

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: CREANDO SENTENCIAS DDL.
- EXERCISE 2: 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 USER nombreTabla MODIFY (columna tipo [propiedades]
2 [columnaSiguiete tipo [propiedades] ...]);
```

EN ORACLE, LOS CAMBIOS QUE SE PERMITEN EN LAS COLUMNAS SON:

- Incrementar precisión o anchura de los tipos de datos.
- Solo se puede reducir la anchura si la máxima de un campo en la columna posee nulos en todos los registros, o todos los valores existentes tienen un tamaño menor, o igual, a la nueva anchura.
- Se puede pasar de **CHAR** a **VARCHAR**, y viceversa (si no se modifica la anchura).
- Se puede pasar de **DATE** a **TIMESTAMP**, y viceversa.
- Cualquier otro cambio solo es posible si la tabla está vacía.
- Si a través de este comando, modificamos el valor de la propiedad **DEFAULT** de una tabla, dicho cambio solo tendrá efecto en la inserción de nuevas filas.

Ejemplo:

```
1 ALTER TABLE facturas MODIFY (fecha TIMESTAMP);
```

En el caso de SQL estándar, en lugar de **MODIFY**, se emplea **ALTER** (que opcionalmente puede ir seguida de **COLUMN**).

Por ejemplo:

```
1 ALTER TABLE facturas ALTER COLUMN fecha TIMESTAMP;
```

RENOMBRAR COLUMNA

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

```
1 ALTER USER 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 **courses_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: 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';
```

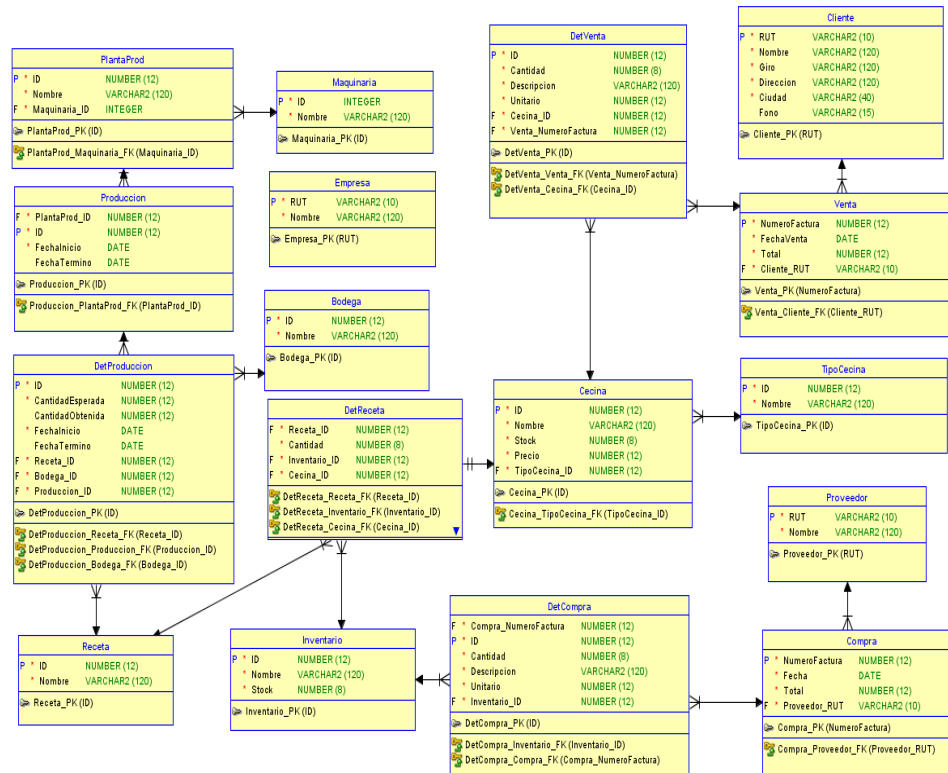
- Incrementa los precios en un 19%. La expresión para el valor puede ser todo lo compleja que se desee.

