

Instrucciones de creación, eliminación, actualización e inserción de datos

Instrucciones de creación, eliminación, actualización e inserción de datos	1
¿Qué aprenderás?	2
Introducción	2
Creación de tablas	3
Comentarios	4
Creando una tabla con clave foráneas	5
Inserción de datos en una tabla	6
Actualización de registros	11
Eliminación de registros	12
Lo pernicioso del método DELETE	14
Añadiendo o eliminado columnas	14
Restricciones	15
Restricciones a nivel de PRIMARY KEY y FOREIGN KEY	17
Eliminación de una tabla	19
Truncado de una tabla	20
Cargar consultas desde un fichero	21
Observaciones finales	22



¡Comencemos!

¿Qué aprenderás?

- Construir bases de datos para el almacenamiento persistente de información.
- Crear tablas con sus atributos y tipos de datos correspondientes para el direccionamiento de datos.
- Crear claves primarias y foráneas para el enlace de referencias entre tablas.
- Importar fichero con extensión .sql

Introducción

Hasta el momento hemos instalado, configurado y aprendido ciertos comandos de Oracle Database, ahora, es momento que comencemos a crear nuestra primera base de datos con las primeras tablas.

El proceso de vida de una tabla en una base de datos parte con el proceso de Crear para luego Insertar, Actualizar y Eliminar, que responde a las operaciones elementales que podemos realizar en una tabla (CRUD). No obstante, las bases de datos no están compuestas de una única tabla sino de varias, que se interconectan y referencian para evitar redundancia (que no se repita el mismo dato) de datos, almacenamientos en cascada de información y normalizaciones.

En este capítulo practicaremos con un ejemplo sobre los directorios de teléfonos y cómo se relacionan con las agendas.

¡Vamos con todo!



Creación de tablas

La primera tarea que debemos realizar **es la creación de la tabla que permitirá almacenar los datos en filas y columnas.**

Para crear una tabla dentro de nuestro motor de base de datos, debemos utilizar el comando **CREATE TABLE** acompañado del nombre de la tabla y declarar los atributos con su respectivo tipo de dato, ingresándolos entre paréntesis. La forma canónica de creación es la siguiente:

```
CREATE TABLE nombre_tabla(  
  columna1 tipo_de_dato1,  
  columna2 tipo_de_dato2,  
  columna3 tipo_de_dato3,  
  PRIMARY KEY (columnaN)  
)
```

Luego de definir los nombres de las columnas y sus tipos de datos separados con “,” definimos también con las palabras reservadas “PRIMARY KEY” cuál de estas columnas tomará el rol de clave primaria.

Volviendo a nuestro ejemplo, con el siguiente código vamos a crear la tabla **Directorio_telefonico** con los campos **nombre, apellido, numero_telefonico, direccion y edad.**

```
CREATE TABLE directorio_telefonico  
(  
  nombre          VARCHAR2(25),  
  apellido         VARCHAR2(25),  
  numero_telefonico VARCHAR2(8),  
  direccion        VARCHAR2(255),  
  edad             NUMBER,  
  PRIMARY KEY (numero_telefonico)  
);
```

Comentarios

Las líneas precedidas por `--` representan comentarios específicos sobre cada línea. El siguiente código es una réplica de la creación de la tabla `directorio_telefonico` realizada en el punto anterior pero con comentarios agregados.

```
-- Creamos una tabla con el nombre directorio_telefonico
CREATE TABLE directorio_telefonico
(
  -- Definimos el campo nombre con el tipo de dato cadena con un largo de
  -- 25 caracteres
  nombre          VARCHAR2(25),
  -- Definimos el campo apellido con el tipo de dato cadena con un largo
  -- de 25 caracteres.
  apellido        VARCHAR2(25),
  -- Definimos el campo numero_telefonico con el tipo de dato cadena con
  -- un largo de 8 caracteres.
  numero_telefonico VARCHAR2(8),
  -- Definimos el campo dirección con el tipo de dato cadena con un largo
  -- de 255 caracteres.
  direccion       VARCHAR2(255),
  -- Definimos el campo edad con el tipo de dato numérico.
  edad            NUMBER,
  -- Definimos que el campo numero_telefonico representará la clave
  -- primaria de la tabla.
  PRIMARY KEY (numero_telefonico)
);
```

Para poder ver la estructura que tiene nuestra tabla, podemos usar el comando: `SELECT *` `FROM nombre_tabla`.

