               CANTIDAD INTEGER,
21               DESCRIPCION CHARACTER VARYING(50),
22               PRECIO NUMERIC(6,2),
23               TOTAL NUMERIC(6,2))
24 AS $$
25 BEGIN
26     RETURN QUERY
27     SELECT VE.NUMERO_VENTA, CL.CLIENTE_ID, CL.NOMBRE,
28            DV.CANTIDAD_ARTICULO AS CANTIDAD,
29            PR.DESCRIPCION, DV.PRECIO_VENTA AS PRECIO,
30            (DV.CANTIDAD_ARTICULO * DV.PRECIO_VENTA) AS TOTAL_PARCIAL
31     FROM DETALLES_VENTA AS DV
32     INNER JOIN PRODUCTO AS PR
33     ON DV.CODIGO_BARRAS = PR.CODIGO_BARRAS
34     INNER JOIN VENTA AS VE
35     ON DV.NUMERO_VENTA = VE.NUMERO_VENTA
36     INNER JOIN CLIENTE AS CL
37     ON VE.CLIENTE_ID = CL.CLIENTE_ID
38     WHERE VE.NUMERO_VENTA = _VENTA;
39 END; $$
40 LANGUAGE PLPGSQL;

```

- **fUTILIDADES:** Esta otra función permite calcular las utilidades que un producto ha dejado al negocio.

```

42 --FUNCIÓN QUE REGRESA UTILIDAD DE UN PRODUCTO--
43 CREATE OR REPLACE FUNCTION fUTILIDADES(_CODIGO BIGINT)
44 RETURNS TABLE(CODIGO BIGINT,
45               DESCRIPCION CHARACTER VARYING(50),
46               UTILIDAD NUMERIC(6,2))
47 AS $$
48 BEGIN
49     RETURN QUERY
50     SELECT PR.CODIGO_BARRAS, PR.DESCRIPCION,
51            SUM(DV.CANTIDAD_ARTICULO * DV.PRECIO_VENTA) - (SUM(DV.CANTIDAD_ARTICULO) * PR.PRECIO_COMPRA)
52     FROM DETALLES_VENTA AS DV
53     INNER JOIN PRODUCTO AS PR
54     ON DV.CODIGO_BARRAS = PR.CODIGO_BARRAS
55     WHERE DV.CODIGO_BARRAS = _CODIGO
56     GROUP BY PR.CODIGO_BARRAS;
57 END; $$
58 LANGUAGE PLPGSQL;

```

- **fMENOSDETRES:** Función que permite obtener el listado de los productos que se encuentran con un stock menor a 3 piezas.

```

62 --FUNCIÓN QUE RETORNA AQUELLOS PRODUCTOS CON EXISTENCIAS MENORES A 3--
63 CREATE OR REPLACE FUNCTION fMENOSDETRES()
64 RETURNS TABLE(CODIGO BIGINT,
65               DESCRIP CHARACTER VARYING(50),
66               EXISTENCIAS INTEGER)
67 AS $$
68 BEGIN
69     RETURN QUERY
70     SELECT CODIGO_BARRAS, DESCRIPCION, STOCK
71     FROM PRODUCTO
72     WHERE STOCK < 3;
73 END; $$
74 LANGUAGE PLPGSQL;
75

```

- **fGANANCIASPERIODO**: Para esta función se aplicó una sobrecarga de parametros, ya que la función con un sólo parámetro calcula, la venta total que hubo en cierta fecha, y la función con dos parámetros permite obtener el total de ventas de un período.

