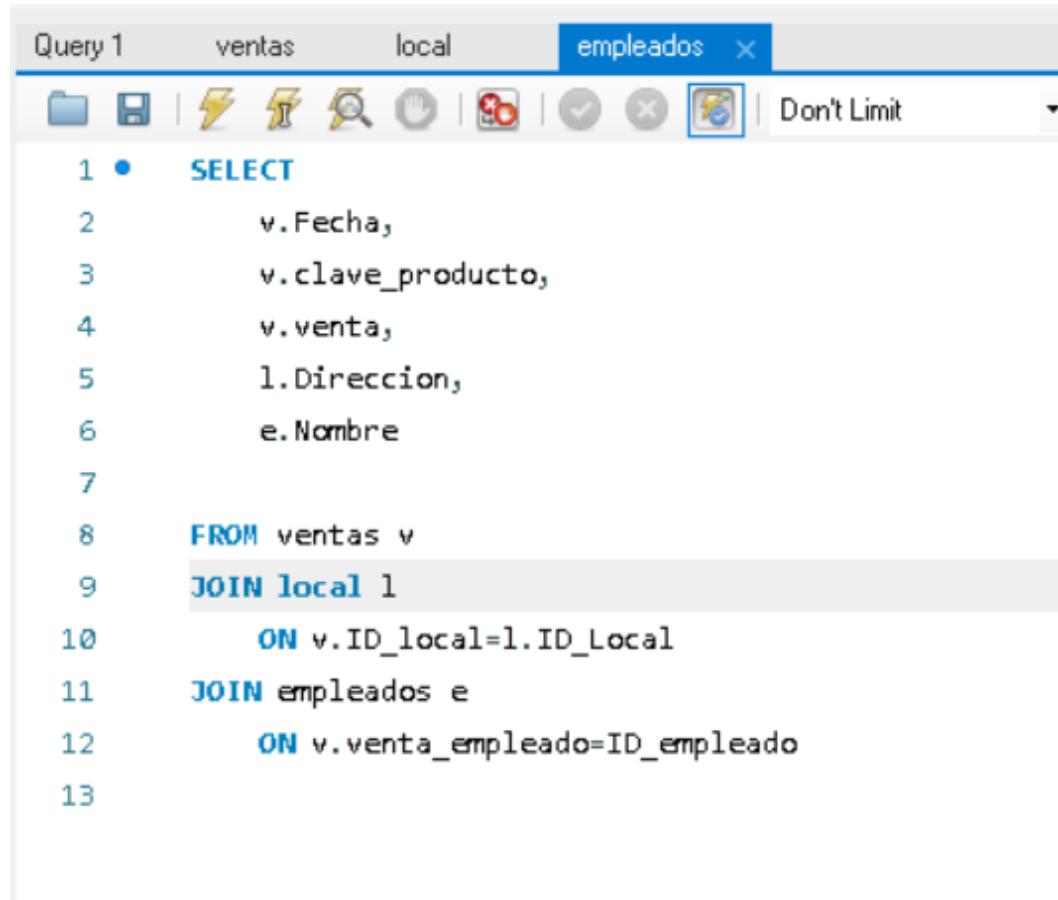


SQL

- JOIN DE UNA MISMA BASE DE DATOS



The screenshot shows a MySQL Workbench interface with a query editor. The title bar of the editor window says "Query 1". Below the title bar, there are tabs for "ventas", "local", and "empleados". The "empleados" tab is currently selected. The toolbar below the tabs contains various icons for database management tasks. The main area of the editor displays a SQL query:

```
1 •  SELECT
2      v.Fecha,
3      v.clave_producto,
4      v.venta,
5      l.Direccion,
6      e.Nombre
7
8  FROM ventas v
9  JOIN local l
10     ON v.ID_local=l.ID_Local
11 JOIN empleados e
12     ON v.venta_empleado=ID_empleado
13
```

- JOIN DE DIFERENTES BASES DE DATOS

The screenshot shows a MySQL Workbench interface. The title bar says 'local' and 'SQL File 4*'. The main area contains the following SQL code:

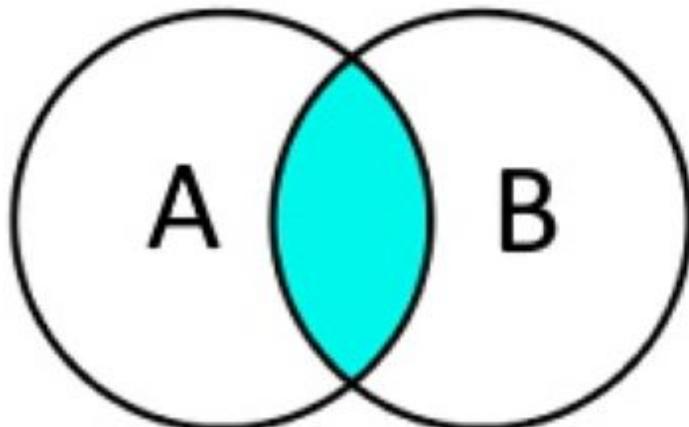
```
1 • SELECT *
2   FROM periodo_1 p
3   JOIN datos.local l
4     ON p.local = l.letra_zona
```

- SELF JOIN

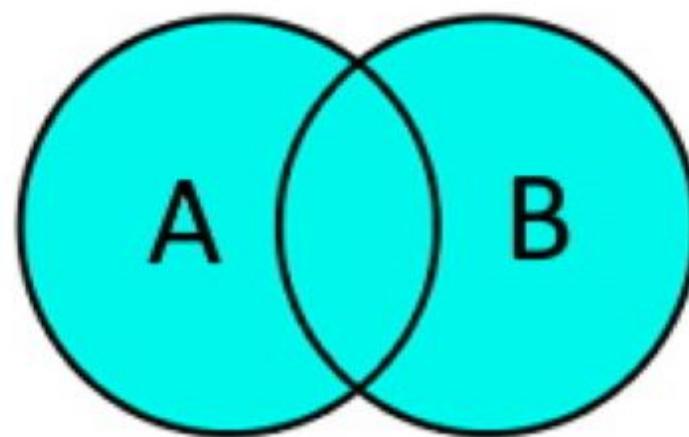
```
1 • SELECT
2     e.ID_Emppleado,
3     e.Nombre,
4     e.ID_Gerente,
5     p.Nombre AS gerente
6 FROM empleados e
7 JOIN empleados p
8     ON e.ID_Gerente = p.ID_Emppleado
```