Es decir:

```
SELECT * FROM directorio_telefonico;
```

Nos entrega como resultado:

```
NOMBRE|APELLIDO|NUMERO_TELEFONICO|DIRECCION|EDAD|
-----|-----|-----|-----|-----|
```

Creando una tabla con clave foráneas

Posterior a la creación de la primera tabla, definir los componentes de la segunda tabla Agenda con los campo **numero_telefonico** y **nick**, asignando una clave foránea proveniente de la tabla **directorio_telefonico**.

Profundicemos sobre la última instrucción. La forma canónica de implementar una clave foránea responde a los siguientes componentes:

```
CREATE TABLE agenda
(
  -- Definimos el campo nick con el tipo de dato cadena con un largo de 25
  -- caracteres
  nick          VARCHAR2(25),
  -- Definimos el campo numero_telefonico con el tipo de dato cadena con
  -- un largo de 8 caracteres.
  numero_telefonico VARCHAR2(8),
  -- Vinculamos una clave foránea entre nuestra columna numero_telefonico
  -- y su símil en la tabla directorio telefónico
  FOREIGN KEY (numero_telefonico) REFERENCES
  directorio_telefonico(numero_telefonico)
);
```

De manera similar, nuestra tabla Agenda se ve de la siguiente manera, sin registros de momento.

```
SELECT * FROM agenda;

NICK|NUMERO_TELEFONICO|
----|-----|
```

Veamos cómo se ve esto en un diagrama, en DBeaver damos clic derecho en: Esquema -> usuario (ejemplo alumno). Seleccionar ver esquema, luego ir a la pestaña Diagrama ER:

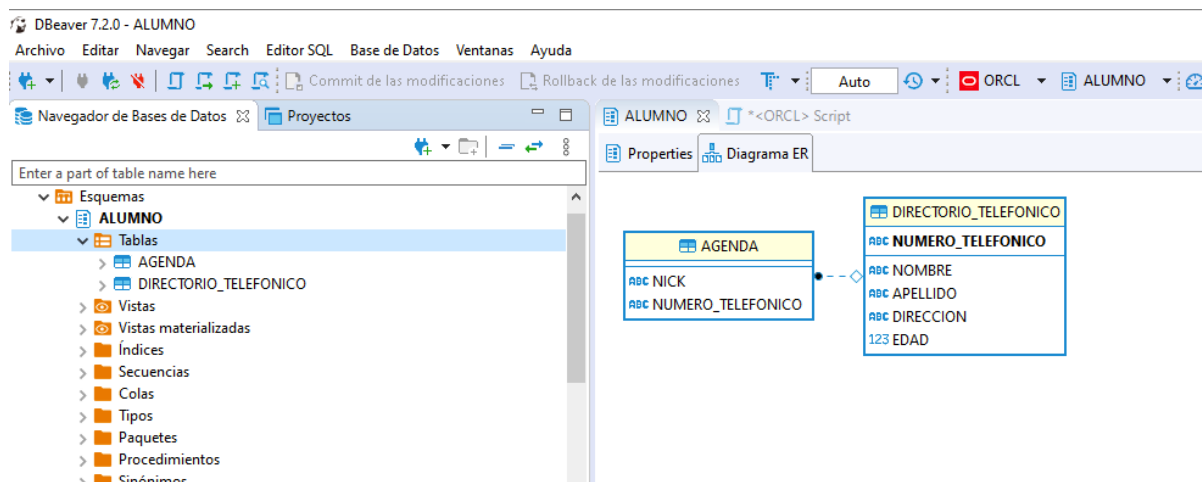


Imagen 1. Diagrama ER generado por DBeaver.

Con este diagrama, es más claro poder ver nuestras tablas y cómo se relacionan entre ellas.

