

BASE DE DATOS SQL

Cuestiones previas

Antes de empezar con la sentencia SELECT, es necesario tener una tabla disponible sobre la cual vamos a aplicar la sentencia.

Vamos a crear un DATABASE, una TABLA, las COLUMNAS y REGISTROS que nos ayudará a repasar conceptos de la clase anterior. El esquema es el siguiente:

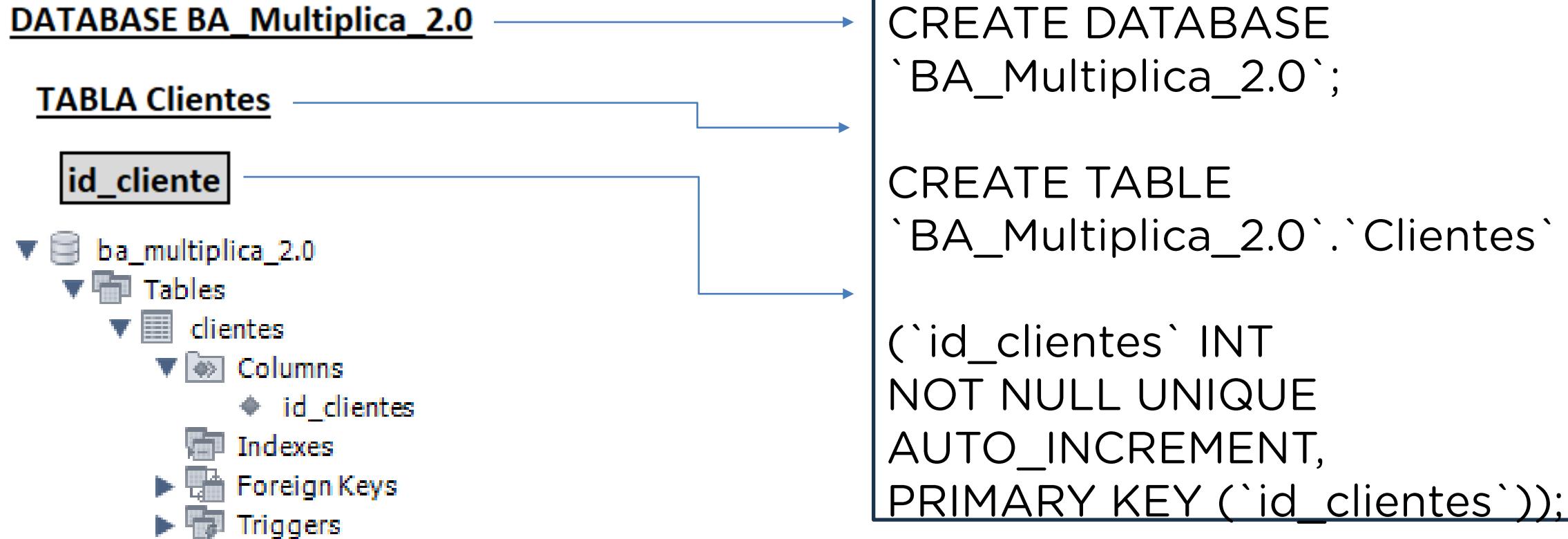
DATABASE BA_Multiplica_2.0

TABLA Clientes

id_cliente	nombre_cliente	apellido_cliente	edad	genero
1	Cintia	Cruz	16	Femenino
2	Vanesa	Rojas	25	Femenino
3	Omar	Bracho	32	Masculino
4	Mauro	Resquin	48	Masculino
5	Sofia	Gramajo	60	Femenino

Cuestiones previas

Vamos a crear el DATABASE “BA_Multiplica_2.0” y luego la TABLA “clientes” con la COLUMNA “id_cliente” con las restricciones que sabemos que deben tener los campos ID’s. Si bien son 2 querys individuales, pueden ejecutarse todas juntas si están en la secuencia correcta. Entonces nos quedaría:



Cuestiones previas

Vamos a crear la columna “nombre_cliente” que sabemos es un dato de tipo texto que no pueden ser valores nulos. Entonces nos quedaría:

DATABASE BA_Multiplica_2.0

TABLA Clientes

<u>id_cliente</u>	<u>nombre_cliente</u>
-------------------	-----------------------

▼ ba_multiplica_2.0

 ▼ Tables

 ▼ clientes

 ▼ Columns

 ◆ id_cliente

 ◆ nombre_cliente

 ▼ Indexes

 ► Foreign Keys

 ► Triggers

ALTER TABLE
`BA_Multiplica_2.0`.`clientes`
ADD COLUMN `nombre_cliente`
VARCHAR(100) NOT NULL;

Nota: Para evitar errores de contexto,
es recomendable referenciar el
“DATABASE.” seguido por el nombre de
la tabla que se quiere modificar.