- JOINS EXTERNOS

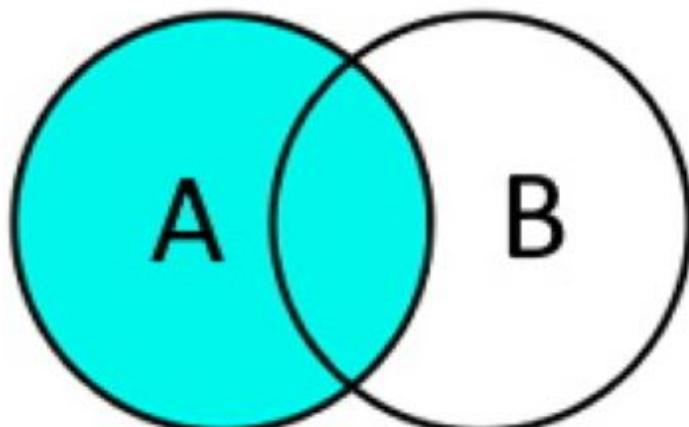
INNER JOIN



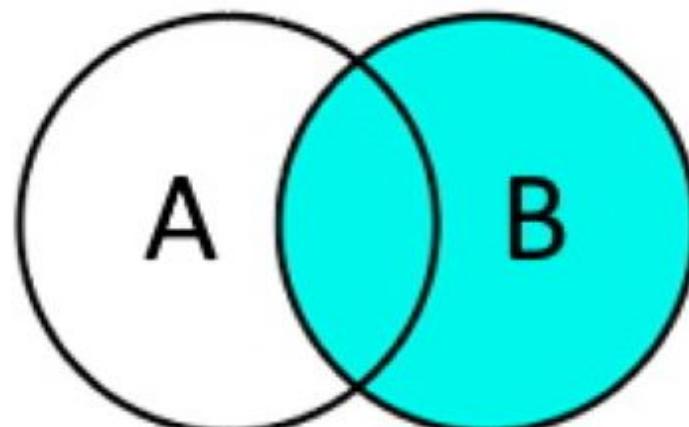
FULL JOIN



LEFT JOIN



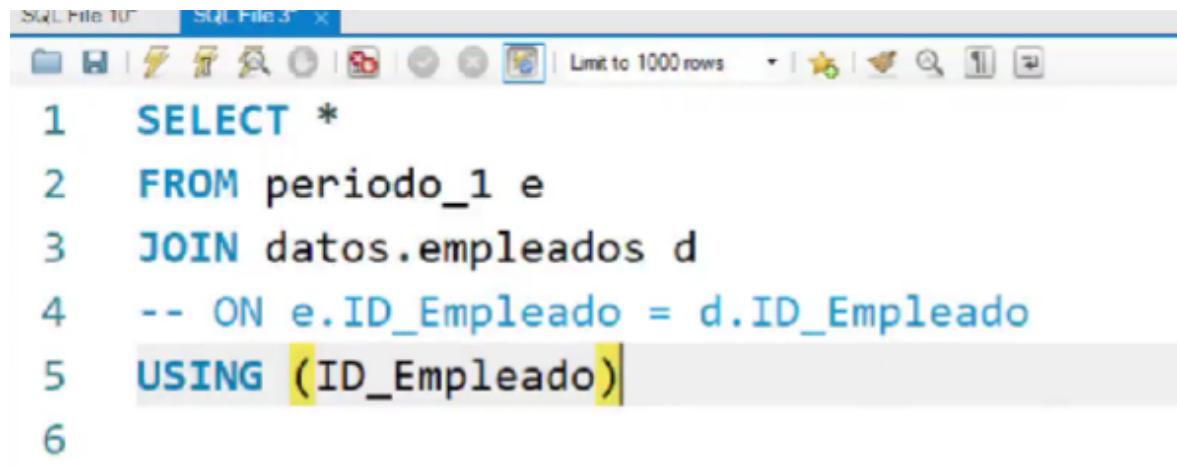
RIGHT JOIN



```
1  SELECT
2      clave_producto,
3      venta,
4      venta_empleado,
5      e.Nombre,
6      e.Apellido
7  FROM empleados e
8  LEFT JOIN ventas v
9      ON v.venta_empleado = e.ID_empleado
```

- USING

Se utiliza para unir 2 columnas de diferentes tablas cuyos nombres son iguales



```
1  SELECT *
2  FROM periodo_1 e
3  JOIN datos.empleados d
4  -- ON e.ID_Empreado = d.ID_Empreado
5  USING (ID_Empreado)
6
```

- CROSS JOIN

cruzar o coombinar los registros de una tabla con otra tabla.