Podemos ver que el usuario que estamos ocupando es el usuario que creamos, es decir Alumno y es en esta base, donde estamos creando la estructura de nuestra agenda. Es posible y sería de muy buen ejercicio, crear un usuario agenda y poblar allí la base de datos. Nosotros seguiremos mientras podamos, explotando a nuestro usuario Alumno.

Inserción de datos en una tabla

Para que una base de datos sea útil, debe contener datos. Para insertar datos a una tabla, existe el comando **INSERT**, que indica la tabla a la que se agregaron, los tipos de datos y los valores que queremos ingresar en el registro.

Esto es un proceso ordenado y debemos especificar, a qué fila pertenecen los datos que estamos ingresando. Es necesario usar un orden claro, ya que si no, la tabla pierde su estructura, haciendo imposible obtener la información buscada y la base de datos perdería su real utilidad.

La forma canónica de la instrucción **INSERT** es la siguiente:

```
INSERT INTO nombre_tabla
(
    columna1,
    columna2,
    columna3
)
VALUES
```

```
(  
    valor1,  
    valor2,  
    valor3  
);
```

Por ejemplo, ingresamos un registro a las tablas `Directorio_telefonico` y `Agenda`:

```
-- Definimos qué tabla vamos a insertar datos  
INSERT INTO directorio_telefonico  
-- Explicitamos cuáles son las columnas a insertar  
(  
    nombre,  
    apellido,  
    numero_telefonico,  
    direccion,  
    edad  
)  
-- Con la instrucción VALUES logramos asociada cada columna con un valor  
específico  
VALUES  
(  
    'Juan',  
    'Perez',  
    '12345678',  
    'Villa Pajaritos',  
    21  
)
```

Comprobemos si tenemos los datos:

```
SELECT * FROM directorio_telefonico;
```

NOMBRE	APELLIDO	NUMERO_TELEFONICO	DIRECCION	EDAD
Juan	Perez	12345678	Villa Pajaritos	21

Hay que mencionar que el formato de la sintaxis no importa, podemos realizar la consulta en una sola línea y funcionaria de la misma forma, recordemos que el punto y coma, marca el final de la consulta o instrucción.

Ingresemos más registros:

```
INSERT INTO directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad)
VALUES
    ('Fabian', 'Salas', '32846352', 'Playa Ancha', 21);

INSERT INTO directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad)
VALUES
    ('John', 'Rodriguez', '23764362', 'Constitución', 21);

INSERT INTO directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad)
VALUES
    ('Braulio', 'Fuentes', '23781363', 'Rancagua', 19);

-- Verificamos los datos en la tabla
SELECT * FROM directorio_telefonico;
```

NOMBRE	APELLIDO	NUMERO_TELEFONICO	DIRECCION	EDAD
Juan	Perez	12345678	Villa Pajaritos	21
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitución	21
Braulio	Fuentes	23781363	Rancagua	19

¿Qué ocurre si tratamos de ingresar el siguiente registro?

```
INSERT INTO directorio_telefonico
    (nombre, apellido, numero_telefonico, direccion, edad)
VALUES
    ('Marta', 'Fuentes', '23781363', 'Santiago', 24);
```

Ocurre el siguiente error:

```
org.jkiss.dbeaver.model.sql.DBSQLException: SQL Error [1] [23000]:
ORA-00001: restricción única (ALUMNO.SYS_C0011096) violada
```

Donde especifica que se está intentado agregar un valor duplicado, violando la restricción de que una clave primaria debe ser de carácter único.

Ahora, ingresemos un par de registros en en la tabla Agenda de la siguiente manera:

```
-- Realicemos el mismo procedimiento en la tabla Agenda
INSERT INTO Agenda
  (nick, numero_telefonico)
VALUES
  ('Juanito', '12345678');
```

Recordemos que hicimos una relación entre la tabla `directorio_telefonico` y `agenda`, indicando que la *clave foránea de la tabla Agenda es el numero_telefonico* de la tabla `Directorio_telefonico`. Entonces, ¿Qué ocurrirá si tratamos de ingresar otro nick con el mismo número?

```
INSERT INTO Agenda
  (nick, numero_telefonico)
VALUES
  ('Juancho', '12345678');
```

No hay problema, ya que el número telefónico existe en la tabla `directorio_telefonico`.