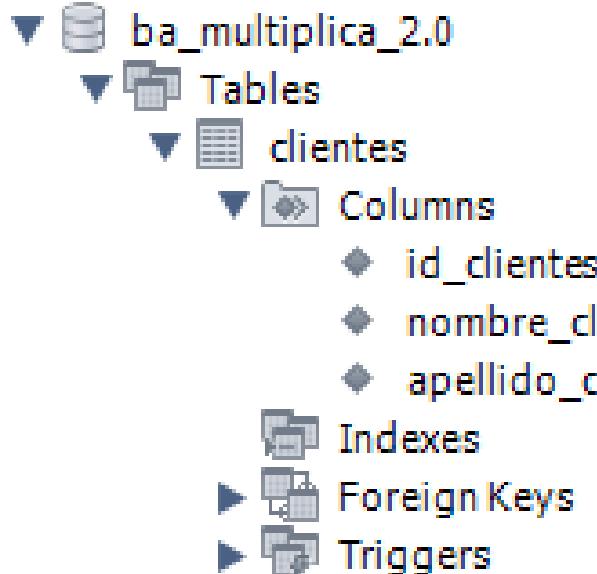
Cuestiones previas

Vamos a crear la columna “apellido_cliente” que sabemos es un dato de tipo texto que no pueden ser valores nulos. Entonces nos quedaría:

DATABASE BA_Multiplica_2.0

TABLA Clientes

<u>id_cliente</u>	<u>nombre_cliente</u>	<u>apellido_cliente</u>
-------------------	-----------------------	-------------------------



```
ALTER TABLE  
`BA_Multiplica_2.0`.`clientes`  
ADD COLUMN `apellido_cliente`  
CHAR(255) NOT NULL;
```

Nota: En este caso, ponemos el tipo texto CHAR que admite como máximo 255 caracteres.

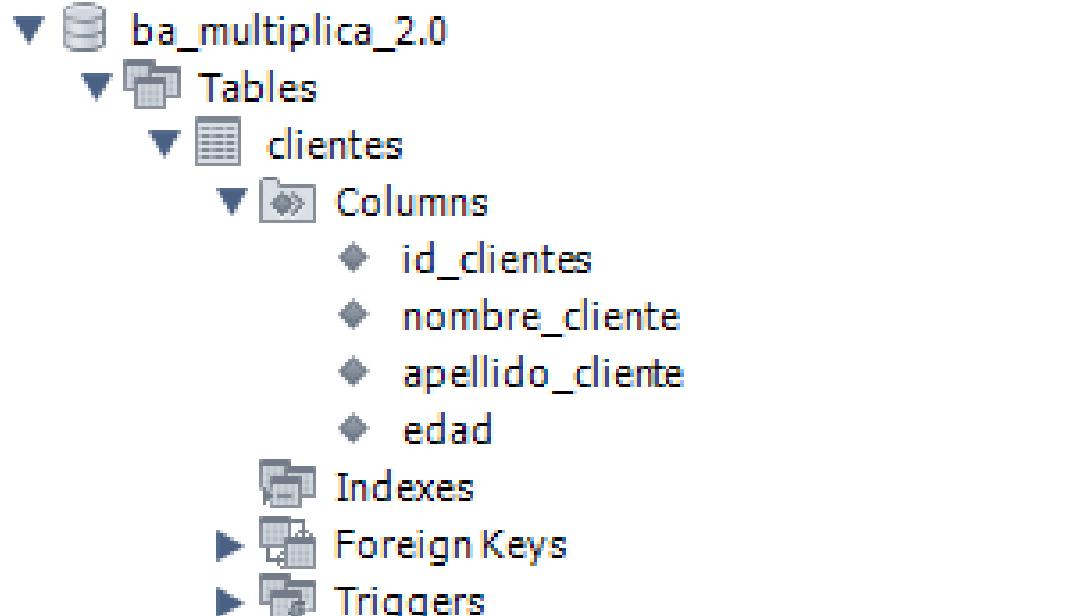
Cuestiones previas

Vamos a crear la columna “edad” que sabemos es un dato de tipo entero que no pueden ser valores nulos. Entonces nos quedaría:

DATABASE BA_Multiplica_2.0

TABLA Clientes

id_cliente	nombre_cliente	apellido_cliente	edad
-------------------	-----------------------	-------------------------	-------------



```
ALTER TABLE  
`BA_Multiplica_2.0`.`clientes`  
ADD COLUMN `edad` INT (3) NOT  
NULL;
```

Nota: después de definir el tipo de dato, establecemos 3 caracteres como máximo ya que sería 999

Cuestiones previas

Vamos a crear la columna “edad” que sabemos es un dato de tipo entero que no pueden ser valores nulos. Entonces nos quedaría:

DATABASE BA_Multiplica_2.0

TABLA Clientes

<u>id_cliente</u>	<u>nombre_cliente</u>	<u>apellido_cliente</u>	<u>edad</u>	<u>genero</u>
-------------------	-----------------------	-------------------------	-------------	---------------

▼ ba_multiplica_2.0

 ▼ Tables

 ▼ clientes

 ▼ Columns

- ◆ id_clientes
- ◆ nombre_cliente
- ◆ apellido_cliente
- ◆ edad
- ◆ genero

 ▼ Indexes

 ▶ Foreign Keys

 ▶ Triggers

ALTER TABLE
`BA_Multiplica_2.0`.`clientes`
ADD COLUMN `genero`
VARCHAR (100) NOT NULL;

Con las columnas creadas,
podemos insertar los registros
por QUERY o manualmente

Cuestiones previas

Creemos una QUERY que inserte los primeros 4 registros, recordando la estructura que debe tener la sentencia:

SENTENCIA + DATABASE . TABLA + COLUMNAS + VALORES A CARGAR

```
INSERT INTO `BA_Multiplica_2.0`.`clientes` (`nombre_cliente`, `Apellido_cliente`, `edad`, `genero`) VALUES ('Cintia', 'Cruz', '16', 'Femenino');
INSERT INTO `BA_Multiplica_2.0`.`clientes` (`nombre_cliente`, `Apellido_cliente`, `edad`, `genero`) VALUES ('Vanesa', 'Rojas', '25', 'Femenino');
INSERT INTO `BA_Multiplica_2.0`.`clientes` (`nombre_cliente`, `Apellido_cliente`, `edad`, `genero`) VALUES ('Omar', 'Bracho', '32', 'Masculino');
INSERT INTO `BA_Multiplica_2.0`.`clientes` (`nombre_cliente`, `Apellido_cliente`, `edad`, `genero`) VALUES ('Mauro', 'Resquin', '48', 'Masculino');
```

id_cliente	nombre_cliente	apellido_cliente	edad	genero
1	Cintia	Cruz	16	Femenino
2	Vanesa	Rojas	25	Femenino
3	Omar	Bracho	32	Masculino
4	Mauro	Resquin	48	Masculino
5	Sofia	Gramajo	60	Femenino

Nota: Como la columna “id_cliente” la definimos como autoincremental, no es necesario insertarle los valores. Comenzará a asignar un valor desde el 1 hacia adelante.

Cuestiones previas

Para observar los datos ingresados, podemos presionar el botón marcado para que se genere automáticamente la QUERY “`SELECT * FROM 'ba_multiplica_2.0'.'clientes'`”

The screenshot shows the MySQL Workbench interface with three main panes:

- Navigator:** Shows the database schema. A red box highlights the 'clientes' table under the 'Tables' section of the 'ba_multiplica_2.0' schema.
- Information:** Displays details about the 'clientes' table.
 - Table: clientes**
 - Columns:**

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
	int(11) AI PK	varchar(100)			
		char(255)			
		int(3)			
		varchar(100)			
- Result Grid:** Shows the data from the 'clientes' table.

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
1		Cintia	Cruz	16	Femenino
2		Vanesa	Rojas	25	Femenino
3		Omar	Bracho	32	Masculino
4		Mauro	Resquin	48	Masculino
**	NULL	NULL	NULL	NULL	NULL

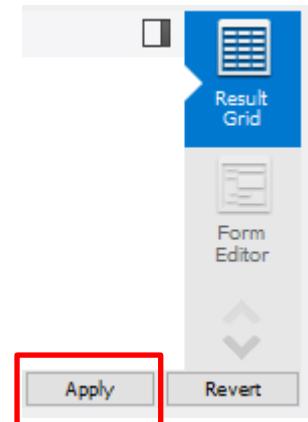
A continuación, vamos a insertar el 5to registro agregándolo manualmente en el panel donde están ubicados los datos.

Cuestiones previas

Nos ubicamos en la siguiente fila, ingresando en cada celda el dato correspondiente. Recordar que “id_clientes” es autoincremental así que no es necesario cargarlo.

Result Grid					
	id_clientes	nombre_cliente	apellido_cliente	edad	genero
1	Cintia	Cruz	16	Femenino	
2	Vanesa	Rojas	25	Femenino	
3	Omar	Bracho	32	Masculino	
4	Mauro	Resquin	48	Masculino	
*	NULL	Sofia	Gramajo	60	Femenino
*	NULL	NULL	NULL	NULL	NULL

Para que el registro se inserte, tenemos que presionar el botón Apply ubicado a la derecha del panel



Finalmente tenemos ambas bases iguales.

id_cliente	nombre_cliente	apellido_cliente	edad	genero
1	Cintia	Cruz	16	Femenino
2	Vanesa	Rojas	25	Femenino
3	Omar	Bracho	32	Masculino
4	Mauro	Resquin	48	Masculino
5	Sofia	Gramajo	60	Femenino

id_clientes	nombre_cliente	apellido_cliente	edad	genero
1	Cintia	Cruz	16	Femenino
2	Vanesa	Rojas	25	Femenino
3	Omar	Bracho	32	Masculino
4	Mauro	Resquin	48	Masculino
5	Sofia	Gramajo	60	Femenino

Select

Se utiliza para especificar los nombres de los campos en los cuales se quiere hacer una consulta. Es la sentencia más importante que debe manejar un analista de datos ya que:

- En la mayoría de los casos un sistema genera sus reportes a través de esta sentencia.
- Se puede extraer información específica de grandes volúmenes de datos almacenados en bases de datos.
- Se pueden optimizar reportes que contienen datos que no son relevantes.
- Se puede explorar como están estructurados los datos de un reporte existente.

Podemos utilizarla especificando los nombres de los campos de una tabla o generando una tabla completa con el comando “*”

En ambos casos, luego se debe referenciar la tabla con la sentencia FROM + nombre de la tabla

Select

Si selecciono toda la tabla:

```
SELECT *  
FROM clientes
```

	id_cliente	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

Si selecciono 2 campos:

```
SELECT id_cliente, nombre_cliente  
FROM clientes
```

	id_cliente	nombre_cliente
▶	1	Cintia
	2	Vanesa
	3	Omar
	4	Mauro
	5	Sofia
●	NULL	NULL

Podemos utilizarla especificando los nombres de los campos de una tabla o generando una tabla completa con el comando “*”

Select (Alias)

Al seleccionar columnas, es posible renombrarla con alias (AS) a cada una, ya sea para abbreviar nombres de columnas o aclarar de que se tratan los datos que contiene.