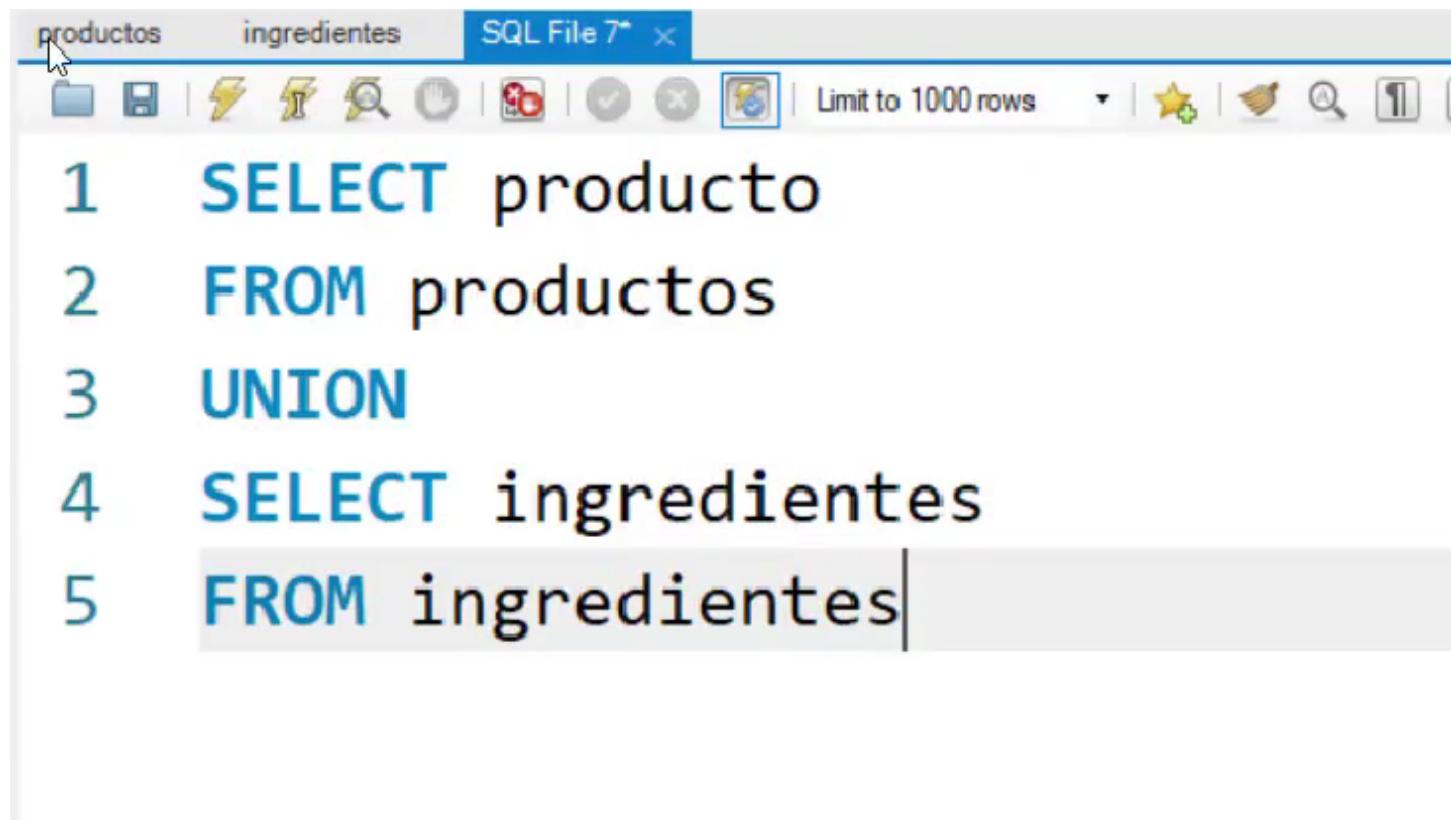
se usa para generar una combinación emparejada de cada fila de la primera tabla con cada fila de la segunda tabla. Este tipo de unión también se conoce como unión cartesiana.

```
1 • SELECT producto, ingredientes
2 FROM productos p
3 CROSS JOIN ingredientes i
4
5
```

- UNION

este tipo de consultas deben tener por regla el mismo numero de columnas(1-1)(2-2)

la primera columna siempre sera la que lleve el nombre de la nueva tabla (en este ejemplo seria producto)



```
1 SELECT producto
2 FROM productos
3 UNION
4 SELECT ingredientes
5 FROM ingredientes|
```

--

EJERCICIOS DE PRACTICA

1-ventas por cliente

(en este caso al traer una nueva tabla tenemos que unirla donde tengan la misma columna en este caso en ID_cliente)