Veamos ambas tablas.

```
SELECT * FROM DIRECTORIO_TELEFONICO;
```

NOMBRE	APELLIDO	NUMERO_TELEFONICO	DIRECCION	EDAD
Juan	Perez	12345678	Villa Pajaritos	21
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitución	21
Braulio	Fuentes	23781363	Rancagua	19

```
SELECT * FROM AGENDA;
```

NICK	NUMERO_TELEFONICO
Juanito	12345678
Juancho	12345678

Ahora, intentemos agregar un número que no existe en la tabla de

Directorio_telefonico:

```
INSERT INTO Agenda  
  (nick, numero_telefonico)  
VALUES  
  ('Fabi', '32846351');
```

El error arrojado es el siguiente:

```
org.jkiss.dbeaver.model.sql.DBSQLException: SQL Error [2291] [23000]:  
ORA-02291: restricción de integridad (ALUMNO.SYS_C0011097) violada -  
clave principal no encontrada
```

Especificando que se viola la restricción de la llave foránea, ya que no existe en la tabla de `directorio_telefonico`.

Para soltarnos un poco, ingresemos valores hasta que las tablas nos entreguen el siguiente resultado:

Tabla `directorio_telefonico`:

nombre	apellido	numero_telefonico	dirección	edad
Juan	Perez	12345678	Villa Pajaritos	21
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitución	21
Braulio	Fuentes	23781363	Rancagua	19
Pedro	Arriagada	38472940	Valdivia	23
Matias	Valenzuela	38473623	Nogales	22
Cristobal	Missana	43423244	Con Con	20
Farid	Zalaquett	38276523	La Florida	20
Daniel	Hebel	43683283	San Bernardo	20
Javiera	Arce	94367238	Quilpue	20

Tabla 1. directorio_telefonico.

Tabla Agenda:

nick	numero_telefonico
Juanito	12345678
Juancho	12345678
Peter	38472940
Mati	38473623
Cris	43423244
Javi	94367238
Farid	32876523
Dani	43683283

Tabla 2. Campos de Agenda.

Actualización de registros

A veces nos vemos en la necesidad de cambiar los registros ya existentes, sin embargo, es riesgoso borrarlos e ingresarlos nuevamente (más adelante se profundizará en esto), por lo que si queremos actualizar los datos de un registro, debemos usar el comando `UPDATE` de la siguiente forma:

```
UPDATE nombre_tabla SET columna1=valor_nuevo WHERE condicion;
```

Juan se cambió de casa a Villa Los Leones, por lo que tenemos que actualizar la tabla `Directorio_telefonico`:

```
UPDATE directorio_telefonico  
SET direccion='Villa Los Leones'  
WHERE nombre='Juan';
```

Quedando la tabla de la siguiente forma:

nombre	apellido	numero_telefonico	dirección	edad
Fabian	Salas	32846352	Playa Ancha	21
John	Rodriguez	23764362	Constitución	21
Braulio	Fuentes	23781363	Rancagua	19
Pedro	Arriagada	38472940	Valdivia	23
Matias	Valenzuela	38473623	Nogales	22
Cristobal	Missana	43423244	Con Con	20
Farid	Zalaquett	38276523	La Florida	20
Daniel	Hebel	43683283	San Bernardo	20
Javiera	Arce	94367238	Quilpue	20
Juan	Perez	12345678	Villa Los Leones	21

Tabla 3. Tabla directorio_telefonico.

Y vemos que la dirección ha sido actualizada. Considera que de no especificar el “where” se actualizarán los valores de esa columna en todos los registros.

