

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: CREANDO SENTENCIAS DDL.
- EXERCISE 2: PASANDO DE UN MODELO RACIONAL A UNO FISICO.
- EXERCISE 3: CREANDO SENTENCIAS DML

EXERCISE 1: CREANDO SENTENCIAS DDL

SENTENCIA DROP

La orden **DROP TABLE** seguida del nombre de una tabla, permite eliminar la tabla en cuestión.

Al borrar una tabla:

- Desaparecen todos los datos.
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará (por lo que conviene eliminarlos).
- Las transacciones pendientes son aceptadas (**COMMIT**), en aquellas bases de datos que tengan la posibilidad de utilizarlas.
- Lógicamente, solo se pueden eliminar las tablas sobre las que tenemos permiso de borrado.
- Normalmente, el borrado de una tabla es irreversible, y no hay ninguna petición de confirmación, por lo que es importante ser muy cuidadoso con esta operación.

SENTENCIA ALTER

Cambiar de nombre a una tabla. De forma estándar (SQL estándar) se realiza la acción:

```
1 ALTER TABLE nombreViejo RENAME TO nombreNuevo;
```

Además, la orden anterior, se realiza mediante la orden **RENAME** (que permite el cambio de nombre de cualquier objeto). Su sintaxis es:

```
1 RENAME nombreViejo TO nombreNuevo;
```

AÑADIR COLUMNAS

```
1 ALTER USER nombreTabla ADD (nombreColumna TipoDatos [Propiedades],  
2 columnaSiguiente tipoDatos [propiedades]...)
```

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos, y sus propiedades si es necesario (al estilo de **CREATE TABLE**).

Ejemplo:

```
1 ALTER TABLE facturas ADD (fecha DATE);
```

Muchas bases de datos (no Postgres) requieren escribir la palabra **COLUMN**, tras la palabra **ADD**.

BORRAR COLUMNAS

```
1 ALTER USER nombreTabla DROP (columna[,columnaSiguiente,... ]);
```

Elimina la columna indicada de manera irreversible. No se puede eliminar una columna si es la única que queda en la tabla. En ese caso, se tiene que usar el comando **DROP TABLE**.

Ejemplo de borrado de columna:

```
1 ALTER TABLE facturas DROP (fecha);
```

MODIFICAR COLUMNAS

Permite cambiar el tipo de datos y propiedades de una determinada columna. Su sintaxis es:

```
1 ALTER TABLE nombreTabla ALTER COLUMN nombre_columna TYPE tipoDato;  
2
```

RENOMBRAR COLUMNA

Esto permite cambiar el nombre de una columna. Su sintaxis es:

```
1 ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo
```

Ejemplo:

```
1 ALTER TABLE facturas RENAME COLUMN fecha TO fechaYhora;
```

VALOR POR DEFECTO

A cada columna se le puede asignar un valor por defecto durante su creación, mediante la propiedad **DEFAULT**. Ésta se usará durante la creación o modificación de la tabla, añadiendo la palabra **DEFAULT** tras el tipo de datos del campo, y colocando detrás el valor que se desea por defecto.

Ejemplo:

```
1 CREATE TABLE articulo (cod NUMBER(7), nombre VARCHAR2(25), precio  
NUMBER(11,2) DEFAULT 3.5);
```

La palabra **DEFAULT** se puede añadir durante la creación o modificación de la tabla (comando **ALTER TABLE**). El valor indicado con **DEFAULT** se aplica cuando añadimos filas a una tabla, dejando el valor de la columna vacío en lugar de **NULL**; a la columna se le asignará el valor por defecto indicado.

RESTRICCIONES DE VALIDACIÓN

Son aquellas que dictan una condición que deben cumplir los contenidos de una columna. Una misma columna puede tener múltiples **CHECKS** en su definición, en este caso, se pondrían varios **CONSTRAINT** seguidos, sin comas.

Ejemplo:

```
1 CREATE TABLE ingresos(  
2   cod NUMBER(5) PRIMARY KEY,  
3   concepto VARCHAR2(40) NOT NULL,  
4   importe NUMBER(11,2) CONSTRAINT ingresos_ck1 CHECK (importe>0)  
5   CONSTRAINT ingresos_ck2 CHECK (importe<8000)  
6 );
```

En este caso, las restricciones **CHECK** prohíben añadir datos cuyo importe no esté entre 0 y 8000.

Aunque sería más cómodo de esta forma:

```
1 CREATE TABLE ingresos(  
2   cod NUMBER(5) PRIMARY KEY,  
3   concepto VARCHAR2(40) NOT NULL,  
4   importe NUMBER(11,2) CONSTRAINT ingresos_ck1  
5   CHECK (importe>0 AND importe<8000)  
6 );
```

AÑADIR RESTRICCIONES A UNA TABLA

Es posible querer añadir restricciones tras haber creado la tabla. En ese caso, se utiliza la siguiente sintaxis:

```
1 ALTER TABLE tabla ADD [CONSTRAINT nombre] tipoDeRestriccion(columnas);
```

tipoDeRestriccion es el texto CHECK, PRIMARY KEY, UNIQUE o FOREIGN KEY

Si deseamos añadir una restricción **NOT NULL**, se realiza mediante **ALTER TABLE... MODIFY**, y luego indicando la restricción que queremos añadir.

BORRAR RESTRICCIONES

Su sintaxis es la siguiente:

```
1 ALTER TABLE tabla DROP {PRIMARY KEY | UNIQUE(listaColumnas) |  
2 CONSTRAINT nombreRestriccion} [CASCADE]
```

La opción **PRIMARY KEY** elimina una clave principal. **UNIQUE** elimina la restricción de unicidad, realizada sobre la lista de columnas indicadas. Y siendo más versátil, la opción **CONSTRAINT** elimina la restricción cuyo nombre se indica.

La opción **CASCADE** hace que se eliminen en cascada las restricciones de integridad que dependen de la restricción eliminada y que, de otro modo, no permitiría eliminarla.

Es decir, no podemos eliminar una clave primaria que tiene claves secundarias relacionadas. Pero si indicamos **CASCADE** al eliminar la clave primaria, todas las restricciones **FOREIGN KEY** relacionadas, también lo harán.

Por ejemplo:

Tras esa definición de tabla, se verá la siguiente instrucción.

```
1 ALTER TABLE curso DROP PRIMARY KEY;
```

Esto produce un error de llave foránea, para evitarlo:

```
1 ALTER TABLE curso DROP PRIMARY KEY CASCADE;
```

Esta instrucción elimina la restricción de clave secundaria **cursos_fk1**, antes de eliminar la principal.

ACTIVACIÓN Y DESACTIVACIÓN DE RESTRICCIONES

A veces conviene desactivar temporalmente una restricción para saltarse las reglas que impone. La sintaxis es:

```
1 ALTER TABLE tabla DISABLE CONSTRAINT nombre [CASCADE];
```

La opción **CASCADE** hace que también se desactiven las restricciones dependientes de la que se desactivó.

ANULA LA DESACTIVACIÓN:

```
1 ALTER TABLE tabla ENABLE CONSTRAINT nombre;
```

Solo se permite volver a activar si los valores de la tabla cumplen la restricción que se activa. Si hubo una desactivación en cascada, habrá que activar cada restricción individualmente.

CAMBIAR DE NOMBRE A LAS RESTRICCIONES

Para hacerlo, se utiliza este comando:

```
1 ALTER TABLE table RENAME CONSTRAINT nombreViejo TO nombreNuevo;
```

CONDICIONALES

CASE no es una función, pero su utilidad se asemeja a una, es decir, devuelve un valor. Se trata de un elemento que da potencia a las instrucciones SQL, ya que simula una estructura de tipo **if-then-else**.