```
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;
```

Observemos el siguiente modelo relacional de la fábrica de cecinas. Como podemos notar, tenemos el diagrama relacional, por lo que crearemos las sentencias en SQL Developer.

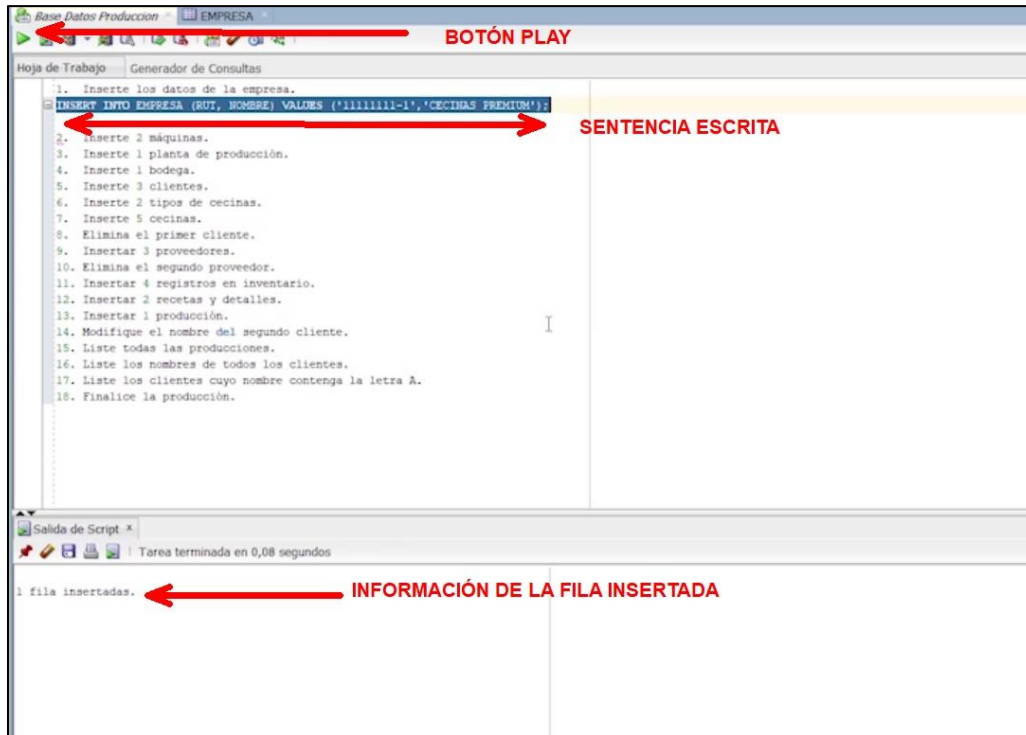


Recordemos que todas las tablas correspondientes a este modelo relacional ya fueron creadas.

Trabajaremos en todas las tablas ya creadas previamente. 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 EMPRESA (RUT, NOMBRE) VALUES ("11111111-1", "CECINAS
2 PREMIUM");
```

Luego de eso seleccionamos el botón verde de Play, y podremos ver la salida de Script indicándonos "1 fila insertada".



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

```
1 INSERT INTO MAQUINARIA (ID, NOMBRE) 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 MAQUINARIA (ID, NOMBRE) VALUES (2, "MAQUINARIA 2");
```