```

78 --FUNCIÓN QUE RETORNA LA VENTA TOTAL DE UNA FECHA O UN PERIODO EN DONDE SE EMPLEÓ--
79 --LA SOBRECARGA DE FUNCIONES, LA PRIMER FUNCIÓN ES PARA UN FECHA EN ESPECÍFICO Y--
80 --LA SEGUNDA PARA UN PERIODO --
81 CREATE OR REPLACE FUNCTION fGANANCIASPERIODO(FECHA1 DATE)
82 RETURNS NUMERIC
83 AS $$
84     DECLARE GANANCIA NUMERIC:= (SELECT SUM(TOTAL_VENTA)
85                                FROM VENTA
86                                WHERE FECHA_VENTA = FECHA1);
87 BEGIN
88     RETURN GANANCIA;
89 END; $$
90 LANGUAGE PLPGSQL;
91
92 CREATE OR REPLACE FUNCTION fGANANCIASPERIODO(FECHA1 DATE, FECHA2 DATE)
93 RETURNS NUMERIC
94 AS $$
95     DECLARE GANANCIA NUMERIC:= (SELECT SUM(TOTAL_VENTA)
96                                FROM VENTA
97                                WHERE FECHA_VENTA
98                                BETWEEN FECHA1 AND FECHA2);
99 BEGIN
100    RETURN GANANCIA;
101 END; $$
102 LANGUAGE PLPGSQL;
103

```

- **DecrementoStock**: Esta función es más un procedimiento almacenado, el cual será ejecutado mediante un trigger que se ha implementado en la tabla *DETALLESVENTA* para así poder ir decrementando el stock de los productos que se vendan, y si se llegará al caso de tener un stock en cero, terminar la transacción y evitar esa venta.

```

107 --FUNCIÓN QUE PERMITE LA VALIDACIÓN DEL STOCK Y TAMBIÉN SU DECREMENTO--
108 --ADEMÁS DE QUE ES PARTE DEL TRIGGER QUE DESENCADENA LA ACCIÓN--
109 CREATE OR REPLACE FUNCTION DecrementoStock()
110 RETURNS TRIGGER
111 AS $$
112 DECLARE CANTIDAD INTEGER;
113 BEGIN
114     CANTIDAD = (SELECT STOCK FROM PRODUCTO WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS);
115     IF (CANTIDAD > 3) THEN
116         UPDATE PRODUCTO SET STOCK = (STOCK - NEW.CANTIDAD_ARTICULO)
117         WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS;
118     ELSEIF (CANTIDAD = 0) THEN
119         RAISE NOTICE 'PRODUCTO AGOTADO EN EXISTENCIAS';
120         ROLLBACK;
121     ELSE
122         UPDATE PRODUCTO SET STOCK = (STOCK - NEW.CANTIDAD_ARTICULO)
123         WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS;
124         RAISE NOTICE 'PRODUCTO PROXIMO A AGOTARSE';
125     END IF;
126     RETURN NEW;
127 END; $$
128 LANGUAGE PLPGSQL;

```

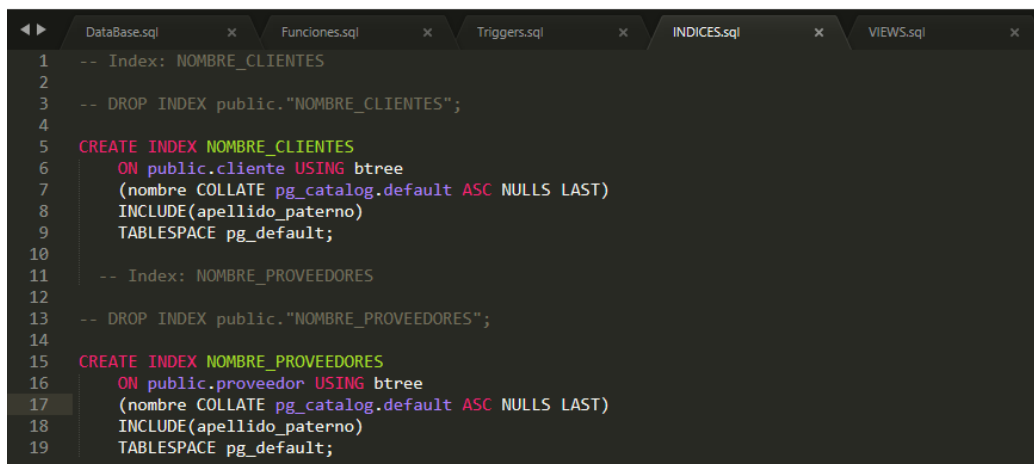
También se adjunta el trigger que ejecutará de manera automática la función anterior.

