Resumen SQL

Antonio Espín Herranz

SQL

- Comandos para la **definición y creación** de una base de datos (create table).
- Comandos para **inserción, borrado o modificación** de datos (insert, delete, update).
- Comandos para la **consulta** de datos seleccionados de acuerdo a criterios complejos que involucran diversas tablas relacionadas por un campo común (select).
- Capacidades aritméticas: En SQL es posible incluir operaciones aritméticas, así como comparaciones, por ejemplo: `A > B + 3`.
- **Asignación y comandos de impresión**: es posible imprimir una tabla construida por una consulta o almacenarla como una nueva tabla.
- **Funciones de agregación**: Operaciones tales como promedio (average), suma (sum), máximo (max), etc. se pueden aplicar a las columnas de una tabla para obtener una cantidad única y, a su vez, incluirla en consultas más complejas.

Componentes del SQL

- El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

- **SQL**: Structure Query Language

- **DML**: Lenguaje de manipulación de datos
  - **SELECT** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
  - **INSERT** Utilizado para cargar lotes de datos en la base de datos en una única operación.
  - **UPDATE** Utilizado para modificar los valores de los campos y registros especificados
  - **DELETE** Utilizado para eliminar registros de una tabla de una base de datos
- **DDL**: Lenguaje de Definición de datos
  - Los que permiten crear y definir nuevas bases de datos, campos e índices.
  - **CREATE** Utilizado para crear nuevas tablas, campos e índices
  - **DROP** Empleado para eliminar tablas e índices
  - **ALTER** Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

SQL: Claúsulas

- Las cláusulas son condiciones utilizadas para concretar que datos son los que se desea seleccionar o manipular.
- **FROM** Utilizada para especificar la tabla de la cual se van a seleccionar los registros
- **WHERE** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
- **GROUP BY** Utilizada para clasificar los registros seleccionados en grupos específicos
- **HAVING** Utilizada para expresar la condición que debe satisfacer cada grupo
- **ORDER BY** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

- ***Select columnas***
- ***From tabla***
- ***Where condición de filas***
- ***Group by columnas***
- ***Having condición grupo***
- ***Order by columnas***

SQL: Operadores

- **Lógicos**
  - **AND** Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
  - **OR** Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
  - **NOT** Devuelve el valor contrario de la expresión

- **Relacionales**:
  - < Menor que
  - > Mayor que
  - <> Distinto de
  - <= Menor o Igual que
  - >= Mayor o Igual que
  - = Igual que
  - **BETWEEN** Utilizado para especificar un intervalo de valores.
  - **LIKE** Para la comparación de una cadena de texto con una expresión regular

SQL: Otros operadores

- Is distinct from
- Is not distinct from
- Is null
- Is not null
- Isnull
- Notnull
- Is true
- Is not true
- Is false
- Is not false

- Documentación: https://www.postgresql.org/docs/16/functions-comparison.html

SQL Otros operadores

- **|| para concatenar texto**:
- Select nombre || ‘ ‘ || apellidos from clientes;

- Operadores aritméticos:
- + - * / %
- ^ potencia
- Select @ -5.0; → 5.0 Valor absoluto.

SQL Otros operadores

- Operadores a nivel de Bits:
- & and
- | or
- # xor
- ~ not
- << desplazamiento a la izq. Equivale a mul. Por 2 elevado num Bits.
- >> desplazamiento a la der. Equivale a div. Por 2 elevado num Bits.

SQL Funciones de Agregado

- Las funciones de agregación se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

- **AVG** Utilizada para calcular el promedio de los valores de un campo determinado
- **COUNT** Utilizada para devolver el número de registros de la selección
- **SUM** Utilizada para devolver la suma de todos los valores de un campo determinado
- **MAX** Utilizada para devolver el valor más alto de un campo especificado
- **MIN** Utilizada para devolver el valor más bajo de un campo especificado
- **STDDEV** La desviación típica

SQL consultas de selección

- La sintaxis básica de una consulta de selección es la siguiente:
- **SELECT** *campos* **FROM** *tabla*;
- En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:
- **SELECT nombre,x,y,z FROM observatorios;**
- Esta consulta devuelve una tabla con el campo nombre y teléfono de la tabla clientes. La tabla devuelta no está almacenada en la base de datos, y por tanto no podrá ser objeto de posteriores consultas, salvo que la guardes de forma explícita con la orden **SELECT INTO**.
- **SELECT nombre,x,y,z INTO resumen FROM observatorios** de esta manera se genera una nueva tabla que contiene sólo las cuatro columnas seleccionadas.