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

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

```
1 INSERT INTO PLANTAPROD (ID, NOMBRE, MAQUINARIA_ID) VALUES (100, "PLANTA  
2 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 BODEGA (ID, NOMBRE) VALUES (1, "BODEGA 1");  
2 Luego insertaremos 3 clientes a la tabla Cliente que cuenta con los  
3 atributos Rut, Nombre, Giro, dirección, Ciudad y Foro. La sintaxis  
4 usada será:  
5 INSERT INTO CLIENTE (RUT, NOMBRE, GIRO, DIRECCIÓN, CIUDAD, FONO)  
6 VALUES ('9546457-1', 'NOMBRE 1', 'GIRO A', 'DIRECCIÓN 1', 'CIUDAD 1',  
7 '968542497');  
8 INSERT INTO CLIENTE (RUT, NOMBRE, GIRO, DIRECCIÓN, CIUDAD, FONO)  
9 VALUES ('1684455-K', 'NOMBRE 2', 'GIRO B', 'DIRECCIÓN 2', 'CIUDAD 2',  
10 '916384002');  
11 INSERT INTO CLIENTE (RUT, NOMBRE, GIRO, DIRECCIÓN, CIUDAD, FONO)  
12 VALUES ('1863715-9', 'NOMBRE 3', 'GIRO C', 'DIRECCIÓN 3', 'CIUDAD 1',  
13 '986465473');  
14 ;
```

Es importante tener claro que debemos ir seleccionando y ejecutando cada una de las inserciones 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 TIPOCECINA (ID, NOMBRE) VALUES (20, 'TIPO A');  
2 INSERT INTO TIPOCECINA (ID, NOMBRE) VALUES (25, 'TIPO B');
```

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

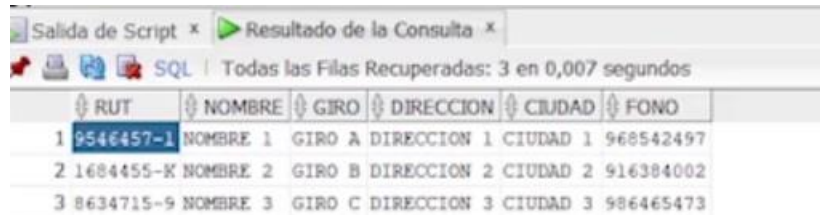
```
1 INSERT INTO CECINA (ID, NOMBRE, STOCK, PRECIO, TIPOCECINA_ID) VALUES (
2 10, 'CECINA 1', 10, 3000, 20);
3 INSERT INTO CECINA (ID, NOMBRE, STOCK, PRECIO, TIPOCECINA_ID) VALUES (
4 15, 'CECINA 2', 10, 3500, 20);
5 INSERT INTO CECINA (ID, NOMBRE, STOCK, PRECIO, TIPOCECINA_ID) VALUES (
6 20, 'CECINA 3', 10, 4500, 20);
7 INSERT INTO CECINA (ID, NOMBRE, STOC, PRECIO, TIPOCECINA_ID) VALUES (
8 25, 'CECINA 4', 10, 8000, 25);
9 INSERT INTO CECINA (ID, NOMBRE, STOCK, PRECIO, TIPOCECINA_ID) VALUES
10 (0, '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
```

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

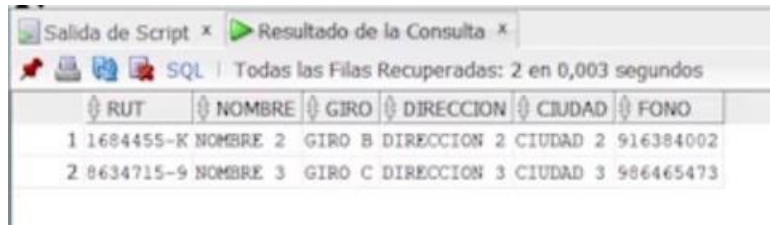


	RUT	NOMBRE	GIRO	DIRECCION	CIUDAD	FONO
1	9546457-1	NOMBRE 1	GIRO A	DIRECCION 1	CIUDAD 1	968542497
2	1684455-K	NOMBRE 2	GIRO B	DIRECCION 2	CIUDAD 2	916384002
3	8634715-9	NOMBRE 3	GIRO C	DIRECCION 3	CIUDAD 3	986465473

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 CLIENTE WHERE RUT = '9.546.457-1';
```