Eliminación de registros

Cuando estamos completamente seguros de que queremos eliminar un registro de una tabla o incluso la tabla completa, y que de verdad estamos seguros de que la condición que usamos es la correcta, podemos utilizar el comando DELETE. Esto, lo que hace es eliminar uno, varios o todos los registros de la tabla según la condición dada. La sintaxis para eliminar toda la tabla es:

```
DELETE FROM tabla;
```

Para poder seleccionar qué registros queremos borrar debemos hacerlo de la siguiente forma:

```
DELETE FROM tabla WHERE condicion;
```

En nuestro ejemplo eliminaremos a John de la tabla de `Directorio_telefonico` de la siguiente forma:

```
DELETE FROM Directorio_telefonico WHERE nombre='John';
```

Si quisiéramos borrar todos los datos de una tabla usaremos:

```
DELETE FROM Agenda;
```

Si revisamos la tabla, veremos que está vacía.

```
SELECT * FROM AGENDA;  
  
NICK|NUMERO_TELEFONICO|  
----|-----|
```

Agreguemos un registro a la tabla Agenda:

```
INSERT INTO Agenda  
VALUES('Luis','12345678');
```

Ahora eliminar a Juan de la tabla de `directorio_telefonico`.

```
DELETE FROM Directorio_telefonico WHERE nombre='Juan';
```

Obtenemos un error recordándonos que no se puede violar la restricción de clave foránea del número telefónico:

```
org.jkiss.dbeaver.model.sql.DBSQLException: SQL Error [2292] [23000]:  
ORA-02292: restricción de integridad (ALUMNO.SYS_C0011097) violada -  
registro secundario encontrado
```

Lo pernicioso del método DELETE

Si bien el comando DELETE existe para ser usado, **debe usarse con mucha precaución**, es más, sólo el administrador de la base de datos debería ser capaz de eliminar datos de ella.

Este comando es susceptible a errores, ya que si no implementamos correctamente la condición en el comando WHERE, podemos eliminar datos que no teníamos pensado borrar y no hay posibilidad de recuperarlos.

Le asignamos todos los privilegios al usuario alumno, sin embargo, deberíamos evaluar si utilizar el comando DELETE, el administrador de la base de datos es el encargado de otorgar ese poder.

Añadiendo o eliminado columnas

Utilizando la definición anterior de nuestra tabla Agenda, se busca agregar ahora una nota. Ante la eventualidad que deseamos añadir/remove una columna específica de una tabla, podemos implementar la siguiente sintaxis:

```
ALTER TABLE
  nombre_tabla
ADD
  nueva_columna tipo_de_dato;
```

Para la aplicación de este tipo de instrucción en nuestro ejemplo, procedemos a ejecutar el siguiente código en nuestra consola de Oracle para la inclusión del campo "nota" en nuestra tabla Agenda.

```
-- Declaramos que alteraremos la tabla Agenda
ALTER TABLE
  Agenda
-- Definimos el campo nick con el tipo de dato cadena con un largo de 25
caracteres.
ADD
  nota VARCHAR2(100);
```

La estructura de la tabla Agenda quedaría entonces de la siguiente manera:

```
NICK | NUMERO_TELEFONICO | NOTA |
---- | -
```

Si deseamos eliminar alguna columna en específico, podemos implementar la instrucción DROP de manera análoga a como lo hicimos con ADD.

```
ALTER TABLE
  nombre_tabla
DROP nueva_columna tipo_de_dato;
```

Restricciones

Puede que existan casos en los que nuestras columnas necesiten reglas que cumplir, como que no hayan valores repetidos o que no existan campos vacíos. Es por eso que aparece el concepto de restricciones en este listado se muestran las principales con su respectiva definición:

- **NOT NULL:** La columna no puede tener valores NULL.
- **UNIQUE:** Todos los valores de la columna deben ser diferentes unos a otros.
- **PRIMARY KEY:** Aplica la clave primaria.
- **FOREIGN KEY:** Aplica la clave foránea.
- **CHECK:** Todos los valores de una columna deben satisfacer una condición en específico.
- **DEFAULT:** Le da un valor por defecto a aquellos registros que no tengan un valor asignado.
- **INDEX:** Sirve para crear y recuperar data de forma rápida.