Sintaxis:

```
1 CASE [expresión]
2 WHEN expresión_comparación1 THEN valor_devuelto1
3 [WHEN expresión_comparación2 THEN valor_devuelto2
4 ...
5 [ELSE valor_devuelto_else]]
6 END;
```

- Se evalúa la expresión.
- Se compara su valor con el de la primera expresión de comparación.

- Si coinciden, se devuelve el valor con el del primer **THEN**.
- Si no, se compara con la expresión del siguiente **WHEN** (si le hay), y así sucesivamente para todos los **WHEN**.
- Si la expresión no coincide con la de ningún **WHEN**, entonces se devuelve el valor que posee el apartado **ELSE** (si le hay).

Ejemplo:

```
1 SELECT nombre,  
2 CASE cotizacion WHEN 1 THEN salario*0.85  
3 WHEN 2 THEN salario*0.93  
4 WHEN 3 THEN salario*0.96  
5 ELSE salario  
6 END  
7 FROM empleados;
```

En el ejemplo, se calcula una columna a partir del salario, de modo que dicho cálculo varía en función de lo que vale la columna cotización. En el caso de que esa columna no valga ni uno, ni dos, ni tres, se mostrará el salario tal cual (para eso sirve el apartado **ELSE**).

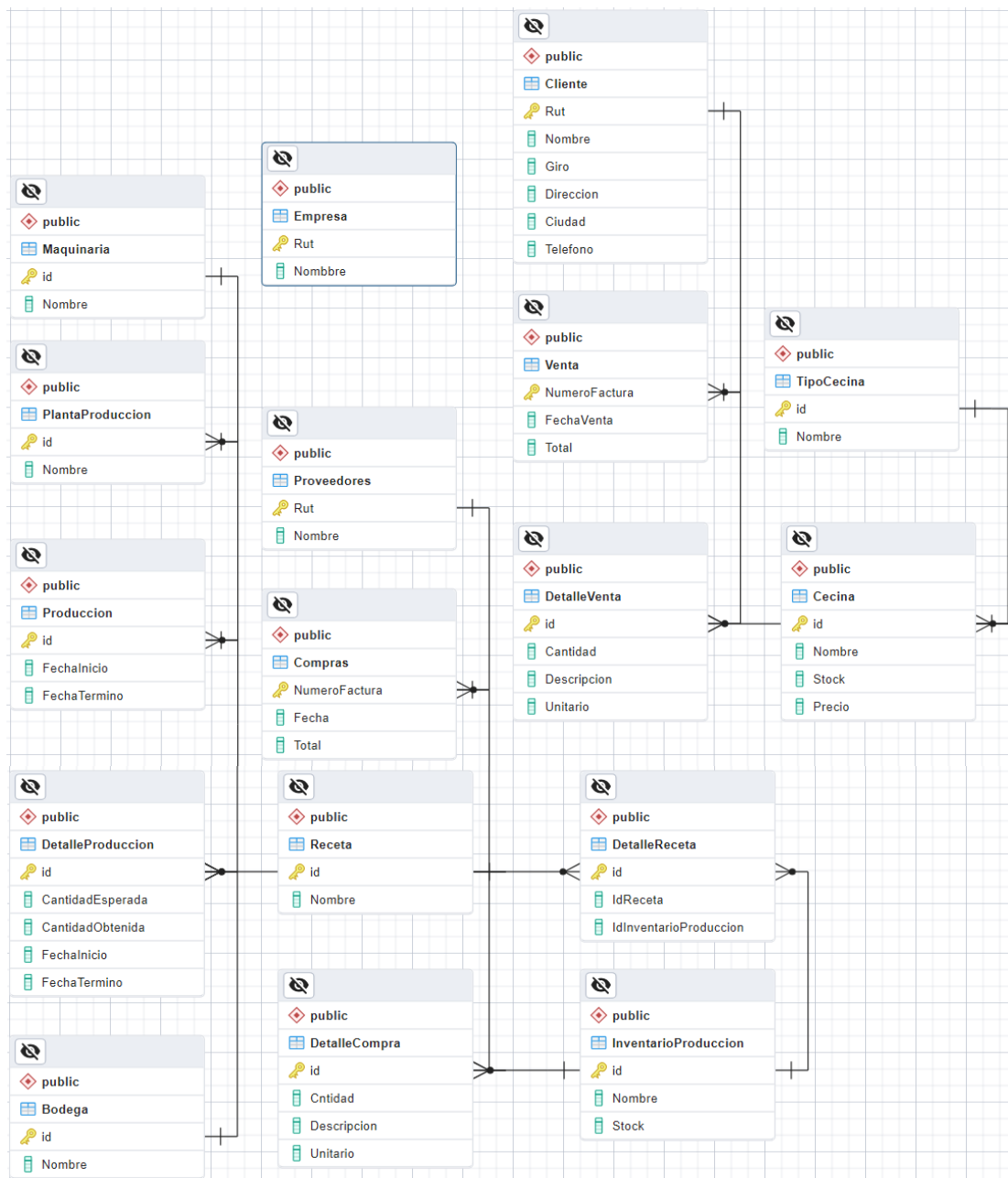
La expresión de comparación puede ser compleja. Para ello, no se indica expresión alguna, y se colocan expresiones más complejas en los apartados **WHEN**.

Ejemplo:

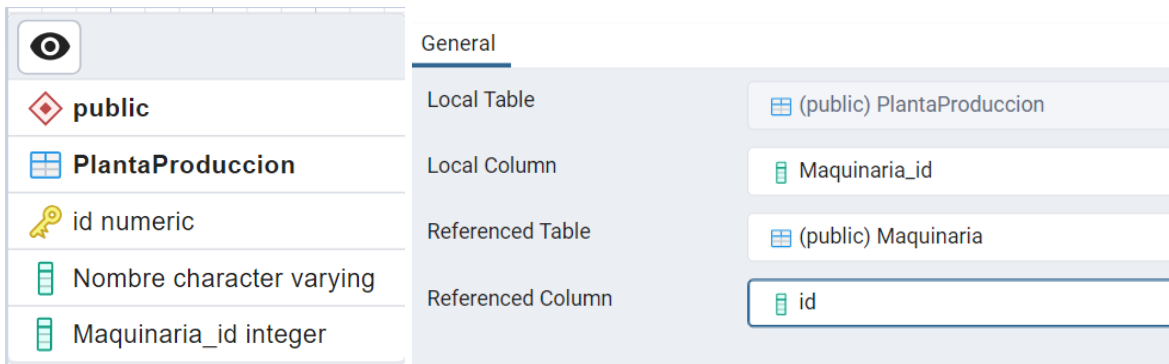
```
1 SELECT nombre,  
2 CASE WHEN nota>=4 AND nota<5 THEN 'Suficiente'  
3 WHEN nota>=5 AND nota<6 THEN 'Notable'  
4 WHEN nota>=6 THEN 'Sobresaliente'  
5 ELSE 'Insuficiente'  
6 END AS calificacion  
7 FROM alumnos
```

EXERCISE 2: PASANDO DE UN MODELO RACIONAL A UNO FISICO.

Comenzaremos considerando el modelo relacional de la fábrica de cecinas. Como podemos observar, contamos con el diagrama relacional.



Lo que necesitamos hacer ahora es asignar un atributo para cada llave foránea, asegurándonos de que tengan el mismo tipo de dato. Comenzaremos con la tabla "PlantaProduccion", donde agregaremos un nuevo atributo llamado "Maquinaria_id", de tipo "integer". Además, modificaremos el tipo de dato del atributo "id" en la tabla "Maquinaria", cambiándolo a "integer". Luego, estableceremos la relación uno a muchos conectando "Maquinaria_id" en la tabla "PlantaProduccion" con el "id" de la tabla "Maquinaria".