SQL Ordenar registros

- Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:
- **SELECT nombre,x,y,z FROM observatorios ORDER BY z;**
- Esta consulta devuelve los nombres de los observatorios junto a sus coordenadas, pero ahora ordenados en función de su altitud.
- Se pueden ordenar los registros por más de un campo, cómo, por ejemplo:
- **SELECT nombre,x,y,z FROM observatorios ORDER BY x,y;**

- **O por la posición del campo:**
- select id, **cargo** from tbpedidos order by **2** desc;

SQL: Consultas con predicado

- Una manera de limitar el número de filas que devuelve el servidor es utilizar predicados en la selección. El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

- ***** Devuelve todos los campos de la tabla. En este caso el servidor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL.
- **SELECT * FROM observatorios;**

- No es conveniente abusar de este predicado ya que obligamos al servidor a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.
- **SELECT indentinm,x,y,z,nombre FROM observatorios;**

SQL: Consultas con predicado

- **DISTINCT** Omite los registros cuyos campos seleccionados coincidan totalmente. Con otras palabras, el predicado
- DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un
- contenido diferente.
- **SELECT DISTINCT indentinm,x,y,z,nombre FROM observatorios;**

- **DISTINC ON (*campo*)** Omite registros que coincidan en el campo seleccionado. Por ejemplo, la siguiente orden devuelve un sólo observatorio por valor de altitud:
- **SELECT DISTINCT ON (z) indentinm,x,y,z,nombre FROM observatorios;**

SQL: Alias

- En determinadas circunstancias es necesario asignar un nuevo nombre a alguna de las columnas devueltas por el servidor. Para ello tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada:
- **SELECT nombre,x AS longitud, y AS latitud, z AS altitud FROM observatorios;**

- Los alias no se pueden utilizar en las condiciones:
- select id, cargo, cargo * 0.21 as **iva** from tbpedidos where ~~iva~~ < 100;

SQL Criterios de selección

- La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT.

- Por ejemplo, para obtener sólo los observatorios situados a más de 500 metros de altitud, la consulta adecuada sería:
- **SELECT nombre,x,y,z FROM observatorios WHERE z > 500;**

SQL Operadores Lógicos

- Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis:
- <expresión1> operador <expresión2>
- En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:
- Falso AND Verdad **Falso**
- Falso AND Falso **Falso**
- Verdad OR Falso **Verdad**
- Verdad OR Verdad **Verdad**
- Falso OR Verdad **Verdad**
- Falso OR Falso **Falso**

SQL Operadores Lógicos

- Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.
- **SELECT nombre,x,y,z FROM observatorios WHERE x > 600000 AND x < 650000;**
- **SELECT nombre,x,y,z FROM observatorios WHERE (x > 600000 AND x < 650000) OR z<200;**
- La última consulta devolverá los observatorios situados entre los valores de X UTM de 600000 y 650000 UTM y aquellos con altitud inferior a 200 metros.

SQL Intervalos de Valores

- Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es: campo **[Not] Between** valor1 **And** valor2
- En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo:
- **SELECT nombre,x,y,z FROM observatorios WHERE x Between 600000 AND 650000;**

- esta orden es equivalente a **SELECT nombre,x,y,z FROM observatorios WHERE x > 600000 AND Edad < 650000;**

SQL: Operador Like

- Productos que se envasan en cajas:
  - select nombre, cantidadporunidad from tbproductos where cantidadporunidad **like** '**%**cajas**%**';
- **Not like**
  - La negación del anterior

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', ' '	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

SQL Expresiones Regulares

- Similar to
  - String **similar to** pattern

- **Metacaracteres (para los patrones):**

|	Alternativa
*	0 o más repeticiones
+	1 o más repeticiones
?	0 o 1 repetición
{m}	m repeticiones
{m,}	m o más repeticiones
{m,n}	entre m y n repeticiones
()	Para crear grupos
[]	Especificar clases y rangos

- Documentación: https://www.postgresql.org/docs/16/functions-matching.html#FUNCTIONS-SIMILARTO-REGEXP

SQL Operador IN

- Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:
- expresión **[Not] In(**valor1, valor2, . . .**)**
- **SELECT * FROM observatorios WHERE indentinm IN(7149,7069);**

- **Pueden ser cadenas:**
- **Select * from pedidos where país in (‘Alemania’, ‘Suiza’, ‘Francia’);**

Is Null / is not null

- Buscar campos vacíos o no:
- select * from tbpedidos where fechaenvio is null;

SQL Limitar el número de registros

- Cuando ejecutamos una consulta se pueden limitar el número de resultados.