Estos se aplican de la siguiente forma:

```
-- Creamos una tabla
CREATE TABLE nombre_tabla
(
-- Declaramos una serie de restricciones a cada campo de dato creado
columna1 tipo_de_dato1 restriccion,
columna2 tipo_de_dato2 restriccion,
columna3 tipo_de_dato3 restriccion
```

```
);
```

PRIMARY KEY y FOREIGN KEY se manejan de forma distinta, los explicaremos en la siguiente sección. Usaremos de ejemplo las mismas tablas que creamos antes, asignándoles restricciones:

Volveremos a crear las tablas directorio_telefónico y Agenda:

```
CREATE TABLE Directorio_telefonico
(
  nombre VARCHAR2(25) NOT NULL,
  apellido VARCHAR2(25),
  numero_telefonico VARCHAR2(8) UNIQUE,
  direccion VARCHAR2(255),
  edad NUMBER
);
```

```
CREATE TABLE Agenda
(
  nick VARCHAR(25) NOT NULL,
  numero_telefonico VARCHAR(8) UNIQUE
);
```

En este caso, hicimos que numero_telefonico sea un valor único que no pueda repetirse, que nombre y nick no puedan tener valores NULL.

Restricciones a nivel de PRIMARY KEY y FOREIGN KEY

Como mencionamos antes, las claves primarias sirven para identificar un registro único en una tabla, y la clave foránea sirve para referenciar esa columna desde otra tabla. Se tiene la siguiente imagen:

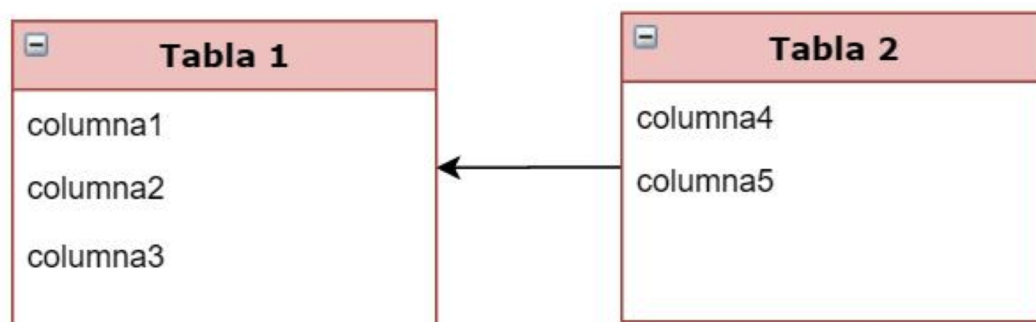


Imagen 2. Primary and Foreign Key.

Suponiendo que la columna1 de la tabla 1 fuese su clave primaria y la columna4 de la tabla 2 la foránea, la forma de aplicarlas es la siguiente:

```
CREATE TABLE tabla1
(
    columna1 tipo_de_dato1 UNIQUE,
    columna2 tipo_de_dato2,
    columna3 tipo_de_dato3,
    PRIMARY KEY (columna1)
);

CREATE TABLE tabla2
(
    columna4 tipo_de_dato4,
    columna5 tipo_de_dato5,
    FOREIGN KEY (columna4) REFERENCES tabla1(columna1)
);
```

Nota que se ha agregado la restricción “UNIQUE” en la columna 1 para justamente restringir que los valores de ese atributo en todos los registros deben ser únicos.

Volvamos al ejemplo de `directorio_telefonico` y `Agenda`, ambos usan los mismos números telefónicos para referirse a las mismas personas.

La diferencia aquí, es que obtenemos los números desde `directorio_telefonico` para llevarlos a `Agenda`.

Es por esto, que será clave primaria (`PRIMARY KEY`) y clave foránea (`FOREIGN KEY`) en `Agenda`.

```
CREATE TABLE directorio_telefonico
(
  nombre VARCHAR2(25),
  apellido VARCHAR2(25),
  numero_telefonico VARCHAR2(8),
  direccion VARCHAR2(255),
  edad NUMBER,
  PRIMARY KEY (numero_telefonico)
);

CREATE TABLE Agenda
(
  nick VARCHAR2(25),
  numero_telefonico VARCHAR2(8),
  nota VARCHAR2(100),
  FOREIGN KEY (numero_telefonico) REFERENCES
  directorio_telefonico(numero_telefonico)
);
```

Eliminación de una tabla

Hemos creado y modificado tablas, añadido registros y definimos las restricciones de nuestro modelo de datos, pero ¿qué pasa si queremos eliminar la tabla por completo?