The screenshot shows the 'General' tab of a table configuration window. On the left, a list of tables includes 'public', 'PlantaProduccion', 'id numeric', 'Nombre character varying', and 'Maquinaria_id integer'. The 'PlantaProduccion' table is selected. On the right, the 'General' tab shows the following configuration:

Property	Value
Local Table	(public) PlantaProduccion
Local Column	Maquinaria_id
Referenced Table	(public) Maquinaria
Referenced Column	id

Al establecer esta relacion, la tabla "PlantaProduccion" mostrará un simbolo de llave en color gris, indicando la presencia de llaves foráneas en la tabla.



The screenshot shows the updated table configuration for 'PlantaProduccion'. The table now has a grey key icon next to the 'Maquinaria_id' attribute, indicating it is a foreign key. The configuration is as follows:

Attribute	Type
id	numeric
Nombre	character varying
Maquinaria_id	integer

Continuamos con la tabla "Produccion", a la que le agregaremos el atributo "PlantaProd_id" de tipo "numeric". Luego, ajustaremos la relación uno a muchos para vincularla con la tabla "PlantaProduccion", utilizando "plantaProd_id" de la tabla "Produccion" y el atributo "id" de la tabla "PlantaProduccion".

One to many relation

General	
Local Table	(public) Produccion
Local Column	PlantaProd_id
Referenced Table	(public) PlantaProduccion
Referenced Column	id

Seguiremos con la tabla "DetalleProduccion", a la que añadiremos tres nuevos atributos debido a sus relaciones de muchos a uno con las tablas "Produccion", "Bodega" y "Receta". Estos nuevos atributos serán "Receta_id", "Bodega_id" y "Produccion_id", todos de tipo "numeric".

Table: DetalleProduccion (public)





General	Columns	Advanced	Constraints
	<div> <div></div> <div></div> <div></div> </div>	<div> <div>FechaInicio</div> <div>date</div> <div></div> </div>	
	<div> <div></div> <div></div> <div></div> </div>	<div> <div>FechaTermino</div> <div>date</div> <div></div> </div>	
	<div> <div></div> <div></div> <div></div> </div>	<div> <div>Receta_id</div> <div>numeric</div> <div></div> </div>	
	<div> <div></div> <div></div> <div></div> </div>	<div> <div>Bodega_id</div> <div>numeric</div> <div></div> </div>	
	<div> <div></div> <div></div> <div></div> </div>	<div> <div>Produccion_id</div> <div>numeric</div> <div></div> </div>	



Ahora, estableceremos la primera relación entre las tablas "DetalleProduccion" y "Produccion", vinculando los atributos "Produccion_id" e "id" respectivamente.

One to many relation





General

Local Table	 (public) DetalleProduccion
Local Column	 Produccion_id
Referenced Table	 (public) Produccion
Referenced Column	 id

La segunda relación será entre las tablas "DetalleProduccion" y "Bodega", enlazando los atributos "Bodega_id" e "id".

One to many relation





General

Local Table	 (public) DetalleProduccion
Local Column	 Bodega_id
Referenced Table	 (public) Bodega
Referenced Column	 id












Y la tercera relación será entre las tablas "DetalleProduccion" y "Receta", vinculando los atributos "Receta_id" e "id".

One to many relation

General

Local Table	 (public) DetalleProduccion
Local Column	 Receta_id
Referenced Table	 (public) Receta
Referenced Column	 id

Finalmente, nuestra tabla "DetalleProduccion" quedará de la siguiente manera:


 public
 DetalleProduccion
 id numeric
 CantidadEsperada numeric
 CantidadObtenida numeric
 FechaInicio date
 FechaTermino date
 Receta_id numeric
 Bodega_id numeric
 Produccion_id numeric



Ahora, procederemos a relacionar las tablas "Receta", "InventarioProduccion" y "DetalleReceta". A la tabla "DetalleReceta" le asignaremos un atributo propio llamado "Cantidad", de tipo "numeric". Estableceremos la primera relación entre esta tabla y la tabla "Receta".

One to many relation

General

Local Table	(public) DetalleReceta
Local Column	IdReceta
Referenced Table	(public) Receta
Referenced Column	id








Luego, relacionaremos la tabla "DetalleReceta" con la tabla "InventarioProduccion", vinculando el atributo "IdInventarioProduccion" con el atributo "id".

One to many relation

General



















Local Table	(public) DetalleReceta
Local Column	IdInventarioProduccion
Referenced Table	(public) InventarioProduccion
Referenced Column	id

Finalmente, la estructura de la tabla “DetalleReceta” quedará de la siguiente manera

	
	public
	DetalleReceta
	id numeric
	IdReceta numeric
	IdInventarioProduccion numeric
	Cantidad numeric

Las siguientes tablas que relacionaremos serán “DetalleCompra”, “Compras” e “InventarioProduccion”. Comenzaremos ajustando la tabla “DetalleCompra”, cambiando el tipo de dato del atributo “Unitario” a “numeric” e incorporando los atributos “Compra_NumeroFactura” e “Inventario_id”, ambos de tipo “numeric”.

Table: DetalleCompra (public)





General	Columns	Advanced	Constraints
	Name		Data type
  	id		numeric v
  	Cantidad		integer v
  	Descripcion		character varying v
  	Unitario		numeric v
  	Compra_NumeroFactura		numeric v
  	Inventario_id		numeric v

Ajustaremos el nombre de la tabla "Compras" siguiendo el estándar de nomenclatura para tablas en bases de datos, el cual dicta que los nombres deben estar en singular. Por lo tanto, la nueva denominación será "Compra".

El próximo paso consiste en establecer la primera relación, que será entre la tabla "DetalleCompra" y la tabla "InventarioProduccion".

One to many relation





General

Local Table	 (public) DetalleCompra
Local Column	 Inventario_id
Referenced Table	 (public) InventarioProduccion
Referenced Column	 id

Posteriormente, crearemos otra relación, esta vez entre la tabla "DetalleCompra" y "Compra", vinculando los atributos "Compra_NumeroFactura" y "NumeroFactura".

One to many relation

General


Local Table	 (public) DetalleCompra
Local Column	 Compra_NumeroFactura
Referenced Table	 (public) Compra
Referenced Column	 NumeroFactura

Adicionalmente, necesitaremos relacionar las tablas "Compra" y "Proveedor". Para lograrlo, primero agregaremos el atributo "Proveedor_Rut" con el tipo de dato "character varying" a la tabla "Compra". Luego, estableceremos una relación uno a muchos conectando los atributos "Proveedor_Rut" y "Rut" de las respectivas tablas.

One to many relation

General


Local Table

 (public) Compra

Local Column

 Proveedor_Rut

Referenced Table

 (public) Proveedor

Referenced Column

 Rut

Continuamos con las tablas "Venta" y "Cliente". En la tabla "Venta", añadiremos el atributo "Cliente_Rut" de tipo "character varying" y, finalmente, estableceremos la relación entre estas dos tablas conectando los atributos "Cliente_Rut" y "Rut".


One to many relation

General


Local Table

 (public) Venta

Local Column

 Cliente_rut

Referenced Table

 (public) Cliente



















Referenced Column

 Rut



Proseguimos con la tabla "DetalleVenta", que tiene relaciones de muchos a uno con las tablas "Venta" y "Cecina". Por lo tanto, incorporaremos dos atributos a la tabla "DetalleVenta", llamados "Cecina_id" y "Venta_NumeroFactura", ambos de tipo "numeric". Además, cambiaremos el tipo de dato del atributo "Unitario" a "numeric". Luego, estableceremos la relación entre "DetalleVenta" y "Cecina" vinculando los atributos "Cecina_id" e "id".

Table: DetalleVenta (public)

General	Columns	Advanced	Constraints
	Name		Data type
  	id		numeric
  	Cantidad		integer
  	Descripcion		character varying
  	Unitario		numeric
  	Cecina_id		numeric
  	Venta_numeroFactura		numeric


One to many relation

General


Local Table

 (public) DetalleVenta

Local Column

 Cecina_id

Referenced Table

 (public) Cecina





Referenced Column

 id

La relacion siguiente será entre las tablas "DetalleVenta" y "Venta".