- Select campos from tabla LIMIT número

- SELECT nombre,z FROM observatorios ORDER BY z DESC **LIMIT 5**;

SQL Agrupar registros

- Funciones de agregado (con **Group by**):
  - Count(* | distinct atributo) → cuenta.
  - Sum(Atributo) → Suma los valores.
  - Avg(Atributo) → Media.
  - Max(Atributo) → Máximo.
  - Min(Atributo) → Mínimo.
  - Stddev: Desviación estándar

- Select codDpto, count(*) from Empleados group by CodDpto.

SQL Agrupar registros

- Otro uso típico de Group By es para quitar los registros repetidos de una tabla.
  - Select nombre from alumnos group by nombre.

- En este caso si en la tabla alumnos hubiera repetidos sólo los mostraría una vez.

SQL Agrupar registros

- **Having**: Establecer condiciones para los registros de un grupo.
- Where lo utilizamos para filtrar filas de una tabla.
- En caso de tener grupos para establecer alguna condición tenemos que utilizar Having.
- Por ejemplo:
  - Obtener los registros duplicados de una tabla.

```
Select nombre from alumnos
group by nombre
having count(*) > 1
```

SQL Agrupar registros

- Varias tablas:
  - **Select** Empleados.CodDpto, Descripcion, sum(sueldo) from **Empleados, Departamentos**

    Where Empleados.CodDpto = Departamentos.CodDpto

    a**nd** Empleados.CodDpto = Departamentos.CodDpto

    Group by Empleados, Descripcion.

Consultas varias tablas

- **cross join**: Producto cruzado. Combina cada registro de la primera tabla con cada registro de la tabla relacionada.
- **inner join:** Unión normal. Muestra sólo registros de ambas tablas que estén relacionados.
- **left join:** Muestra todos los registros de la primera tabla y sólo los registros relacionados en la segunda.
- **right join:** Muestra todos los registros de la segunda tabla y sólo los registros relacionados en la primera.

Consultas varias tablas

- Resolver preguntas de este estilo:

- Dos equipos (de futbol y baloncesto) y queremos saber
  - Quien se ha apuntado sólo a futbol,
  - Quien se ha apuntado sólo a baloncesto.
  - Quien se ha apuntado a ambos equipos.

- Hacer el ejemplo.

SELECT * FROM a
INNER JOIN b ON a.key = b.key

SELECT * FROM a
LEFT JOIN b ON a.key = b.key

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key

POSTGRESQL JOINS

POSTGRESQL TUTORIAL

SELECT * FROM a
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL

SELECT * FROM a
FULL JOIN b ON a.key = b.key

SELECT * FROM a
FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL

SQL Conjuntos

- Union, Intersect y Except (diferencia)

- Sintaxis:
  - query1 UNION [ALL] query2
  - query1 INTERSECT [ALL] query2
  - query1 EXCEPT [ALL] query2

SQL Conjuntos

```
SELECT select_list_1
FROM table_expresssion_1
UNION
SELECT select_list_2
FROM table_expression_2
```

Subconsultas: Any, In, Some, All, Exists

- **operand comparison_operator ANY (subquery)**
  - La palabra clave **ANY** , que debe seguir a un operador de comparación, significa “return TRUE si la comparación es TRUE para ANY (cualquiera) de los valores en la columna que retorna la subconsulta.
  - SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);

Subconsultas: Any, In, Some, All, Exists

- **operand IN (subquery) :**

  Si el valor está dentro del subconjunto.

  Select * From empleados

  Where codDpto in (select codDpto

  From Departamentos where (situacion = ‘3 planta’)

Subconsultas: Any, In, Some, All, Exists

- **operand comparison_operator SOME (subquery)**
- La palabra **SOME** es un alias para ANY. Por lo tanto, estos dos comandos son el mismo:
  - SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
  - SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);

Subconsultas: Any, In, Some, All, Exists

- *operand comparison_operator* ALL (*subquery*)

- La palabra **ALL**, que debe seguir a un operador de comparación, significa “return TRUE si la comparación es TRUE para ALL todos los valores en la columna que retorna la subconsulta.”