Es riesgoso y se hará en muy pocas ocasiones, ya que una vez eliminada la tabla no tenemos forma de recuperar los datos que almacenaba, ni su estructura.

Insertemos un registro en las tablas creadas recientemente, para ver cómo se comportan:

```
INSERT INTO
  directorio_telefonico
VALUES
  ('Juan', 'Perez', 1234, 'Av.Falsa 123', 20);

INSERT INTO
  Agenda
VALUES
  ('Juanito', 1234, 'Juanito dice hola');
```

Imaginemos que queremos eliminar la tabla Agenda, lo haremos de la siguiente manera:

```
DROP TABLE Agenda;
```

La respuesta es:

```
Updated Rows      0
Query DROP TABLE Agenda
Finish time Thu Sep 10 23:04:05 BRT 2020
```

Al consultar sobre la tabla Agenda.

```
select * from Agenda;
```

Nos muestra lo siguiente:

```
SQL Error [942] [42000]: ORA-00942: la tabla o vista no existe
```

La tabla ha sido eliminada completamente.

Truncado de una tabla

Si lo que realmente necesitamos es eliminar sólo los registros, pero no la tabla en sí, usaremos el comando `TRUNCATE`, de esta manera, limpiaremos todo lo que hayamos insertado en ella.

Debes utilizar este comando sólo si es realmente necesario, ya que al igual que el comando `DROP` no es posible recuperar la información una vez eliminada.

Creemos nuevamente la tabla Agenda y veamos lo que ocurre:

```
CREATE TABLE Agenda
(
  nick VARCHAR2(25),
  numero_telefonico VARCHAR2(8),
  nota VARCHAR2(100),
  FOREIGN KEY
    (numero_telefonico)
  REFERENCES
    Directorio_telefonico(numero_telefonico)
);
```

Ingresamos un par de registros.

```
INSERT INTO Agenda
VALUES
  ('Juanito',1234,'Juanito dice hola');

SELECT * FROM AGENDA;
```

NICK	NUMERO_TELEFONICO	NOTA
Juanito	1234	Juanito dice hola

Ejecutemos el comando `TRUNCATE` sobre la tabla, de la siguiente manera:

```
TRUNCATE TABLE Agenda;
```

La respuesta es:

```
Updated Rows      0
Query TRUNCATE TABLE Agenda
Finish time Thu Sep 10 23:16:08 BRT 2020
```

Al consultar sobre la tabla `Agenda`, nos muestra lo siguiente:

```
select * from Agenda;

NICK|NUMERO_TELEFONICO|NOTA|
----|-----|----|
```

Cargar consultas desde un fichero

A veces tenemos una gran cantidad de sentencias que queremos realizar, y estar tipeando una a una puede ser algo engorroso, sumado a la pérdida de tiempo, y a que se está muy propenso a cometer pequeños errores.

Es por eso que podemos crear ficheros con extensión `.sql` que nos permitirá escribir todos nuestros comandos SQL para poder cargarlos.

La sintaxis para realizar esta operación es:

- Desde un cliente gráfico:

```
Open --> File --> /ubicaciOnDelArchivo/tuArchivo.sql.
```

- Desde SQLPLUS.

```
ubicaciOnDelArchivo/tuArchivo.sql
```

Donde `tuArchivo.sql`, es el archivo donde dejaste tus instrucciones guardadas, aunque DBeaver las guarda automáticamente.

Observaciones finales

Como observación final debemos tener en cuenta que Oracle cuenta con un sistema de seguridad, este es para confirmar una acción y evitar que borremos - por ejemplo - una tabla muy importante del sistema y no la podamos recuperar.

Para confirmar nuestras acciones en una instrucción o en un conjunto de instrucciones, ejecutaremos el siguiente comando. Recuerden, que una vez confirmada todas nuestras acciones, no hay vuelta atrás.

```
-- Confirma toda acción realizada.  
commit;
```

Podemos ver también que en ocasiones podemos usar varchar en vez de varchar2, usar la segunda forma, es muchísimo más óptima, así que a tener esto en cuenta. Para el caso de usar INT en vez de NUMBER, no presenta desventajas, ya que son el mismo tipo a menos que NUMBER especifique por ejemplo puntos flotantes.