One to many relation





General

Local Table	 (public) DetalleVenta
Local Column	 Venta_numeroFactura
Referenced Table	 (public) Venta
Referenced Column	 NumeroFactura

Finalmente, trabajaremos con la tabla "Cecina", donde agregaremos un atributo llamado "TipoCecina_id" de tipo "numeric". Crearemos la relación entre esta tabla y la tabla "TipoCecina", conectando los atributos "TipoCecina_id" e "id".

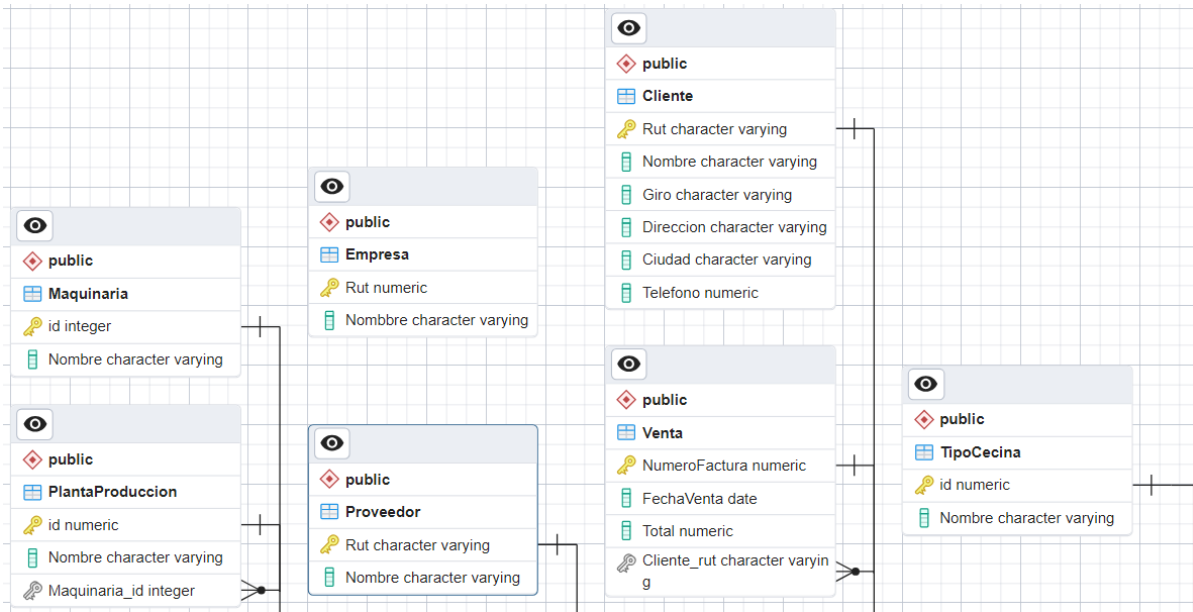
One to many relation

General

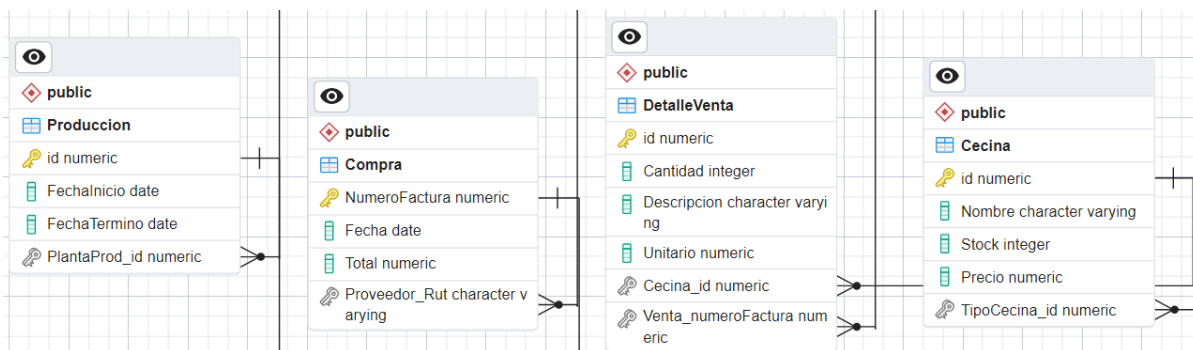
Local Table	 (public) Cecina
Local Column	 TipoCecina_id
Referenced Table	 (public) TipoCecina
Referenced Column	 id

Las tablas quedaran de la siguiente manera:

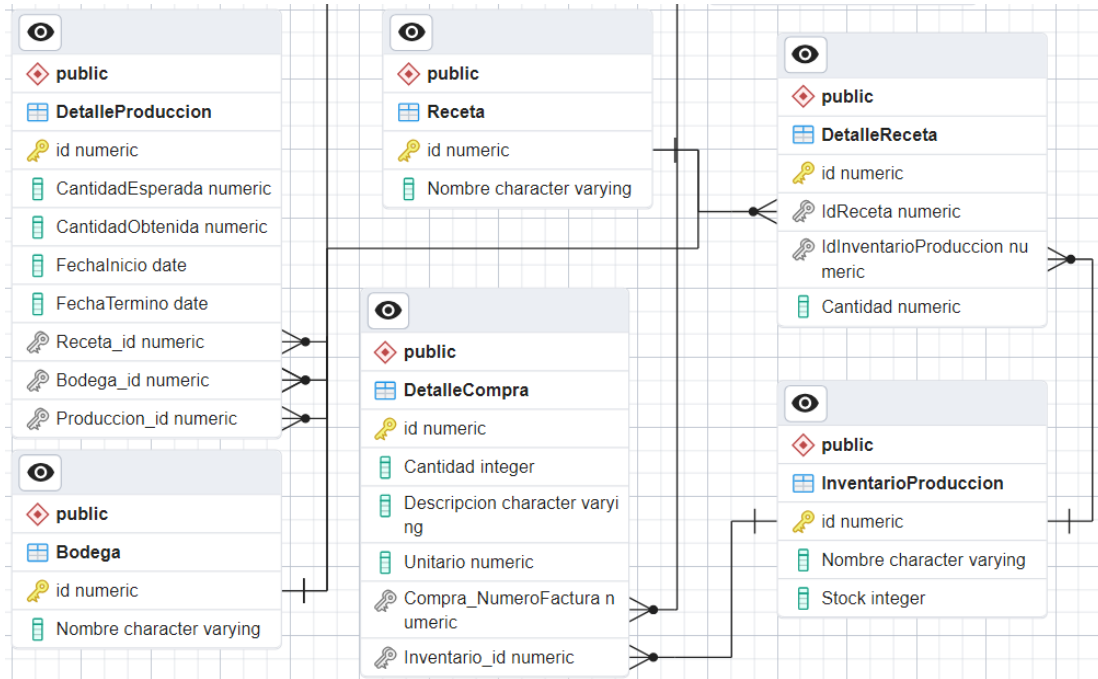
Primera parte



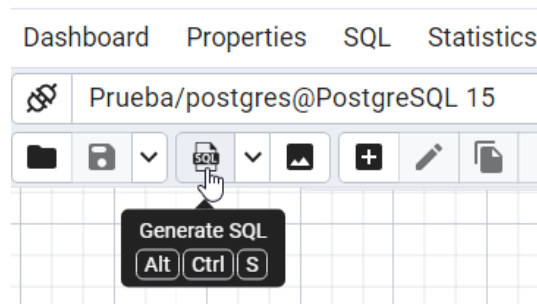
Segunda parte



Tercera parte



Una vez que tengamos nuestras tablas listas, procederemos a convertir nuestro modelo relacional en un modelo físico. Para ello, haremos clic en "Generate SQL" para obtener el código SQL.