Creemos una query con todas las columnas en mayúsculas y sacando la palabra “cliente”:

```
SELECT
id_clientes AS ID,
nombre_cliente AS NOMBRE,
apellido_cliente AS APELLIDO,
edad AS EDAD,
genero AS GENERO

FROM
`BA_Multiplica_2.0`.`clientes`;
```

	ID	NOMBRE	APELLIDO	EDAD	GENERO
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
	5	Sofia	Gramajo	60	Femenino

Podemos utilizarla especificando los nombres de los campos de una tabla o generando una tabla completa con el comando “*”

Select (ALIAS)

Si selecciono toda la tabla

```
SELECT *  
FROM clientes
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

Si selecciono 2 campos

```
SELECT id_cliente,  
nombre_cliente  
FROM
```

	id_clientes	nombre_cliente
▶	1	Cintia
	2	Vanesa
	3	Omar
	4	Mauro
	5	Sofia
●	NULL	NULL

Podemos utilizarla especificando los nombres de los campos de una tabla o generando una tabla completa con el comando “*”

Select

Para potenciar el uso de la sentencia, se pueden agregar cláusulas para especificar la información que se requiere. Entre ellas tenemos:

WHERE

Nos permite definir filtros que debe cumplir cada uno de los registros que obtenemos en el resultado

GROUP
BY

Opera sobre conjuntos de filas para dar un resultado por grupo. Debe combinarse con una función de agregación (Tener en cuenta COUNT)

HAVING

Especifica una condición de búsqueda para un grupo. Se comporta como WHERE pero se aplica a grupos. Se utiliza para filtrar filas de un resultado ya agrupado a través de la cláusula GROUP BY.

ORDER
BY

Se utiliza en caso de necesitar obtener el resultado en un orden particular. Puede ser ascendente o descendente

Podemos utilizarla especificando los nombres de los campos de una tabla o generando una tabla completa con el comando “*”

Select – Secuencia de aplicación de las cláusulas

sentencia:

```
SELECT columnas  
FROM tabla  
WHERE condición  
GROUP BY columnas  
HAVING condición  
ORDER BY columnas [ASC | DESC]  
;
```

- **Columnas:** es una lista de una o más columnas separadas por comas. Si se utiliza el asterisco (*), se seleccionarán todas las columnas de la tabla.
- **Tabla:** es el nombre de la tabla de la cual se quieren seleccionar datos.
- **Condición:** es una expresión que se utiliza para filtrar los datos de acuerdo con ciertos criterios. Puede ser una condición simple o una condición compuesta por varias expresiones lógicas (AND, OR, NOT).
- **GROUP BY:** se utiliza para agrupar los datos en función de una o más columnas. Solo se pueden utilizar las columnas que aparecen en la lista de selección.
- **HAVING:** se utiliza para filtrar los datos después de que se han agrupado utilizando la cláusula GROUP BY.
- **ORDER BY:** se utiliza para ordenar los datos en función de una o más columnas. Se puede especificar si el orden es ascendente (ASC) o descendente (DESC).

Select - Where

La utilizamos cuando queremos aplicar un filtro a nuestra query. Vamos a generar toda la tabla con la cláusula WHERE en clientes por el género “Femenino”

WHER
E

```
SELECT * FROM `clientes` WHERE genero  
= 'Femenino'
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

GROU
P BY

HAVIN
G

ORDE
R BY

Nota: tener en cuenta que al aplicar WHERE por un campo de texto, se debe respetar las mayúsculas que tenga los registros

Select - Where (ejemplo con > y <)

Se pueden aplicar los signos “<” ”>” siempre y cuando el campo sea de tipo numérico.
Vamos a generar toda la tabla aplicandolo sobre “edad” con ambos casos:

WHER

E

GROU

P BY

HAVIN

G

ORDE

R BY

SELECT * FROM `clientes` WHERE edad > 30

	id_cliente	nombre_cliente	apellido_cliente	edad	genero
▶	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
●	5	Sofia	Gramajo	60	Femenino
•	NULL	NULL	NULL	NULL	NULL

SELECT * FROM `clientes` WHERE edad < 30

	id_cliente	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
●	NULL	NULL	NULL	NULL	NULL

Select - Where (ejemplo con BETWEEN)

Se puede utilizar cuando se necesita que el filtro se aplique con 2 condiciones. Vamos a generar toda la tabla cuyos datos sean mayores a 20 y menores a 50:

WHER
E

```
SELECT * FROM `clientes` WHERE edad BETWEEN 20 AND 50;
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	2	Vanesa	Rojas	25	Femenino
	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
●	HULL	HULL	HULL	HULL	HULL

GROU
P BY

HAVIN
G

ORDE
R BY

Primero debe colocarse el valor menor, luego de la cláusula “AND” y por último el valor mayor.

La cláusula “AND” puede aplicarse para agregar más cláusulas “WHERE”. Vamos a ver ejemplos más adelante.

Select - Where (ejemplo con IN)

Se puede utilizar cuando se necesita que el filtro sea con valores específicos. Vamos a generar toda la tabla para los clientes con “ID” 3 y 5

WHERE

```
SELECT * FROM `clientes` WHERE id_clientes IN (3, 5);
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	3	Omar	Bracho	32	Masculino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

GROU
P BY

Se puede aplicar en campos de texto estableciendo los valores con comilla simple, vamos a crearla con el campo nombre “Mauro” y “Cintia”:

HAVING

```
SELECT * FROM `clientes` WHERE id_clientes IN (3, 5);
```

ORDER
BY

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	4	Mauro	Resquin	48	Masculino
●	NULL	NULL	NULL	NULL	NULL

Select - Where (ejemplo con LIKE)

Se utiliza cuando se necesita que el filtro contenga algunos caracteres. Esta cláusula debe combinarse con el comodín "%" que nos ayudará a autocompletar los resultados.