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



	RUT	NOMBRE	GIRO	DIRECCION	CIUDAD	FONO
1	1684455-K	NOMBRE 2	GIRO B	DIRECCION 2	CIUDAD 2	916384002
2	8634715-9	NOMBRE 3	GIRO C	DIRECCION 3	CIUDAD 3	986465473

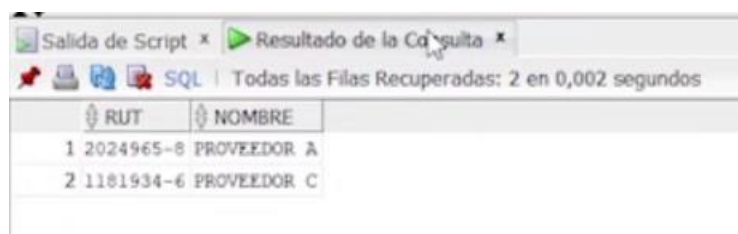
Continuaremos insertando tres proveedores. La tabla Proveedor contiene los atributos **Rut** y **Nombre**. Escribiremos:

```
1 INSERT INTO PROVEEDOR (RUT, NOMBRE) VALUES ('20.249.658-7', 'PROVEEDOR
2 A');
3 INSERT INTO PROVEEDOR (RUT, NOMBRE) VALUES ('10.259.852-1', 'PROVEEDOR
4 B');
5 INSERT INTO PROVEEDOR (RUT, NOMBRE) VALUES ('11.819.346-3', 'PROVEEDOR
6 C');
```

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

```
1 DELETE PROVEEDOR WHERE RUT = '10.259.852-1';
```

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



	RUT	NOMBRE
1	2024965-8	PROVEEDOR A
2	1181934-6	PROVEEDOR C

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

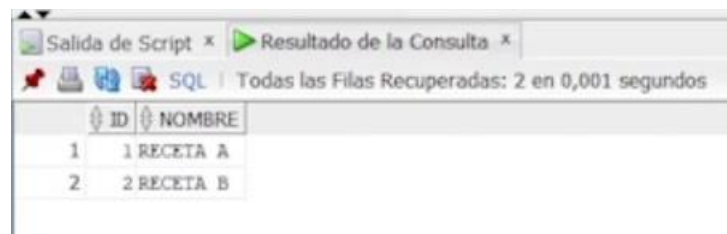
```
1 INSERT INTO INVENTARIO (ID, NOMBRE, STOCK) VALUES (1, 'PRODUCTO A',  
2 10);  
3 INSERT INTO INVENTARIO (ID, NOMBRE, STOCK) VALUES (2, 'PRODUCTO B', 5);  
4 INSERT INTO INVENTARIO (ID, NOMBRE, STOCK) VALUES (3, 'PRODUCTO C',  
5 100);  
6 INSERT INTO INVENTARIO (ID, NOMBRE, STOCK) VALUES (4, 'PRODUCTO D',  
7 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 RECETA (ID, NOMBRE) VALUES (1, 'RECETA A');  
2 INSERT INTO RECETA (ID, NOMBRE) VALUES (2, 'RECETA B');
```

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

```
1 SELECT * FROM RECETA
```



Salida de Script x Resultado de la Consulta x

SQL | Todas las Filas Recuperadas: 2 en 0,001 segundos

ID	NOMBRE
1	1 RECETA A
2	2 RECETA B

Continuaremos con la tabla DetReceta, ingresando en ella dos valores:

```
1 INSERT INTO DETRECETA (RECETA_ID, CANTIDAD) VALUES (1, 2); INSERT INTO  
2 DETRECETA (RECETA_ID, CANTIDAD) VALUES (2, 2);
```

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:

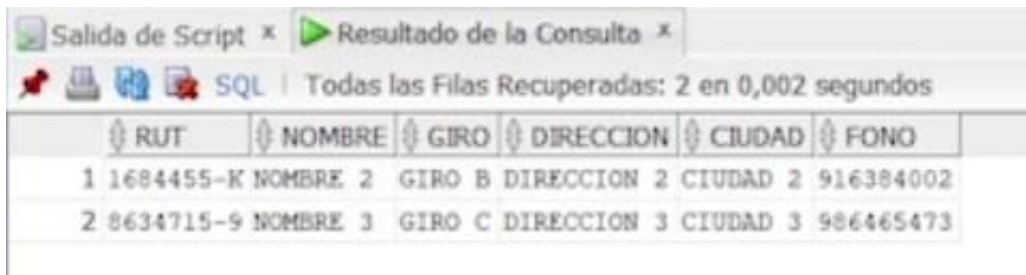

```
1 INSERT INTO PRODUCCION(ID, FECHAINICIO, FECHATERMINO, PLANTAPROD_ID)
2 VALUES (1, '10-03-2021', NULL, 100)
```

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.



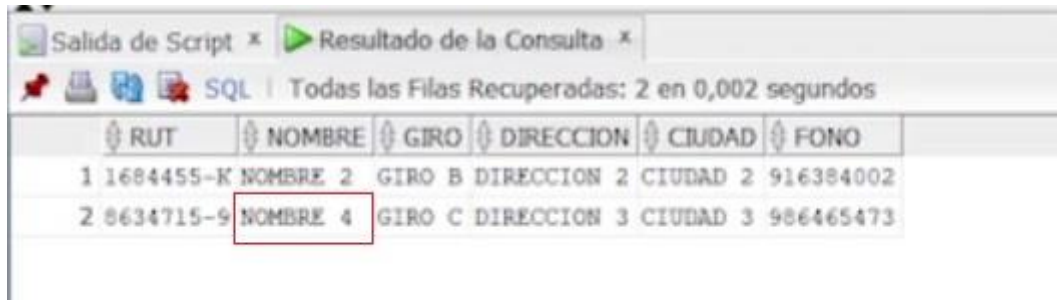
	RUT	NOMBRE	GIRO	DIRECCION	CIUDAD	FONO
1	1684455-K	NOMBRE 2	GIRO B	DIRECCION 2	CIUDAD 2	916384002
2	8634715-9	NOMBRE 3	GIRO C	DIRECCION 3	CIUDAD 3	986465473

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

```
1 UPDATE CLIENTE SET NOMBRE = 'NOMBRE 4' WHERE RUT = '8634715-9';
```

De esta forma, estamos indicando que cambiaremos el nombre a “nombre 4” cuando el RUT sea 8634715-9.

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

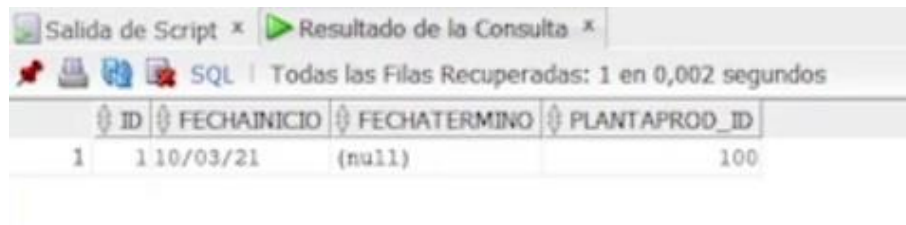


RUT	NOMBRE	GIRO	DIRECCION	CIUDAD	FONO
1 1684455-K	NOMBRE 2	GIRO B	DIRECCION 2	CIUDAD 2	916384002
2 8634715-9	NOMBRE 4	GIRO C	DIRECCION 3	CIUDAD 3	986465473

Listaremos ahora las producciones agregadas, usando el comando:

```
1 SELECT * FROM PRODUCCION
```