Inmediatamente, revisaremos el script generado, donde inicialmente encontraremos la declaración "Begin". Posteriormente, observaremos la sentencia "Create table" para cada una de nuestras tablas, detallando las primary keys y los tipos de datos correspondientes.

```
2  -- Please log an issue at https://redmine.postgresql.o
3  BEGIN;
4
5
6  CREATE TABLE IF NOT EXISTS public."Empresa"
7  (
8      "Rut" numeric NOT NULL,
9      "Nombre" character varying,
10     PRIMARY KEY ("Rut")
11 );
12
13 CREATE TABLE IF NOT EXISTS public."PlantaProduccion"
14 (
15     id numeric NOT NULL,
16     "Nombre" character varying,
17     "Maquinaria_id" integer,
18     PRIMARY KEY (id)
19 );
```

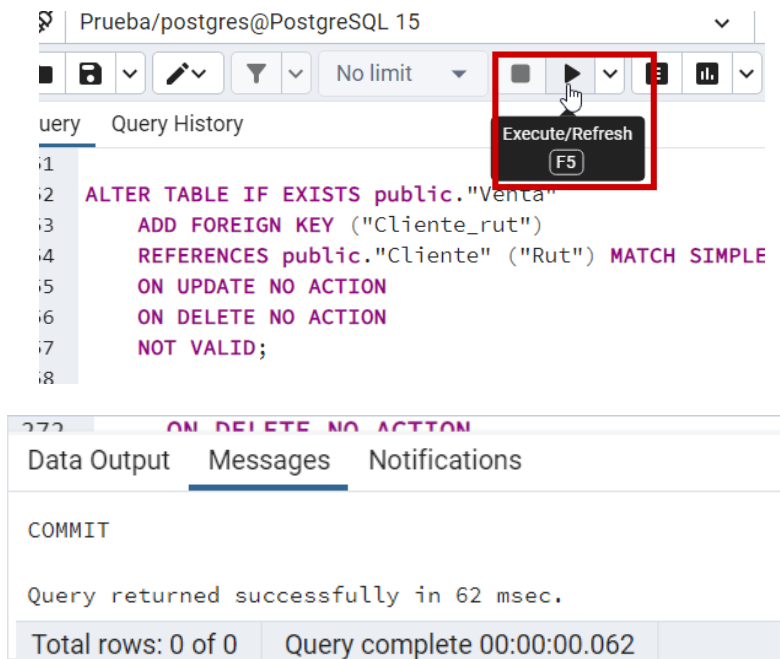
Al desplazarnos hacia abajo en el script, encontraremos las sentencias "Alter table", las cuales añaden las llaves foráneas. El script concluye con la sentencia "End".

```
ALTER TABLE IF EXISTS public."DetalleReceta"
ADD FOREIGN KEY ("IdInventarioProduccion")
REFERENCES public."InventarioProduccion" (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID;

END;
```



Luego, realizaremos una prueba de nuestro código para verificar si las tablas se crean en nuestra base de datos. Para esto, haremos clic en "Execute". Si todo sale como esperamos, veremos el mensaje indicando que la consulta se ha ejecutado satisfactoriamente.



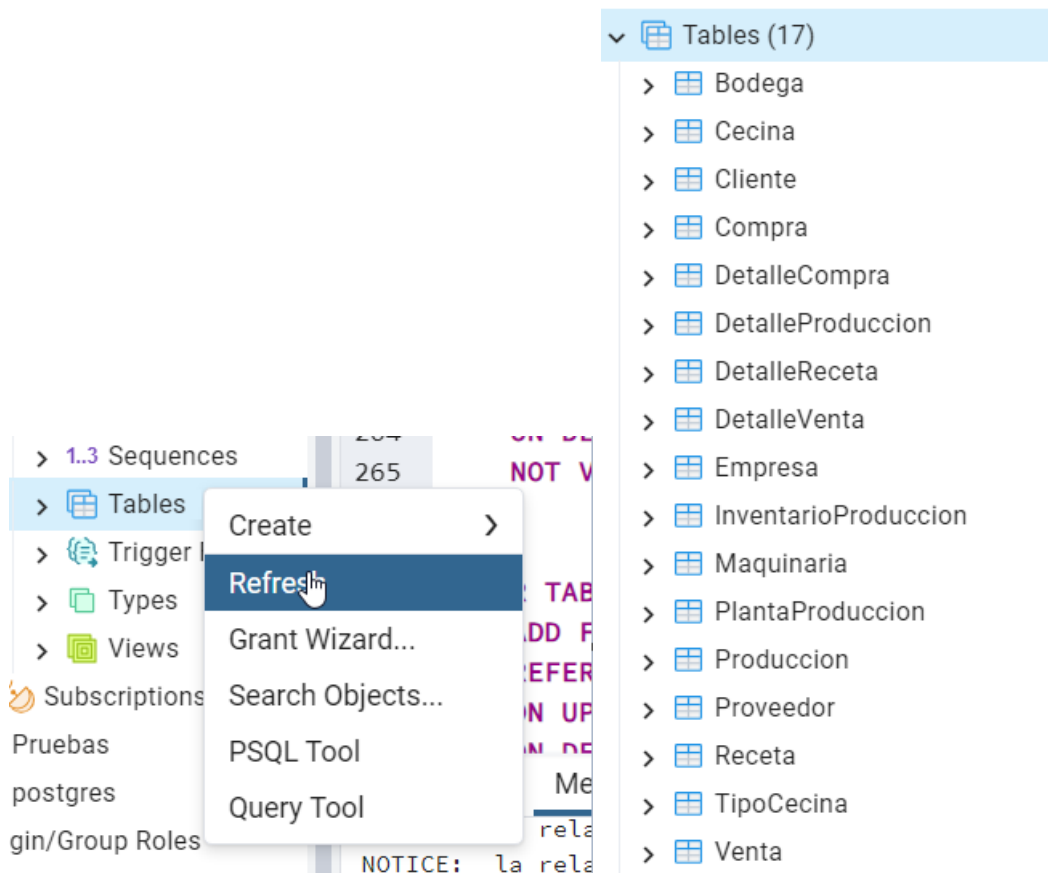
The screenshot shows a PostgreSQL query editor window titled "Prueba/postgres@PostgreSQL 15". The query editor has a toolbar with icons for saving, undo, redo, and running the query. The "Execute/Refresh" button (F5) is highlighted with a red box. Below the toolbar, the query text is displayed:

```
1 ALTER TABLE IF EXISTS public."Venta"  
2 ADD FOREIGN KEY ("Cliente_rut")  
3 REFERENCES public."Cliente" ("Rut") MATCH SIMPLE  
4 ON UPDATE NO ACTION  
5 ON DELETE NO ACTION  
6 NOT VALID;  
7  
8
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Messages" tab is selected, showing the following output:

```
COMMIT  
  
Query returned successfully in 62 msec.  
  
Total rows: 0 of 0 | Query complete 00:00:00.062
```

Para confirmar, nos dirigiremos a la sección "Tables", haremos clic derecho y seleccionaremos "Refresh". Si todo está correcto, visualizaremos la lista completa de todas las tablas creadas mediante el script.



EXERCISE 3: CREANDO SENTENCIAS DML

SENTENCIA DELETE

Elimina los registros de la tabla que cumplan la condición indicada.

Sintaxis:

```
1 DELETE [FROM] tabla [WHERE condición];
```

Ejemplo:

```
1 DELETE FROM empleados WHERE seccion=23;
```

Hay que tener en cuenta que el borrado de un registro no puede provocar fallos de integridad, y que la opción de integridad **ON DELETE CASCADE** hace que no sólo se borren las filas indicadas, sino todas las relacionadas.

SENTENCIA UPDATE

La modificación de los datos de las filas se realiza mediante la instrucción **UPDATE**.

Sintaxis:

```
1 UPDATE tabla
2 SET columna1=valor1 [,columna2=valor2...]
3 [WHERE condición];
```

Se modifican las columnas indicadas en el apartado **SET**, con los valores indicados. La cláusula **WHERE** permite especificar qué registros serán modificados.

Ejemplos:

- Actualiza la región de los clientes de Santiago, para que aparezca como R. Metropolitana.