- Por ejemplo:
  - SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);

Subconsultas: Any, In, Some, All, Exists

- El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

- SELECT * FROM observatorios WHERE EXISTS (SELECT z FROM observatorios WHERE nombre ’Lorca’);

- Devolverá todos los registros porque en la base de datos existen observatorios cuyo nombre incluye la palabra ’Lorca’.

- SELECT * FROM observatorios WHERE EXISTS (SELECT z FROM observatorios WHERE nombre ’Pamplona’);

SQL: Consultas de Acción

- Eliminación **Delete**: Elimina los registros que cumplen los criterios.

- Sintaxis:
  - DELETE FROM Tabla WHERE criterios

- Ejemplo:
  - DELETE FROM Empleados WHERE Cargo = 'Vendedor';

SQL: Consultas de Acción

- Cuando utilizamos delete para eliminar registros de una tabla y esa tabla tiene un campo secuencia si después del borrado damos de alta algún registro los valores de este campo no se reutilizan para los nuevos registros.

- Ver ejemplo.

SQL: Consultas de Acción

- Disponemos de los siguientes registros:
- Y realizamos las siguientes operaciones:
  - Delete from Departamentos where id = 2;
  - Insert into Departamentos (departamento) values (‘Exportación’);

Id	Departamento
1	Compras
2	Ventas
3	Contabilidad

Id	Departamento
1	Compras
3	Contabilidad
4	Exportación

Cuando realizamos una operación de inserción los campos autoincrement se generan de forma automática.

SQL: Consultas de Acción

- Postgre no soporta “delete join” pero si se puede utilizar **using** en una claúsula delete.

- DELETE FROM tabla1 USING tabla2 WHERE tabla1.campo = tabla2.campo

SQL Consultas de Acción

- Se pueden utilizar subconsultas para eliminar registros:

- Delete from tabla1 Where campo in (select campo from tabla2)

SQL: Consultas de Acción

- Eliminación **Truncate**: Elimina todas las filas y los campos autoincrement los vuelve a inicializar.

- Sintaxis:
  - TRUNCATE tabla

- Ejemplo:
  - TRUNCATE departamentos [**RESTART IDENTITY**]
    - Cuando queremos reiniciar las secuencias

SQL: Consultas de Acción

- Inserción de datos: **Insert into**

- Sintaxis a)
  - **INSERT INTO** Tabla (campo1, campo2, .., campoN) **VALUES** (valor1, valor2, ..., valorN)

- Sintaxis b)
  - **INSERT INTO** Tabla (campo1, campo2, ..., campoN) **SELECT** TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN **FROM** TablaOrigen

SQL: Consultas de Acción

- Ejemplos:
  - INSERT INTO Clientes.Cliente_Nuevos SELECT Clientes_Viejos.* FROM Clientes_Viejos;

  - INSERT INTO Empleados (Nombre, Apellido, Cargo)
    VALUES ('Luis', 'Sánchez', 'Becario');

- También soporta esta sintaxis cuando queremos insertar más de un registro con una sentencia.

```
INSERT INTO `gruposcursos` (`id`, `descripcion`) VALUES
(1, 'Java'), (2, 'Seminarios Prácticos'), (3, 'Visual Basic'),
(4, 'Bases de Datos'));
```

SQL: Consultas de Acción

- Cuando tenemos un campo serial dentro de la tabla no hace falta indicarlo.
- insert into tabla(otro_campo) values (valor);

- El campo serial se incrementa automáticamente y no recupera los valores si se elimina algún registro.

SQL: Consultas de Acción

- Actualización Update:

- Sintaxis:
  - **UPDATE** Tabla **SET** Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN
    WHERE Criterio;

- **EJEMPLOS:**
  - UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte = Transporte * 1.03 WHERE PaisEnvío = 'ES';

SQL Consultas de Acción

```
UPDATE product p  ← Tabla 1
SET  net_price = price - price * discount
FROM  product_segment s  ← Tabla 2
WHERE   p.segment_id = s.id;
```

SQL With

- Permite definir identificadores que se sustituyen por una cadena, y luego los utilizamos en la siguiente consulta:

- Ejemplo:

```
with pedidos_suiza as
(select * from tbpedidos where paisdestinatario='Suiza')
select sum(cargo) as total from pedidos_suiza;
```