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

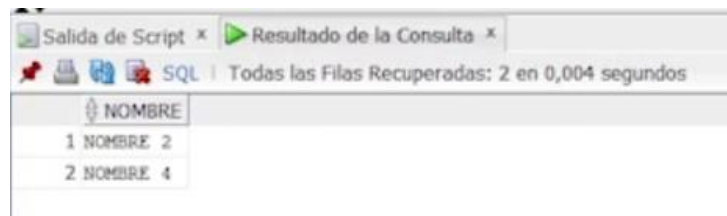


ID	FECHAINICIO	FECHATERMINO	PLANTAPROD_ID
1	1 10/03/21	(null)	100

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

```
1 SELECT NOMBRE FROM CLIENTE
```

Obteniendo como resultado:

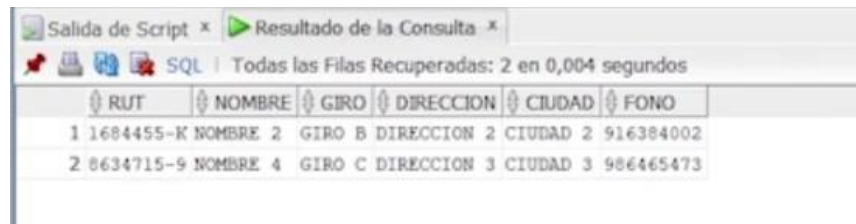


NOMBRE
1 NOMBRE 2
2 NOMBRE 4

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.



Salida de Script x Resultado de la Consulta x

Todas las Filas Recuperadas: 2 en 0,004 segundos

RUT	NOMBRE	GIRO	DIRECCION	CIUDAD	FONO
1 1684455-K	NOMBRE 2	GIRO B	DIRECCION 2	CIUDAD 2	916384002
2 8634715-9	NOMBRE 4	GIRO C	DIRECCION 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:

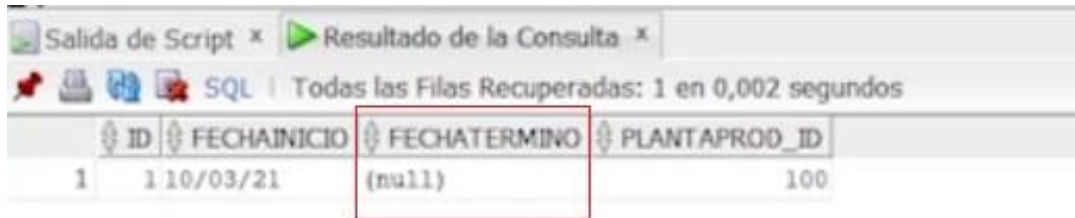
```
1 SELECT * FROM CLIENTE WHERE NOMBRE NOT LIKE '%A%';
```

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.

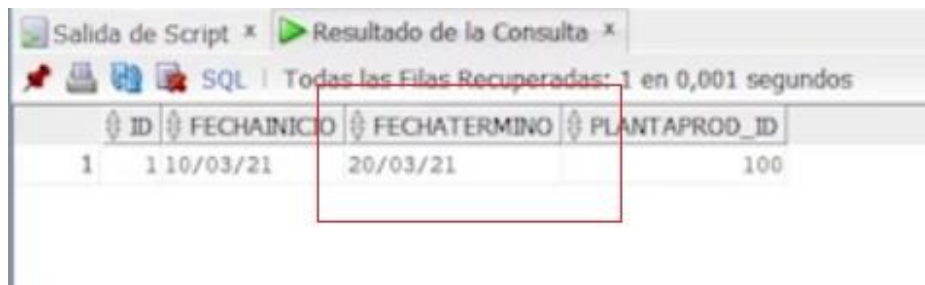


ID	FECHAINICIO	FECHATERMINO	PLANTAPROD_ID
1	1 10/03/21	(null)	100

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

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

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



ID	FECHAINICIO	FECHATERMINO	PLANTAPROD_ID
1	1 10/03/21	20/03/21	100

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