The screenshot shows a MySQL Workbench interface with a query editor titled "SQL File 5". The query is:

```
1 SELECT v.venta, c.nombre
2 FROM ventas v
3 JOIN clientes c ON v.ID_cliente = c.ID_cliente
```

2-ventas por zona

En este caso se unen por la columna zona

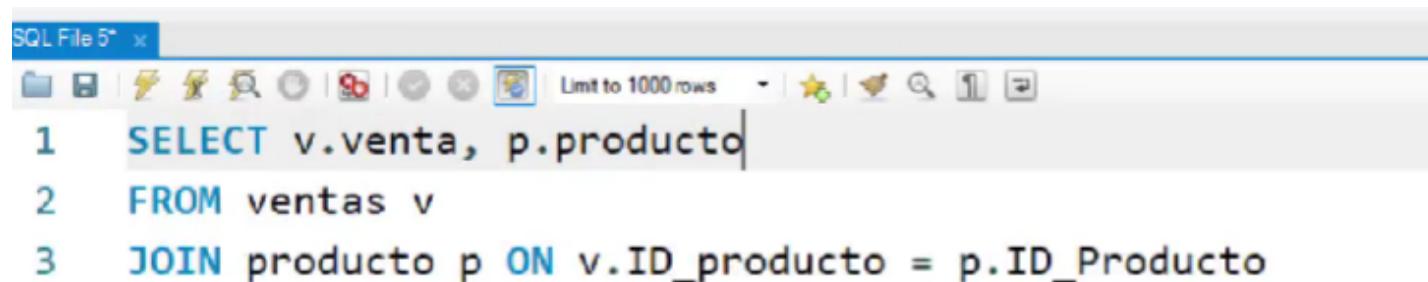
The screenshot shows a MySQL Workbench interface with a query editor titled "SQL File 5". The query is:

```
1 SELECT v.venta, z.Zona
2 FROM ventas v
3 JOIN zona z ON v.ID_zona = z.ID_zona
```

3-Ventas por vendedor (usamos la tabla de ventas y de vendor y se unir en ID_vendedor)

```
SELECT v.venta, ve.nombre  
FROM ventas v  
JOIN vendedor ve ON v.ID_vendedor = ve.ID_vendedor
```

4-Ventas por producto



```
SQL File 5* ×  
Limit to 1000 rows  
1 SELECT v.venta, p.producto  
2 FROM ventas v  
3 JOIN producto p ON v.ID_producto = p.ID_Producto
```

5-Multiples tablas JOIN

Informacion de venta por cliente y vendedor(3 tablas)

SQL File 5

```
1 SELECT v.venta, c.nombre, ve.nombre AS vendedor
2 FROM ventas v
3 JOIN clientes c ON v.ID_cliente = c.ID_cliente
4 JOIN vendedor ve ON v.ID_vendedor = ve.ID_vendedor
```