WHER
E

Apliquemos el filtro para los resultados que comiencen con "Vane"

```
SELECT * FROM `clientes` WHERE nombre_cliente LIKE 'Vane%';
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	2	Vanesa	Rojas	25	Femenino
●	NULL	NULL	NULL	NULL	NULL

GROU
P BY

HAVIN
G

ORDE
R BY

Apliquemos el filtro para los resultados que finalicen con "ia"

```
SELECT * FROM `clientes` WHERE nombre_cliente LIKE '%ia';
```

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
●	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

Select - Where (ejemplo con LIKE)

También se puede utilizar de manera que los resultados contengan caracteres con los comodines al principio y al final. Apliquemosla con apellidos que contengan “ra”

WHERE

```
SELECT * FROM `clientes` WHERE apellido_cliente LIKE '%ra%';
```

GROU
P BY

	<code>id_clientes</code>	<code>nombre_cliente</code>	<code>apellido_cliente</code>	<code>edad</code>	<code>genero</code>
▶	3	Omar	Bracho	32	Masculino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

HAVING

ORDER
BY

Nota: El uso de estas cláusulas nos servirá para verificar la omisión de resultados cuando los filtros con aplicados sobre campos de tipo texto.

Select - Where (ejemplo con AND)

Aplicamos “AND” para agregar varios filtros en distintos campos y nos devuelve los registros que cumplen ambos. Es requisito que se cumplan ambas condiciones

WHER
E

La aplicamos en género “Femenino” y los nombres que terminan en “ia”:

```
SELECT * FROM `clientes`  
WHERE  
genero = 'Femenino'  
AND  
nombre_cliente LIKE '%ia';
```

GROU
P BY

HAVIN
G

ORDE
R BY

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
5	Sofia	Gramajo	60	Femenino	
*	NULL	NULL	NULL	NULL	NULL

Select - Where (ejemplo con OR)

Aplicamos “OR” para agregar varios filtros en distintos campos y devuelve los registros que las cumplen. Al contrario que “AND”, estos filtros son independientes entre si

WHER
E

La aplicamos en id_clientes < 3 y id_clientes > 3:

```
SELECT * FROM `clientes` WHERE  
id_clientes < 3  
OR  
id_clientes > 3;
```

GROU
P BY

HAVIN
G

ORDE
R BY

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	4	Mauro	Resquin	48	Masculino
	5	Sofia	Gramajo	60	Femenino
✳	HULL	HULL	HULL	HULL	HULL

Select - GROUP BY

Vamos a generar una agrupación de los géneros que tiene la tabla. Como primer paso, vamos a hacer el conteo del campo “genero”:

WHERE
E

```
SELECT COUNT(genero) FROM `clientes`
```

	count(genero)
▶	5

GROUP BY

Ahora, agregamos la clausula para que nos muestre como se agrupan esos 5 registros por el campo género

HAVING

```
SELECT COUNT(genero) FROM `clientes` GROUP BY genero
```

ORDER
BY

	count(genero)
▶	3
	2

Select - GROUP BY

Ahora, ya sabemos que el conteo de registros de la columna “genero” se agrupa en 2 resultados posibles y que un grupo tiene 3 registros y otro tiene 2

WHER
E

Para completar la consulta, podemos agregar el campo “genero” para saber cuál es cada uno.

GROU
P BY

```
SELECT genero, COUNT(genero) FROM `clientes` GROUP BY genero;
```

HAVIN
G

ORDE
R BY

	genero	count(genero)
▶	Femenino	3
	Masculino	2

Select - HAVING

Como se trata de un filtro aplicable al resultado de una consulta con la cláusula WHERE, vamos a crear ambas para notar la diferencia.

WHERE

Vamos a hacer una consulta de toda la tabla con el filtro “Femenino”

GROU
P BY

```
SELECT * FROM `clientes` WHERE genero =  
'Femenino'
```

HAVIN
G

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	5	Sofia	Gramajo	60	Femenino
●	NULL	NULL	NULL	NULL	NULL

ORDE
R BY

Select - HAVING

Ahora vamos a crear un filtro sobre el resultado, donde establecemos que nos devuelva los que tienen una edad mayor a 30:

WHERE

```
SELECT * FROM `clientes` WHERE genero = 'Femenino' HAVING edad > 30;
```

GROUP BY

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	5	Sofia	Gramajo	60	Femenino
•	NULL	NULL	NULL	NULL	NULL

HAVING

ORDER BY

Nota: HAVING suele utilizarse cuando ya no es suficiente aplicar filtros con WHERE y en bases de datos más complejas al contrario de nuestra tabla con pocos registros.

Select - ORDER BY

Se utiliza cuando necesitamos que el resultado se ordene por un criterio ascendente o descendente. Veamos ambos casos ordenados por el campo edad:

WHERE

SELECT * FROM `clientes` ORDER BY edad DESC

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	1	Cintia	Cruz	16	Femenino
	2	Vanesa	Rojas	25	Femenino
	3	Omar	Bracho	32	Masculino
	4	Mauro	Resquin	48	Masculino
*	5	Sofia	Gramajo	60	Femenino
	NULL	NULL	NULL	NULL	NULL

GROUP BY

SELECT * FROM `clientes` ORDER BY edad ASC

	id_clientes	nombre_cliente	apellido_cliente	edad	genero
▶	5	Sofia	Gramajo	60	Femenino
	4	Mauro	Resquin	48	Masculino
	3	Omar	Bracho	32	Masculino
	2	Vanesa	Rojas	25	Femenino
*	1	Cintia	Cruz	16	Femenino
	NULL	NULL	NULL	NULL	NULL

HAVING

ORDER BY

Nota: También puede aplicarse en campos de texto

Veamos DROP e INSERT en el programa...

Distinct y limit

La cláusula DISTINCT se utiliza en la sentencia SELECT para eliminar duplicados de los resultados de una consulta. Esto puede ser útil en situaciones en las que se desea obtener una lista única de valores de una columna determinada.

La cláusula LIMIT se utiliza para limitar el número de filas que se devuelven en los resultados de una consulta. Esto puede ser útil cuando se trabaja con tablas grandes y se desea limitar la cantidad de datos que se manejan. N es el número máximo de filas que se devolverán en los resultados de la consulta

Sentencia distinct:

```
SELECT DISTINCT columnas  
FROM tabla  
WHERE condición;
```

Sentencia limit:

```
SELECT columnas  
FROM tabla  
WHERE condición  
LIMIT n;
```

Operadores lógicos

La expresión de la cláusula WHERE puede incluir operadores de comparación, como "=", "<", ">", "<=", ">=", "<>", así como operadores lógicos, como "AND", "OR" y "NOT". Además, se pueden utilizar funciones y operadores aritméticos para construir expresiones más complejas.

Operador	Descripción
=	Igualdad
>,<	Mayor, Menor
>=,<=	Mayor o igual, Menor o igual
<>	No Igual
[NOT] BETWEEN	Entre dos valores (inclusive)
IS [NOT] NULL	Comparación con valores nulos
[NOT] LIKE	Comparación de strings con comodines
[NOT] IN	Pertenencia a una lista de valores
NOT	Negación

Ejemplo:

```
SELECT *  
FROM clientes WHERE  
edad >= 18;
```

Otros ejemplos

Ejemplo 1:

```
SELECT *
  FROM productos
 WHERE precio > 50 AND precio < 100;
```

Ejemplo 2:

```
SELECT *
  FROM ordenes
 WHERE categoria = 'Electrónica' AND cantidad > 10;
```

Ejemplo 3:

```
SELECT *
  FROM clientes
 WHERE nombre LIKE 'A%';
```

Order by

La cláusula ORDER BY se utiliza en la sentencia SELECT para ordenar los resultados de una consulta según una o más columnas específicas. ASC o DESC: son palabras clave que se utilizan para indicar el orden ascendente o descendente, respectivamente. Si no se especifica nada, se utiliza el orden ascendente por defecto.

sentencia:

```
SELECT columnas  
FROM tabla  
WHERE condición  
ORDER BY columna1 [ASC|DESC], columna2  
[ASC|DESC], ...;
```

Ejemplo:

```
SELECT *  
FROM productos  
ORDER BY precio DESC;
```

```
SELECT *  
FROM ordenes  
ORDER BY fecha DESC, cantidad  
DESC;
```

group by

La cláusula GROUP BY se utiliza en la sentencia SELECT para agrupar los resultados de una consulta según los valores de una o más columnas. Esta cláusula es muy útil para realizar cálculos y resúmenes de datos.

sentencia:

```
SELECT columna1, columna2, ...,
funcion_agregado(columna
FROM tabla
WHERE condición
GROUP BY columna1, columna2, ...;
```

- Columna1, Columna2, ...: son las columnas por las cuales se desea agrupar los resultados.
- Funcion_agregado: es una función de agregado (como SUM, AVG, MAX, MIN, COUNT, etc.) que se utiliza para realizar cálculos en los datos agrupados.
- Tabla: es el nombre de la tabla de la cual se quieren seleccionar datos.
- Condición: es una expresión que se utiliza para filtrar los datos de acuerdo con ciertos criterios.

Having

La cláusula GROUP BY puede ir acompañada por la cláusula HAVING, que se utiliza para filtrar los resultados de una consulta después de haber sido agrupados.

sentencia:

```
SELECT columna1, columna2, ...,
funcion_agregado(columna
FROM tabla
WHERE condición
GROUP BY columna1, columna2, ...
HAVING condición_agregado;
```

- Condición_agregado: es una expresión que se utiliza para filtrar los resultados de la función de agregado

ejemplos

Ejemplo 1:

```
SELECT producto, SUM(cantidad) AS total_ventas
FROM ventas
GROUP BY producto;
```

Ejemplo 2:

```
SELECT ciudad, AVG(edad) AS promedio_edad
FROM clientes
GROUP BY ciudad
HAVING AVG(edad) > 30;
```

Integremos las cláusulas

Supongamos que tenemos una tabla "pedidos" que contiene información sobre los pedidos realizados por los clientes de una tienda en línea, incluyendo el ID del pedido, el ID del cliente, la fecha del pedido y el monto total del pedido.

Queremos encontrar los clientes que hayan realizado al menos 3 pedidos por un monto total de al menos \$500 en los últimos 6 meses. Para ello, podemos utilizar la siguiente consulta:

```
SELECT cliente_id, COUNT(*) AS cantidad_pedidos,  
SUM(monto_total) AS total_compras  
FROM pedidos  
WHERE fecha_pedido >= DATEADD(month, -6,  
GETDATE())  
GROUP BY cliente_id  
HAVING COUNT(*) >= 3 AND SUM(monto_total) >= 500;
```

Entendamos la consulta