SQL Prepare

- Para crear consultas parametrizadas:
- **Prepare** nombre_prepare(tipo1, tipo2, …) as
- Consulta: insert into, select, update, delete ,,,
- Los parámetros se numeran de **$1** … **$n**
- Para ejecutar: execute nombre_prepare([parámetros])
- Para borrar un prepare: **deallocate** [all] nombre_prepare

SQL: DDL

- Lenguaje para definir objetos de la BD.
- Hay 3 instrucciones principal: **Create**, **Drop** y **Alter**
- Por ejemplo:
  - Create table tabla1 (…);
  - Drop table tabla1;
  - Alter table tabla1 …

SQL DDL

- Creación y borrado de forma condicional.
- Create table if not exists nombreTabla …
- Drop table if exists nombreTabla;

- Renombrar una tabla:
  - *alter table nombre_tabla rename to nombre_tabla2;*

SQL: DDL

- *Create table nombre-tabla as query*

- Con el resultado de la consulta se crea una nueva tabla.
- Las columnas de la consulta se crean en la nueva tabla.

SQL:DDL

CREATE DATABASE db_name

OWNER = role_name

TEMPLATE = template

ENCODING = encoding

LC_COLLATE = collate

LC_CTYPE = ctype

TABLESPACE = tablespace_name

CONNECTION LIMIT = max_concurrent_connection

SQL: DDL

- Nombre_bd. El nombre de la BD.
- Nombre del rol: El propietario de la BD, por defecto postgres
- Plantilla: Se puede especificar el nombre de la plantilla para crear la nueva BD.
- Codificación: Permite especificar el juego de caracteres.
- Tablespace_name: El nombre del tablespace: sirve para especificar ubicaciones distintas a pg_default,
- Collate: Orden de clasificación de las cadenas.
- Type_c: clasificación de caracteres para la nueva BD.
- Límite de conexiones: -1, sin límite

SQL: DDL

```
CREATE TABLE [IF NOT EXISTS] table_name (
   column1 datatype(length) column_contraint,
   column2 datatype(length) column_contraint,
   column3 datatype(length) column_contraint,
   table_constraints
);
```

SQL: DDL

- Constraints:
  - **NOT NULL** : garantiza que los valores de una columna no puedan ser NULL.
  - **UNIQUE** : garantiza que los valores de una columna sean únicos en todas las filas de la misma tabla.
  - **PRIMARY KEY**: una columna de clave principal identifica de forma única las filas de una tabla. Una tabla puede tener una y sólo una clave primaria. La restricción de clave principal le permite definir la clave principal de una tabla, pero se puede formar con varias columnas.
  - **CHECK** : una restricción que garantiza que los datos deben satisfacer una expresión booleana.
  - **FOREIGN KEY**: garantiza que los valores de una columna o un grupo de columnas de una tabla existan en una columna o grupo de columnas de otra tabla. A diferencia de la clave principal, una tabla puede tener muchas claves externas.

SQL: DDL

- Constraints ejemplos:
- Un campo clave primaria y serial
  - Campo serial primary key

- Una clave primaria compuesta:
  - Campo1 tipo1 not null,
  - Campo2 tipo2 not null,
  - Primary key(campo1, campo2)

SQL: DDL

- Constraints ejemplos:

- Una clave foránea:
  - Campo1 tipo1 not null,
  - Foreign key (campo1) references otra_tabla (campo_otra_tabla) [on delete (cascade | restrict | set null)]

SQL: DDL

- Se pueden indicar:
  - **Cascade**: Borra o actualiza en la tabla padre y en las tablas relacionadas.
  - **Restrict**: Rechaza la operación de actualización / borrado.
  - **Set null**: Borra o actualiza en la tabla padre y establece a null en las hijas.
  - **No action**: No hace nada.

SQL: DDL

- Las constraints también se pueden añadir a posteriori cuando la tabla ya está creada.

- Para ello utilizamos alter table sobre la tabla a modificar.

- Ejemplo:

```
ALTER TABLE some_child
ADD CONSTRAINT parent_id_fk
FOREIGN KEY (parent_id) REFERENCES parent(id)
ON DELETE SET NULL;
```

SQL DDL

- Se puede escribir la definición completa de la BD en un script (utilizar **query tool** seleccionando previamente la BD).

- Al ejecutar el script se ejecuta sobre la BD seleccionada.

- Hay que crear previamente la BD que las tablas. Primero creamos la BD y luego se ejecutan las sentencias DDL.