```

1 CREATE TRIGGER DECREMENTO_STOCK
2 BEFORE INSERT ON DETALLES_VENTA
3 FOR EACH ROW
4 EXECUTE PROCEDURE DecrementoStock();

```

Por último, el proyecto requería de la creación e implementación de índices, por lo tanto se pasó a crear índices en las tablas de cliente y proveedor, viendo a un futuro que los registros de estas crezcan de manera tal que el manejador tarde en poder encontrar un conjunto o un sólo registro. Cabe mencionar que los índices fueron creados sobre los campos de nombre y apellido paterno de ambas relaciones, también se empleó un tipo de búsqueda b-tree para así poder hacer que el manejador se encuentre optimizado en cuestión de manejar grandes cantidades de registros dentro de las tablas ya mencionadas.



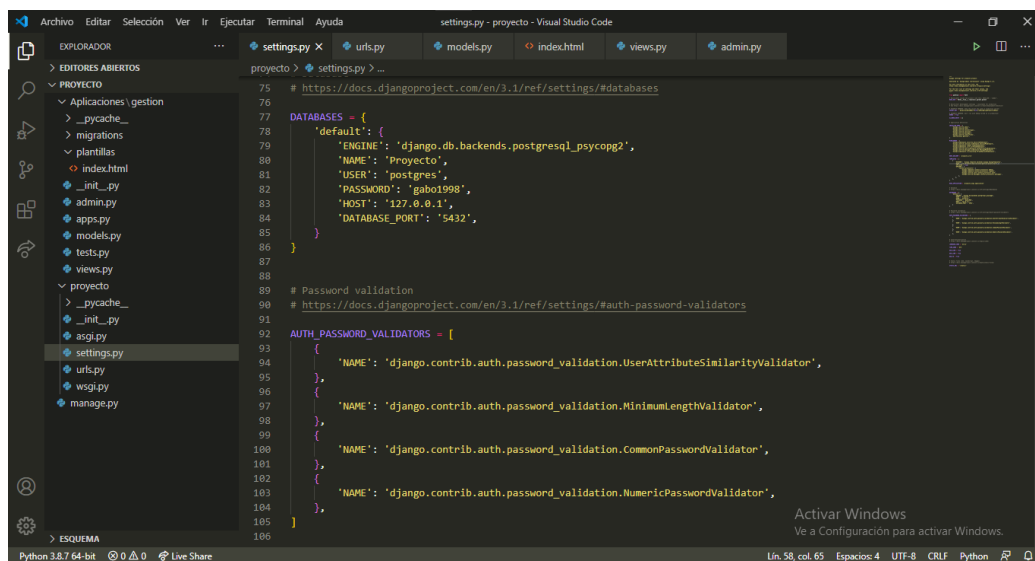
```
1  -- Index: NOMBRE_CLIENTES
2
3  -- DROP INDEX public."NOMBRE_CLIENTES";
4
5  CREATE INDEX NOMBRE_CLIENTES
6      ON public.cliente USING btree
7      (nombre COLLATE pg_catalog.default ASC NULLS LAST)
8      INCLUDE(apellido_paterno)
9      TABLESPACE pg_default;
10
11  -- Index: NOMBRE_PROVEEDORES
12
13  -- DROP INDEX public."NOMBRE_PROVEEDORES";
14
15  CREATE INDEX NOMBRE_PROVEEDORES
16      ON public.proveedor USING btree