```
SELECT cliente_id, COUNT(*) AS cantidad_pedidos,  
SUM(monto_total) AS total_compras  
FROM pedidos  
WHERE fecha_pedido >= DATEADD(month, -6,  
GETDATE())  
GROUP BY cliente_id  
HAVING COUNT(*) >= 3 AND SUM(monto_total) >= 500;
```

En esta consulta, la cláusula WHERE filtra los registros para seleccionar solo aquellos pedidos realizados en los últimos 6 meses. La cláusula GROUP BY agrupa los resultados por cliente_id y calcula la cantidad total de pedidos (usando la función COUNT) y el monto total de compras (usando la función SUM). Finalmente, la cláusula HAVING filtra los resultados para seleccionar solo aquellos clientes que hayan realizado al menos 3 pedidos y cuyo monto total de compras sea de al menos \$500.

join

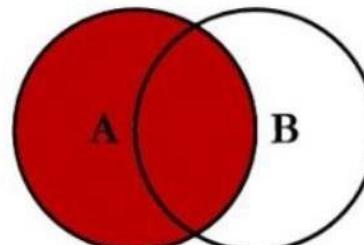
Supongamos dos tablas. En la tabla pedidos que contiene información histórica tenemos la clave foránea producto_id que permite relacionarla con la tabla productos.

Si necesitamos saber alguna característica específica de los producto como por ejemplo cual es su categoría debemos realizar un Join para generar la consulta.

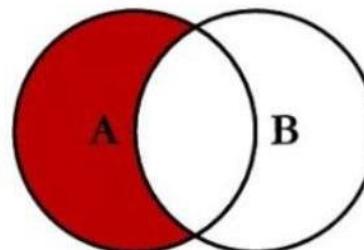
El JOIN es una operación fundamental en SQL que nos permite combinar datos de dos o más tablas en función de una o varias columnas comunes.

Tipos de uniones

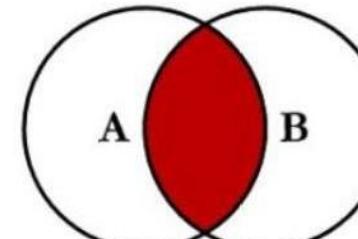
SQL JOINS



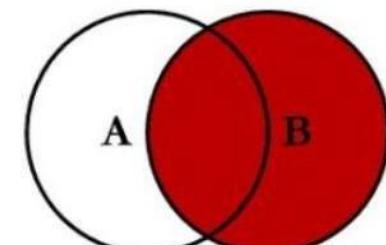
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



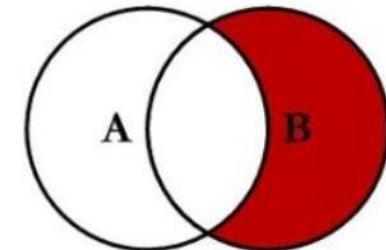
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL.
```



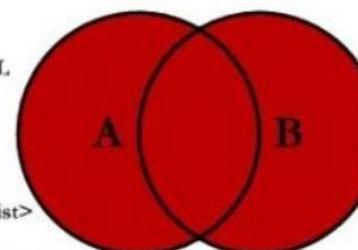
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



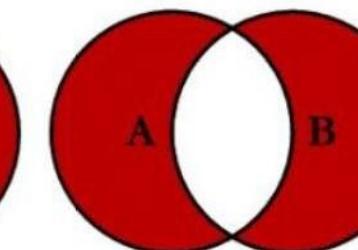
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL.
```



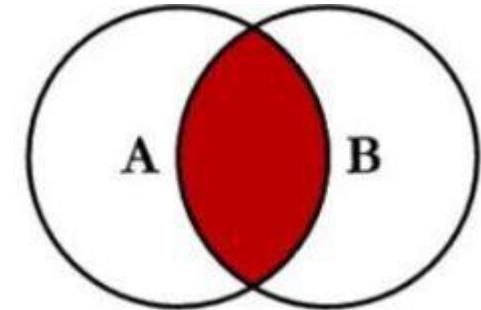
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



© C.L. Moffatt, 2008

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Inner join



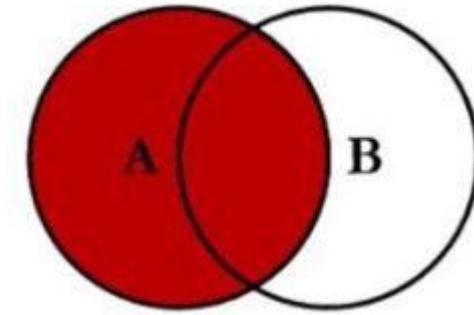
Este es el tipo de JOIN más común. Devuelve solo los registros que tienen coincidencias en ambas tablas en función de la columna o columnas especificadas en la cláusula ON.

Se suelen usar alias para nombrar para simplificar las consultas. Aunque se tiene que respetar correctamente sobre todo cuando hay una columna que posee el mismo nombre en ambas tablas.

sentencia:

```
SELECT tabla1.columna1,  
tabla2.columna2, ...  
FROM tabla1 a  
INNER JOIN tabla2 b  
ON a.columna = b.columna;
```

left join

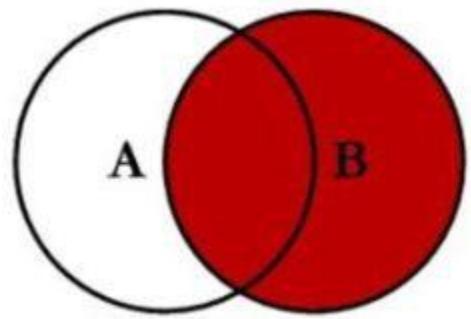


Este tipo de JOIN devuelve todos los registros de la tabla izquierda y los registros correspondientes de la tabla derecha. Si no hay coincidencias en la tabla derecha, se devolverá NULL.

sentencia:

```
SELECT tabla1.columna1, tabla2.columna2, ...
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