```
1 UPDATE clientes
2 SET provincia= 'R.Metropolitana'
3 WHERE provincia='Santiago';
```

- Incrementa los precios en un 19%. La expresión para el valor puede ser todo lo compleja que se desee.

```
1 UPDATE productos SET precio = precio * 1.19;
```

- Utilizan funciones de fecha para conseguir que los partidos que se jugaban hoy, pasen a jugarse el siguiente martes.

```
1 UPDATE partidos
2 SET fecha= NEXT_DAY(SYSDATE, 'Martes' )
3 WHERE fecha=SYSDATE;
```

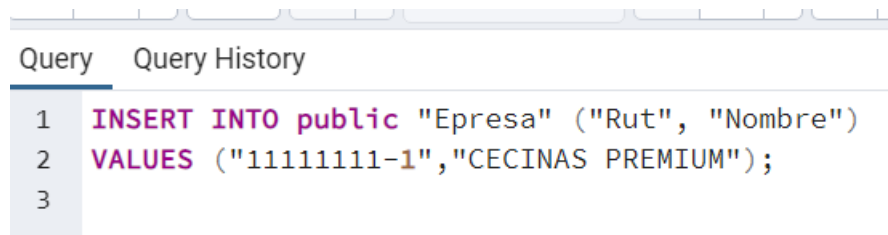
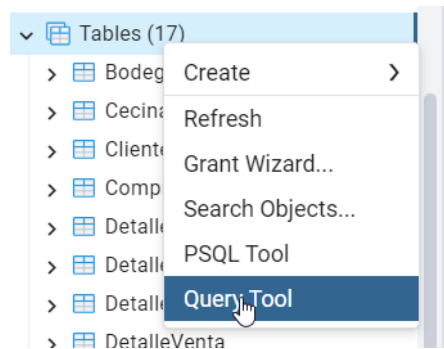
Para este ejemplo vamos a considerar el modelo físico de la fábrica de cecinas creado en el ejemplo anterior.

Trabajaremos en todas las tablas comenzaremos insertando los datos de

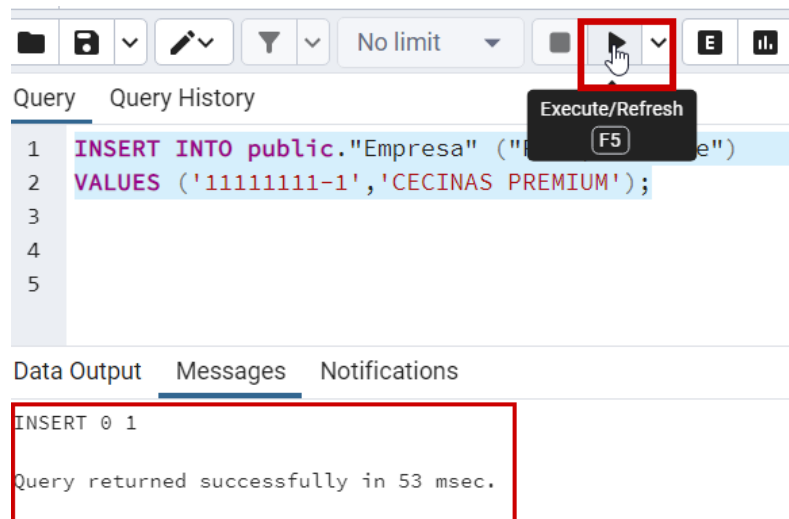
la empresa en la tabla **"Empresa"**. La entidad contiene los atributos **Rut** y **Nombre**. Para agregar datos, escribiremos con la siguiente sintaxis:

```
1 INSERT INTO public."Empresa" ("Rut", "Nombre")
2 VALUES ('111111-1', 'CECINAS PREMIUM');
```

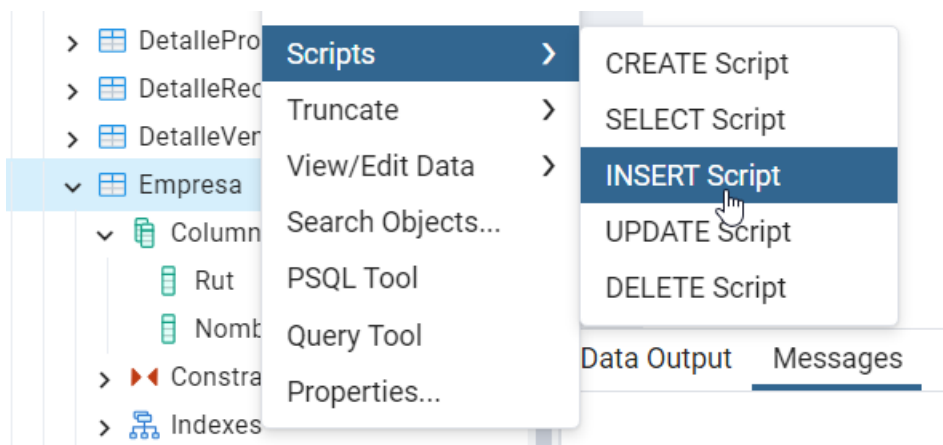
Existen dos métodos para insertar datos en una tabla. El primero consiste en copiar y pegar el código proporcionado arriba. Para ello, haz clic derecho en "Tables" y luego selecciona "Query Tool". Aparecerá una ventana donde podrás ingresar la consulta deseada. Asegúrate de tener en cuenta las comillas necesarias para un correcto funcionamiento.



Luego, selecciona el código, presionamos "Execute" y verificamos la salida del script para confirmar que la información se ha insertado correctamente.



La segunda opción es hacer clic derecho en la tabla que deseamos utilizar, seleccionamos "Script" y luego "INSERT Script". Esto abrirá automáticamente la pestaña con la consulta, donde solo tendremos que reemplazar los "?" con los datos correspondientes.



Query Query History

```
1 INSERT INTO public."Empresa"(  
2     "Rut", "Nombre")  
3     VALUES (?, ?);
```

Query Query History

```
1 INSERT INTO public."Empresa"(  
2     "Rut", "Nombre")  
3     VALUES ('11111111-1', 'CECINAS PREMIUM');  
4
```

Al hacer clic en "Execute", obtendremos la misma salida de script que se mostró en la primera forma.

Data Output	Messages	Notifications
<pre>INSERT 0 1 Query returned successfully in 53 msec.</pre>		

Continuaremos insertando un nuevo dato a la tabla **Maquinaria**. Ésta contiene los atributos **ID** y **Nombre**. Escribiremos:

```
1 INSERT INTO public."Maquinaria"(id, "Nombre")  
   VALUES (1, 'MAQUINARIA 1');
```

Query Query History

```
1 INSERT INTO public."Maquinaria"(  
2     id, "Nombre")  
3     VALUES (1, 'MAQUINARIA 1');
```

Destacaremos que los valores numéricos no llevan comillas.



Colocaremos un segundo valor ingresado en la misma tabla, con la siguiente sintaxis:

```
1 INSERT INTO public."Maquinaria"(id, "Nombre")  
VALUES (2, 'MAQUINARIA 2');
```

Query	Query History
1 INSERT INTO public."Maquinaria"(2 id, "Nombre") 3 VALUES (2, 'MAQUINARIA 2');	

Data Output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 48 msec.		

El ID, al ser una llave primaria, debe ser un valor único e irrepetible.

Continuaremos insertando un dato en la tabla **PlantaProduccion**, utilizando la siguiente sintaxis:

```
1 INSERT INTO public."PlantaProduccion"(id, "Nombre", "Maquinaria_id")  
2 VALUES (100, 'PLANTA 1', 1);
```

Query	Query History
1 INSERT INTO public."PlantaProduccion"(2 id, "Nombre", "Maquinaria_id") 3 VALUES (100, 'PLANTA 1', 1);	

Luego de eso, insertaremos un dato a la tabla Bodega que cuenta con los atributos **ID** y **nombre**. La sintaxis será:

```
1 INSERT INTO public."Bodega"(id, "Nombre")
2 VALUES (1, 'BODEGA 1');
```

Query Query History

```
1 INSERT INTO public."Bodega"(
2     id, "Nombre")
3     VALUES (1, 'BODEGA 1');
```

Luego insertaremos 3 registros a la tabla Cliente que cuenta con los atributos Rut, Nombre, Giro, dirección, Ciudad y Foro. La sintaxis usada será:

```
1 INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion",
2 "Ciudad", "Telefono") VALUES ('9546457-1', 'NOMBRE 1', 'GIRO 1',
3 'DIRECCIÓN 1', 'CIUDAD 1', 968542497);
4 INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion",
5 "Ciudad", "Telefono")VALUES ('1684455-K', 'NOMBRE 2', 'GIRO 2',
6 'DIRECCIÓN 2', 'CIUDAD 2', 916384002);
7 INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion",
8 "Ciudad", "Telefono")VALUES ('1863715-9', 'NOMBRE 3', 'GIRO 3',
9 'DIRECCIÓN 3', 'CIUDAD 3', 986465473);
```

```
INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion", "Ciudad", "Telefono")
VALUES ('9546457-1', 'NOMBRE 1', 'GIRO 1', 'DIRECCIÓN 1', 'CIUDAD 1', 968542497);

INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion", "Ciudad", "Telefono")
VALUES ('1684455-K', 'NOMBRE 2', 'GIRO 2', 'DIRECCIÓN 2', 'CIUDAD 2', 916384002);

INSERT INTO public."Cliente"("Rut", "Nombre", "Giro", "Direccion", "Ciudad", "Telefono")
VALUES ('1863715-9', 'NOMBRE 3', 'GIRO 3', 'DIRECCIÓN 3', 'CIUDAD 3', 986465473);
```

Es importante tener claro que debemos ir seleccionando y ejecutando cada una de las inserciones que se van haciendo.

Continuaremos insertando dos tipos de cecinas. La tabla "**TipoCecina**" tiene los atributos **Id** y **Nombre**. Para insertar datos lo haremos de la siguiente forma:

```
1 INSERT INTO public."TipoCecina"(id, "Nombre")
2     VALUES (20, 'TIPO A');
3 INSERT INTO public."TipoCecina"(id, "Nombre")
4     VALUES (25, 'TIPO B');
```

Query Query History

```
1  INSERT INTO public."TipoCecina"(  
2      id, "Nombre")  
3      VALUES (20, 'TIPO A');  
4  
5  INSERT INTO public."TipoCecina"(  
6      id, "Nombre")  
7      VALUES (25, 'TIPO B');
```

Debemos recordar ir ejecutando estas inserciones. Seguiremos con la inserción de 5 cecinas en la tabla correspondiente. Para hacerlo escribiremos:

```
1 INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio",
2 "TipoCecina_id")
3     VALUES (10, 'CECINA 1', 10, 3000, 20);
4 INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio",
5 "TipoCecina_id")
6     VALUES (15, 'CECINA 1', 10, 3500, 20);
7 INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio",
8 "TipoCecina_id")
9     VALUES (20, 'CECINA 1', 10, 4500, 20);
10 INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio",
11 "TipoCecina_id")
12     VALUES (25, 'CECINA 1', 10, 8000, 25);
13 INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio",
14 "TipoCecina_id")
15     VALUES (30, 'CECINA 1', 10, 1800, 25);
```

```
INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio", "TipoCecina_id")
VALUES (10, 'CECINA 1', 10, 3000, 20);

INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio", "TipoCecina_id")
VALUES (15, 'CECINA 2', 10, 3500, 20);

INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio", "TipoCecina_id")
VALUES (20, 'CECINA 3', 10, 4500, 20);

INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio", "TipoCecina_id")
VALUES (25, 'CECINA 4', 10, 8000, 25);

INSERT INTO public."Cecina"(id, "Nombre", "Stock", "Precio", "TipoCecina_id")
VALUES (30, 'CECINA 5', 10, 1800, 25);
```


TIPOCECINA_ID es la llave foránea del **id** la tabla **TipoCecina**, es por eso que deben coincidir con los existentes (20 y 25).

Eliminaremos a un cliente de la tabla **Cliente**, específicamente al primero insertado. Para eso, seleccionaremos a los clientes de la tabla:

```
1 SELECT * FROM "Cliente";
```

Query		Query History
1	SELECT * FROM "Cliente";	
2		

Al hacerlo, se nos desplegará la información de la tabla **Cliente**:

	Rut [PK] character	Nombre character vary	Giro character vary	Direccion character varying	Ciudad character vary	Telefono numeric 
1	9546457-1	NOMBRE 1	GIRO 1	DIRECCIÓN 1	CIUDAD 1	968542497
2	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
3	1863715-9	NOMBRE 3	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473

Ahora que pudimos ver cuáles son los clientes que tenemos registrados en nuestra tabla, escribiremos el comando para eliminar al primero. Esto lo haremos escribiendo:

```
1 DELETE FROM public."Cliente"  
   WHERE "Rut" = '9546457-1';
```

```
DELETE FROM public."Cliente"  
   WHERE "Rut" = '9546457-1';
```

Una vez ejecutada esta sentencia, nuevamente ejecutaremos el comando **SELECT** para corroborar que el cliente 1 haya sido eliminado:

Query		Query History				
1	SELECT * FROM "Cliente";					
Data Output		Messages				
	Rut [PK] character	Nombre character vary	Giro character vary	Direccion character varyin	Ciudad character vary	Telefono numeric
1	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
2	1863715-9	NOMBRE 3	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473

Continuaremos insertando tres proveedores. La tabla Proveedor contiene los atributos **Rut** y **Nombre**. Escribiremos:

```
1 INSERT INTO public."Proveedor"("Rut", "Nombre")
2   VALUES ('20.249.658-7', 'PROVEEDOR 1');
3 INSERT INTO public."Proveedor"("Rut", "Nombre")
4   VALUES ('10.259.825-1', 'PROVEEDOR 2');
5 INSERT INTO public."Proveedor"("Rut", "Nombre")
6   VALUES ('11.819.346-3', 'PROVEEDOR 3');
```

```
INSERT INTO public."Proveedor"(
  "Rut", "Nombre")
VALUES ('20.249.658-7', 'PROVEEDOR 1');

INSERT INTO public."Proveedor"(
  "Rut", "Nombre")
VALUES ('10.259.825-1', 'PROVEEDOR 2');

INSERT INTO public."Proveedor"(
  "Rut", "Nombre")
VALUES ('11.819.346-3', 'PROVEEDOR 3');
```

Ahora que los hemos insertado, eliminaremos al segundo proveedor utilizando el comando **DELETE**, de la siguiente forma:

```
1 DELETE FROM public."Proveedor"  
   WHERE "Rut" = '10.259.825-1';
```

Utilizando el comando **SELECT**, corroboramos que el segundo proveedor se haya eliminado:

Query

Query History

1

SELECT * FROM "Proveedor"

Data Output

Messages

Notifications

</

Continuamos desarrollando nuestro ejemplo, insertando cuatro registros en la tabla "InventarioProduccion", que cuenta con los atributos: **Id**, **Nombre** y **Stock**. Lo haremos de la siguiente forma.