17      (nombre COLLATE pg_catalog.default ASC NULLS LAST)
18      INCLUDE(apellido_paterno)
19      TABLESPACE pg_default;
```


PRESENTACIÓN:

La segunda parte del proyecto consistía en realizar una interfaz gráfica para poder tener control sobre las insercciones de la base de datos. Para fines más didácticos se decidió como equipo emplear el framework Django, el cual brinda una gran facilidad de manejo de bases de datos, además de que se hace uso de buenas técnicas de desarrollo de proyectos.

Hay bque mencionar que para poder trabajar en esta parte del proyecto desde dicho framework se debía tener una gran habilidad con el paradigma orientado a objetos, pues la creación de la base de datos, no se hacía desde el manejador, sino más bien desde el framework, y creando migraciones en donde, Django se encargaba de realizar estas operaciones de compatibilidad. Pero este proceso se irá explicando poco a poco, y se da inicio con la configuración del servidor para poder tener una conexión con el manejador POSTGRES, pues esta herramienta cuenta con configuraciones por default de SQLite.

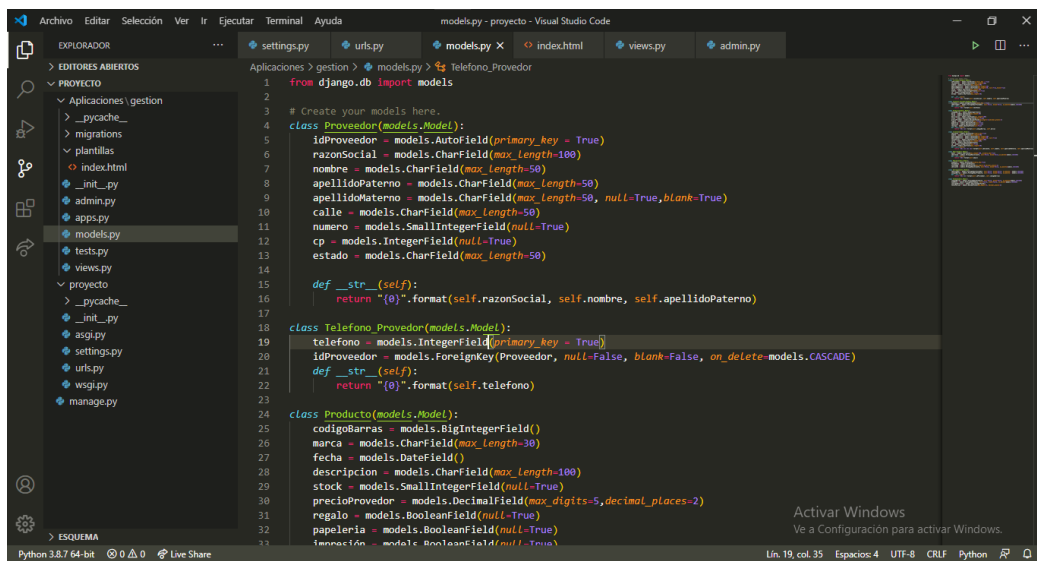
La siguiente imagen muestra esta parte de configuración.