SQL: DDL

- Tipos de datos:
  - Para los campos autoincrement se pueden utilizar secuencias o tipos serial.
  - CREATE TABLE tablename (colname SERIAL);

  - Es equivalente a:
    - CREATE SEQUENCE tablename_colname_seq;
    - CREATE TABLE tablename (
    - colname integer NOT NULL DEFAULT nextval('tablename_colname_seq'));
    - ALTER SEQUENCE tablename_colname_seq OWNED BY tablename.colname;

  - Texto → varchar(n)
  - Enteros → int / integer

SQL: DLL

- Resumen de los tipos más utilizados en PostgreSQL
- **Tipos Numéricos**:
  - **Números enteros**, disponemos tres data type dependiendo del rango de números que vayamos a almacenar. El tipo de datos más típico es el integer, más conocido como int.
  - **Números decimales**, disponemos de cuatros tipos de este fragmento según los decimales que queramos establecer.
  - **Seriales**, dependiendo de la longitud de registros que vayamos a tener en nuestra tabla podemos utilizar tres tipos de seriales, éstos son valores autoincremental.

SQL: DDL

- **Tipos de caracteres**:
  - Disponemos de tres tipos para almacenar cadenas dependiendo del número de caracteres que queramos contener.
  - Tenemos dos tipos de longitud fija, **character varying(n) y character(n),** más utilizados como **varchar(n) y char(n)** respectivamente.
  - El otro tipo para almacenar cadenas es el **text**, éste último permite contener cadenas de longitud ilimitada.

SQL: DDL

- **Tipos de fechas (Date / Time)**
  - Otro de los tipos más utilizados son los de tipo de fecha y hora.
  - PostgreSQL nos permite separar la fecha y la hora principalmente en dos tipos, date Type para sólo la fecha y time Type para sólo la hora.
  - También podemos obtener la fecha y la hora a la vez en un único tipo, con o sin la zona horaria éste tipo es llamado timestamp. Disponemos de un tipo interval con el que podemos establece un intervalo temporal, por ejemplo los años, meses, etc.

SQL: DDL

- **Tipos boolean**
  - Este tipo de dato es utilizado para evaluar un estado en verdadero o falso según la condición que necesitamos.

a	*b*	*a* **AND** *b*	*a* **OR** *b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

dblink

- Para realizar consultas entre varias bases de datos

- O hacer consultas a una BD remota.
- Lo primero activar la extensión **dblink.**
  - **Create extension dblink;**
  - Si la extensión se activa desde psql y tenemos abierto pgAdmin, mejor cerrarlo y volverlo abrir (puede que no se active hasta que se abra de nuevo).
  - La extensión se activa a nivel de BD

dblink

- Cuando realizamos una consulta con dblink tenemos que indicar los parámetros de la conexión ya sea a partir de un identificado (con dblink_connect) o directamente en dblink.

- A la consulta que traemos con dblink hay que ponerla un alias e indicar el nombre y el tipo de las columnas de la consulta.

dblink

- Funciones:
  - **dblink**: para realizar una consulta
  - **dblink_connect**: para crear una conexión
  - **dblink_disconnect**: para cerrar una conexión

  - Para las conexiones tenemos que indicar una serie de parámetros.

dblink

- Parámetros:
  - **host**: nombre o dirección IP. Si no lo indicamos tomará el servidor actual: localhost o la 127.0.0.1
  - **dbname**: El nombre de la BD
  - **user**: El usuario con el que vamos a conectar.
  - **port**: Número de puerto (por defecto: 5432)
  - **password**: password del usuario.

Ejemplo

- select * from dblink(‘host=localhost port=5432 dbname=empresa3 user=postgres password=pass_user options=-csearch_path=‘, 'select id,nombre from public.tbcategorias’) as t1(id integer, nombre varchar);

dblink

- Crear una conexión con nombre:
  - select dblink_connect(**'conexion1**','dbname=empresa3 user=postgres password=user_pass');

- Consulta a través de la conexión:
  - *select * from dblink(**'conexion1**', 'select * from tbcategorias') as t1(id integer, nombre varchar);*

- *Cerrar la conexión:*
  - *select dblink_disconnect(**'conexion1**');*

dblink

- create schema temporal1;
- create table temporal1.tbidpedidos as (select t3.id

from dblink('conexion1','select id from public.tbpedidos')

as t3(id integer));

- Se puede crear una tabla trayendo los registros de la conexión remota y luego realizar las consultas en local para que sea más rápido.