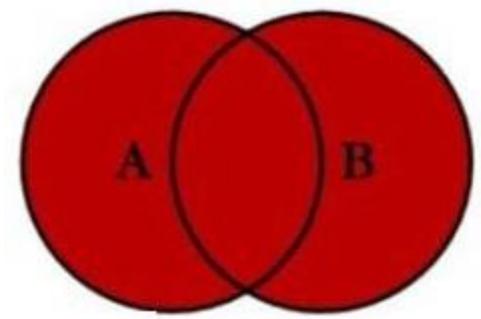
right join



Este tipo de JOIN devuelve todos los registros de la tabla derecha y los registros correspondientes de la tabla izquierda. Si no hay coincidencias en la tabla izquierda, se devolverá NULL.

sentencia:

```
SELECT tabla1.columna1, tabla2.columna2, ...
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna = tabla2.columna;
```



Full outer join

Este tipo de JOIN devuelve todos los registros de ambas tablas y combina los registros que tienen coincidencias en función de la columna o columnas especificadas en la cláusula ON.
Si no hay coincidencias, se devolverá NULL.

sentencia:

```
SELECT tabla1.columna1, tabla2.columna2, ...
FROM tabla1
FULL OUTER JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

ejemplo

id_cliente	nombre	email
1	Juan	juan@example.com
2	Maria	maria@example.com
3	Pedro	pedro@example.com
4	Ana	ana@example.com

id_pedido	id_cliente	fecha	total
1	1	2022-01-01	100
2	2	2022-01-02	50
3	1	2022-01-03	75
4	5	2022-01-04	200
5	NULL	2022-01-05	50

Ejemplo inner join

```
SELECT pedidos.id_pedido, clientes.nombre,  
pedidos.fecha, pedidos.total  
FROM pedidos  
INNER JOIN clientes  
ON pedidos.id_cliente = clientes.id_cliente;
```

id_pedido	nombre	fecha	total
1	Juan	2022-01-01	100
2	María	2022-01-02	50
3	Juan	2022-01-03	75

Ejemplo left join

```
SELECT pedidos.id_pedido, clientes.nombre,  
pedidos.fecha, pedidos.total  
FROM pedidos  
RIGHT JOIN clientes  
ON pedidos.id_cliente = clientes.id_cliente;
```

id_pedido	nombre	fecha	total
1	Juan	2022-01-01	100
2	Maria	2022-01-02	50
3	Juan	2022-01-03	75
4	NULL	2022-01-04	200

Ejemplo right join

```
SELECT clientes.id_cliente, clientes.nombre,  
pedidos.fecha, pedidos.total  
FROM clientes  
RIGHT JOIN pedidos  
ON clientes.id_cliente = pedidos.id_cliente;
```

<code>id_cliente</code>	<code>nombre</code>	<code>fecha</code>	<code>total</code>
1	Juan	2022-01-01	100
1	Juan	2022-01-03	75
2	Maria	2022-01-02	50
5	NULL	2022-01-04	200
NULL	NULL	2022-01-05	50

Ejemplo full outer join

```
SELECT clientes.id_cliente, clientes.nombre,  
pedidos.fecha, pedidos.total  
FROM clientes  
FULL OUTER JOIN pedidos  
ON clientes.id_cliente = pedidos.id_cliente;
```

	id_cliente	nombre	fecha	total
	1	Juan	2022-01-01	100
	1	Juan	2022-01-03	75
	2	Maria	2022-01-02	50
	3	Pedro	NULL	NULL
	4	Ana	NULL	NULL
	5	Luisa	2022-01-04	200
	NULL	NULL	2022-01-05	50

Ejemplo integrador

Imaginemos que tienes dos tablas: "productos" y "ventas". La tabla "productos" tiene información sobre los productos que vende tu tienda, como su nombre, descripción, precio y categoría. La tabla "ventas" tiene información sobre todas las ventas que se han realizado, incluyendo qué producto se vendió, cuántos se vendieron y a qué precio se vendieron. Sin embargo, la tabla "ventas" no tiene información sobre la categoría de cada producto.

Quieres crear un informe que muestre la cantidad total de ventas y los ingresos totales por categoría de producto. Para ello, necesitas combinar la información de las dos tablas utilizando INNER JOIN. El resultado sería algo así:

```
SELECT productos.categoría,  
       SUM(ventas.cantidad) AS cantidad_total,  
       SUM(ventas.cantidad * ventas.precio) AS  
           ingresos_totales  
  FROM productos  
INNER JOIN ventas  
    ON productos.id = ventas.id_producto  
GROUP BY productos.categoría
```

UNION

Esta operación combina el resultado de dos o más consultas SELECT en una sola tabla.

Es importante tener en cuenta que las columnas en ambas consultas deben ser del mismo tipo de datos y tener el mismo número de columnas en el mismo orden. Además, UNION elimina automáticamente los duplicados en el resultado combinado. Si desea incluir duplicados en el resultado combinado, utilice UNION ALL en lugar de UNION.

```
SELECT columna1, columna2, ...
FROM tabla1
UNION [ALL]
SELECT columna1, columna2, ...
FROM tabla2;
```

UNION

Supongamos que tenemos dos tablas: "empleados1" y "empleados2", ambas con la misma estructura de columnas. Queremos obtener la lista completa de empleados de ambas tablas, pero queremos excluir los registros duplicados de la lista final:

```
SELECT nombre, apellido, edad, departamento  
FROM empleados1  
UNION  
SELECT nombre, apellido, edad, departamento  
FROM empleados2;
```



BA MULTIPLICA 2.0

jóvenes X jóvenes



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BA
Joven