```
75 # https://docs.djangoproject.com/en/3.1/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.postgresql_psycopg2',
80         'NAME': 'proyecto',
81         'USER': 'postgres',
82         'PASSWORD': 'gabo1998',
83         'HOST': '127.0.0.1',
84         'DATABASE_PORT': '5432',
85     }
86 }
87
88
89 # Password validation
90 # https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators
91
92 AUTH_PASSWORD_VALIDATORS = [
93     {
94         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
95     },
96     {
97         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
98     },
99     {
100         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
101     },
102     {
103         'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
104     },
105 ]
106
```

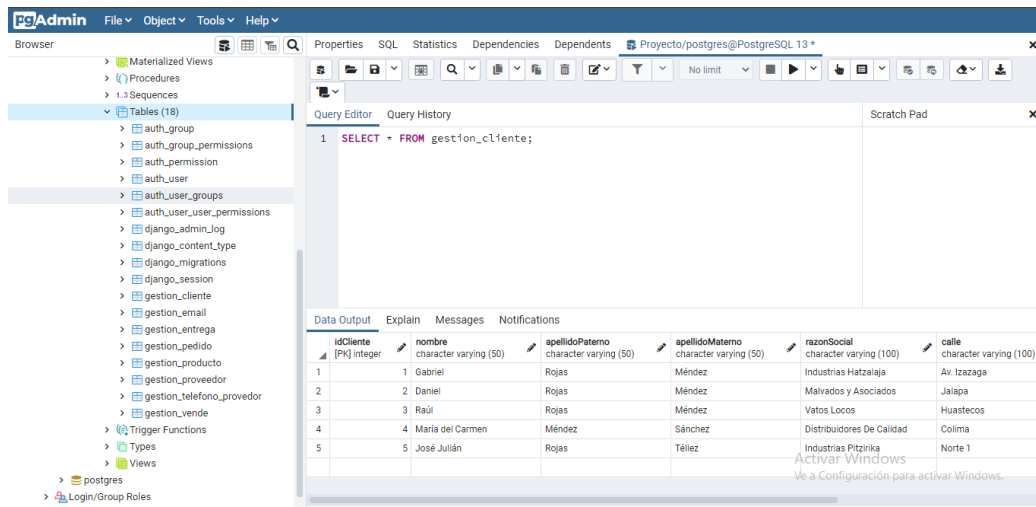
Antes se debía descargar una librería de POSTGRES a Django para así después poder modificar los datos de enlace entre el servidor de Django y el manejador.

Una vez realizadas estas configuraciones, se debía ir a modelar las relaciones que debía contener la base de datos, y esto se hacía mediante el paradigma de programación orientada a objetos. De esta manera era como primero se debía modelar las entidades fuertes, aquellas que no dependieran de alguna otra entidad y así posteriormente las entidades que contaran con claves foraneas de estas primeras tablas modeladas.

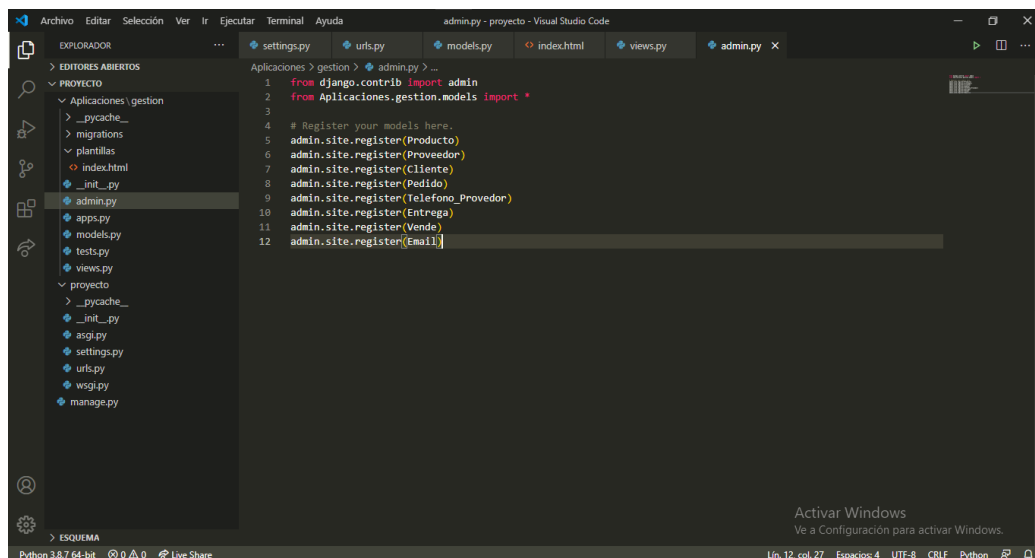