```
1 INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")  
2   VALUES (1, 'PRODUCTO A', 10);  
3 INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")  
4   VALUES (2, 'PRODUCTO B', 5);  
5 INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")  
6   VALUES (3, 'PRODUCTO C', 100);
```

```
7 INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")
8     VALUES (4, 'PRODUCTO D', 33);
```

```
INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")
    VALUES (1, 'PRODUCTO A', 10);

INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")
    VALUES (2, 'PRODUCTO B', 5);

INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")
    VALUES (3, 'PRODUCTO C', 100);

INSERT INTO public."InventarioProduccion"(id, "Nombre", "Stock")
    VALUES (4, 'PRODUCTO D', 33);|
```

Seguimos insertando 2 datos a la tabla Receta, la cual contiene dos atributos que son **ID** y **nombre**. Para llevar a cabo esta indicación, escribiremos lo siguiente:

```
1 INSERT INTO public."Receta"(id, "Nombre" VALUES (1, 'RECETA A');
2 INSERT INTO public."Receta"(id, "Nombre")VALUES (2, 'RECETA B');
```

```
1 INSERT INTO public."Receta"(id, "Nombre")
2     VALUES (1, 'RECETA A');
3
4 INSERT INTO public."Receta"(id, "Nombre")
5     VALUES (2, 'RECETA B');|
```

Haremos un **SELECT** para poder ver los datos ingresados en esta tabla, utilizando el comando:

```
1 SELECT * FROM RECETA
```



Query

Query History


1


SELECT * FROM "Receta";


Data Output


Messages


Notifications























	id [PK] numeric	Nombre character varying
1	1	RECETA A
2	2	RECETA B

Continuaremos con la tabla "DetalleReceta", ingresando en ella dos valores:

```
1 INSERT INTO public."DetalleReceta"(id, "IdReceta",  
2 "IdInventarioProduccion", "Cantidad")  
3     VALUES (1, 2, 3, 5);  
4 INSERT INTO public."DetalleReceta"(id, "IdReceta",  
5 "IdInventarioProduccion", "Cantidad")  
6     VALUES (2, 2, 4, 7);
```

```
INSERT INTO public."DetalleReceta"(id, "IdReceta", "IdInventarioProduccion", "Cantidad")  
VALUES (1, 2, 3, 5);  
  
INSERT INTO public."DetalleReceta"(id, "IdReceta", "IdInventarioProduccion", "Cantidad")  
VALUES (2, 2, 4, 7);
```

Seguimos insertando una producción con todas sus recetas. Para eso, iremos a tabla Producción y observamos que tiene los atributos: **Id**, **FechaInicio**, **FechaTermino**, **PlantaPro_ID**. Lo haremos utilizando la siguiente sintaxis:

```
1 INSERT INTO PRODUCCION(ID, FECHAINICIO, FECHATERMINO, PLANTAPROD_ID)
2 VALUES (1, '10-03-2021', NULL, 100)
```

Query Editor

```
INSERT INTO public."Produccion"(id, "FechaInicio", "FechaTermino",
"PlantaProd_id") VALUES (1, '2021-03-10', NULL, 100);
```

Ejecutamos, y ya tenemos guardado el dato.

Ahora, realizaremos la modificación de un dato en una tabla. Específicamente el dato nombre del segundo cliente guardado en la tabla Cliente. Para eso, lo primero que haremos será realizar un **SELECT** para ver los clientes que tenemos:

```
1 SELECT * FROM CLIENTE
```

Esto nos imprimirá todos los datos que tenemos guardados en nuestra tabla.

Query Query History

```
1 SELECT * FROM "Cliente";
```

Data Output Messages Notifications

	Rut [PK] character va	Nombre character var	Giro character var	Direccion character varyin	Ciudad character var	Telefono numeric
1	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
2	1863715-9	NOMBRE 3	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473



Cambiaremos el nombre de “nombre 3”, a “nombre 4”. Para ello utilizaremos el siguiente comando:

```
1 UPDATE public."Cliente"  
2     SET "Nombre"= 'NOMBRE 4'  
3     WHERE "Rut" = '1863715-9';
```









De esta forma, estamos indicando que cambiaremos el nombre a “nombre 4” cuando el RUT sea 1863715-9.

Finalmente, haremos un nuevo **SELECT** para revisar el cambio realizado.

Query Query History

1 **SELECT** * **FROM** "Cliente";

Data Output Messages Notifications



	Rut [PK] character	Nombre character var	Giro character var	Direccion character varying	Ciudad character var	Telefono numeric
1	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
2	1863715-9	NOMBRE 4	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473

Listaremos ahora las producciones agregadas, usando el comando:

```
1 SELECT * FROM PRODUCCION;
```



Obteniendo un solo resultado, ya que es el único que tenemos agregado:

Query

Query History

1

SELECT * FROM "Produccion";

Data Output

Messages

Notifications

<

Seguiremos listando solamente los nombres de la tabla **Cliente**, para eso escribiremos:

```
1 SELECT "Nombre" FROM "Cliente";
```

Obteniendo como resultado:

Query

Query History

1

SELECT "Nombre" FROM "Cliente";

Data Output

Messages

Notifications



Continuaremos filtrando los nombres de la tabla Cliente, que contenga la letra O mayúscula. Para ello escribiremos:


```
1 SELECT * FROM "Cliente" WHERE "Nombre" LIKE '%O%';
```

Obtendremos como resultado ambos nombres, ya que éstos contienen la letra O mayúscula en ellos.

Query Query History

1 SELECT * FROM "Cliente" WHERE "Nombre" LIKE '%O%';

Data Output Messages Notifications

	Rut [PK] character	Nombre character varyin	Giro character varyin	Direccion character varyin	Ciudad character varyin	Telefono numeric 
1	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
2	1863715-9	NOMBRE 4	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473

Si realizamos esta misma prueba con la letra o minúscula, no obtendremos ningún resultado, ya que nuestra consulta distingue entre mayúscula y minúsculas.

Podemos también filtrar por nombres que no contengan una letra en específico, por ejemplo, la A mayúscula. Para eso, anteponeamos **NOT** a **LIKE**, de la siguiente manera:

```
1 SELECT * FROM "Cliente" WHERE "Nombre" NOT LIKE '%A%';
```

Query Query History

```
1 SELECT * FROM "Cliente" WHERE "Nombre" NOT LIKE '%A%';
```

Data Output Messages Notifications

	Rut [PK] character varying	Nombre character varying	Giro character varying	Direccion character varying	Ciudad character varying	Telefono numeric
1	1684455-K	NOMBRE 2	GIRO 2	DIRECCIÓN 2	CIUDAD 2	916384002
2	1863715-9	NOMBRE 4	GIRO 3	DIRECCIÓN 3	CIUDAD 3	986465473

Finalmente, vamos a colocar una fecha de término a la producción, colocando un dato al atributo.

Así:

```
1 SELECT * FROM PRODUCCION
```

Podemos observar que el dato **FechaTermino** aparece nulo.

Query Query History

```
1 SELECT * FROM "Produccion";
```

Data Output Messages Notifications

	id [PK] numeric	FechaInicio date	FechaTermino date	PlantaProd_id numeric
1	1	2021-03-10	[null]	100

Lo cambiaremos utilizando **UPDATE**. Y escribimos:

```
1 UPDATE PRODUCCIÓN SET FECHATERMINO = '20-03-2021' WHERE ID = 1;
```

Query

Query History

2

3 UPDATE "Produccion" SET "FechaTermino" = '2021-03-10'

4 WHERE "id" = 1

Data Output

Messages

Notifications

UPDATE 1

Query returned successfully in 49 msec.

Nuevamente hacemos un **SELECT**, y podemos ver el dato cambiado.

Query

Query History

1 SELECT * FROM "Produccion";









2

3 |

Data Output

Messages

Notifications



	id [PK] numeric	FechaInicio date	FechaTermino date	PlantaProd_id numeric
1	1	2021-03-10	2021-03-10	100

De esta forma, hemos podido analizar las instrucciones para insertar, eliminar y cambiar datos en las tablas de nuestra base de datos.