```
1 from django.db import models
2
3 # Create your models here.
4 class Proveedor(models.Model):
5     idProveedor = models.AutoField(primary_key = True)
6     razonSocial = models.CharField(max_length=100)
7     nombre = models.CharField(max_length=50)
8     apellidoPaterno = models.CharField(max_length=50)
9     apellidoMaterno = models.CharField(max_length=50, null=True, blank=True)
10    calle = models.CharField(max_length=50)
11    numero = models.SmallIntegerField(null=True)
12    cp = models.IntegerField(null=True)
13    estado = models.CharField(max_length=50)
14
15    def __str__(self):
16        return "{}".format(self.razonSocial, self.nombre, self.apellidoPaterno)
17
18 class Telefono_Proveedor(models.Model):
19     telefono = models.IntegerField(primary_key = True)
20     idProveedor = models.ForeignKey(Proveedor, null=False, blank=False, on_delete=models.CASCADE)
21
22     def __str__(self):
23         return "{}".format(self.telefono)
24
25 class Producto(models.Model):
26     codigoBarras = models.BigIntegerField()
27     marca = models.CharField(max_length=30)
28     fecha = models.DateField()
29     descripcion = models.CharField(max_length=100)
30     stock = models.SmallIntegerField(null=True)
31     precioProveedor = models.DecimalField(max_digits=5, decimal_places=2)
32     regalo = models.BooleanField(null=True)
33     papeleria = models.BooleanField(null=True)
34     innoxiacion = models.BooleanField(null=True)
```

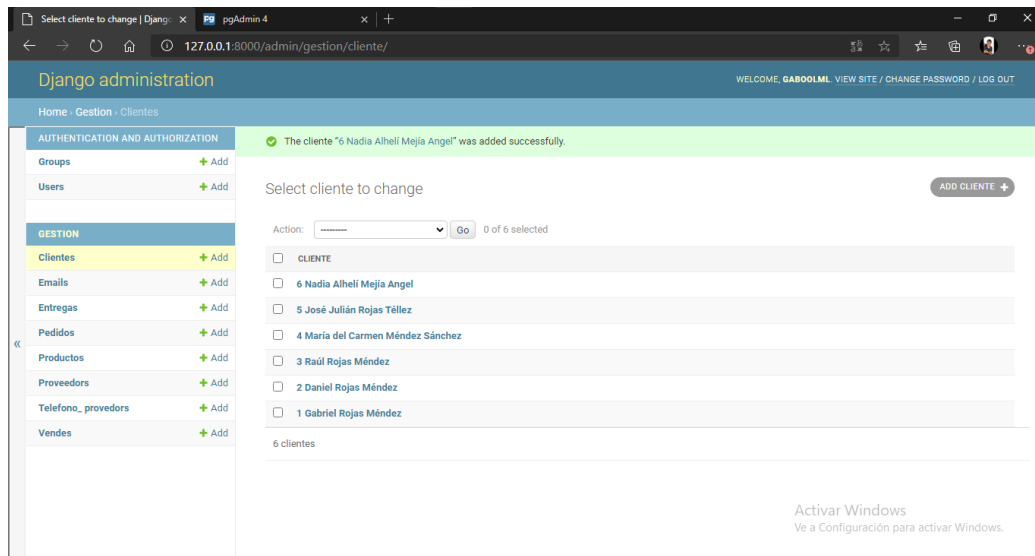
Ya con los modelos hechos, se tenían que realizar las migraciones correspondientes para que estas tablas se crearán en POSTGRES.



Una vez que las tablas estaban creadas se debían configurar el panel de control de Django para así poder desde aquí administrar la base de datos, aprovechando que por defecto este framework ya te deja usar su interfaz para poder tener visualmente el comportamiento de la base.



19



Y así finalmente la interfaz proporcionada fue tan amigable para poder realizar las inserciones necesarias en las tablas creadas, cumpliendo con el objetivo de ser amigable para realizar estas operaciones.

CONSLUSIONES:

- **Rojas Méndez Gabriel:** el desarrollo de este proyecto me permitió poder implementar al máximo los conocimientos adquiridos durante el semestre, poniéndome como reto el diseñar una base de datos, configurarla y hacerla funcional. Posteriormente el verdadero reto vino en crear una interfaz gráfica para poder así administrar dicha base, sin embargo el conocer herramientas de desarrollo de software permitió que además de explorar este entorno de trabajo, se adquiriera experiencia, pues Django es hoy en día una de las herramientas más empleadas para el desarrollo de apps que requieran de un manejo de información.

Por último hay que mencionar que si la situación actual hubiera sido distinta, este proyecto habría tenido una mayor calidad y funcionalidad, pues el haber podido desarrollar una página web me habría dejado un sentimiento de mayor satisfacción, sin embargo quedo conforme con el desarrollo de este proyecto.

- **Loperena Alcántara Marisol Viridiana:** Este proyecto ha sido un trabajo muy completo, no solo aplicamos lo visto en clases de teoría sino que se complementó con cada una de las prácticas realizadas en laboratorio.

También de manera personal este proyecto ha tenido muchas dificultades como el iniciar con 5 compañeros, pero finalmente entre los dos resultó un apoyo mutuo.

Otra dificultad fue no haber estado al 100% durante el proyecto debido a situación de enfermedad y otro caso personal, pero el apoyo brindado por Gabriel fue lo que sostuvo nuestro proyecto.

Sin duda la parte de conexión cliente servidor, ha sido más complicado debido a que aún no hemos tomado clases de estos temas y fue la parte a la que más se le dedicó tiempo.