I

ventas por cliente, zona y producto

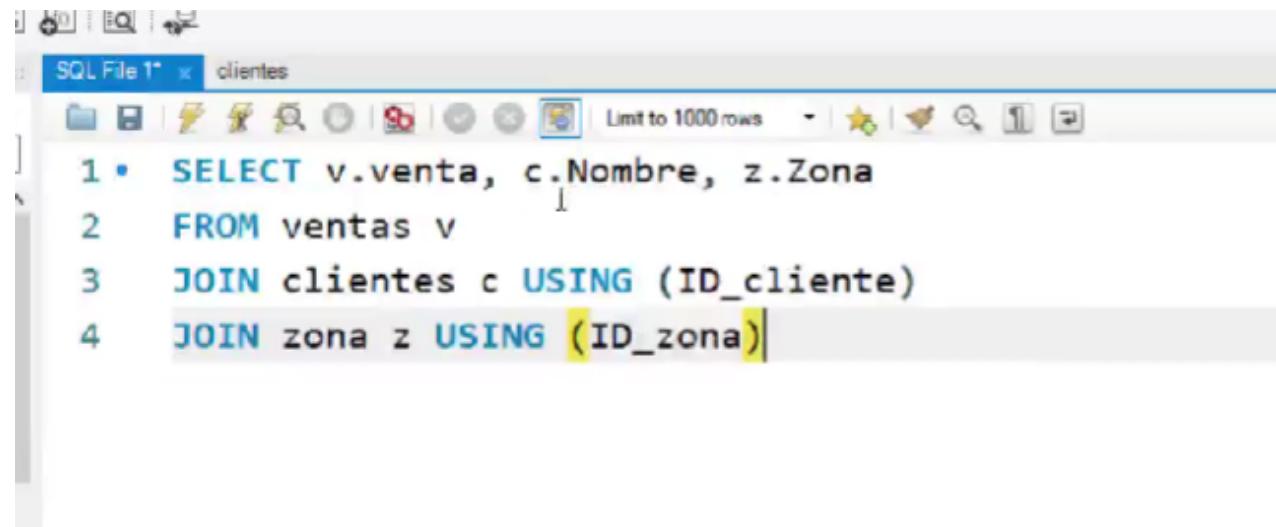
```
SELECT v.venta, c.nombre, z.zona, p.producto
FROM ventas v
JOIN clientes c ON v.ID_Cliente = c.ID_cliente
JOIN zona z ON v.ID_zona = z.ID_zona
JOIN producto p ON v.ID_producto = p.ID_producto
```

Encontrar al vendedor que no ha vendido nada

The screenshot shows a SQL editor interface with a menu bar at the top. The menu items include Database, Server, Tools, Scripting, and Help. Below the menu is a toolbar with various icons. A tab labeled "SQL File 1" is active. The main area contains a SQL query:

```
1 •  SELECT venta, ve.nombre
2   FROM ventas v
3   RIGHT JOIN vendedor ve
4  ON v.ID_vendedor = ve.ID_vendedor
5  ORDER BY venta
```

venta con el cliente y zona usando using



```
SQL File 1* clientes
1 •  SELECT v.venta, c.Nombre, z.Zona
2   FROM ventas v
3   JOIN clientes c USING (ID_cliente)
4   JOIN zona z USING (ID_zona)
```

venta por vendedor y por producto

SQL File 1* x vendedor

The screenshot shows a MySQL Workbench interface. The title bar says "SQL File 1* x vendedor". Below the title bar is a toolbar with various icons. A dropdown menu labeled "Limit to 1000 rows" is open. The main area contains the following SQL code:

```
1 • SELECT v.venta, ve.Nombre, p.producto
2   FROM ventas v
3   JOIN vendedor ve USING (ID_vendedor)
4   JOIN producto p USING (ID_producto)
```

Unir 2 consultas A Y B (UNION)

```
1 -- clientes a y b
2
3 • SELECT nombre, Clasificacion_credito
4 FROM clientes
5 WHERE Clasificacion_credito = "A"
6 UNION
7 SELECT nombre, Clasificacion_credito
8 FROM clientes
9 WHERE Clasificacion_credito = "B"
```

recuerden la misma cantidad de columnas en el select

```
1 -- VENDEDORES CASADOS Y > 50 , SOLTEROS Y < 30
2
3 • SELECT nombre, edad, estado_civil
4 FROM vendedor
5 WHERE estado_civil = "casado" AND edad > 50
6 UNION
7 SELECT nombre, edad, estado_civil
8 FROM vendedor
9 WHERE estado_civil = "soltero" AND edad < 30
```

RESUMIR DATOS

Funciones Agregacion

Query 1 ventas ventas

```
1 • SELECT
2     MAX(venta) AS maximo,
3     MIN(venta) AS minimo,
4     AVG(venta) AS promedio,
5     SUM(venta) AS Total,
6     COUNT(DISTINCT venta) AS cantidad
7 FROM ventas
```

Limit to 1000 rows

```
1 • SELECT
2     ID_local,
3     SUM(venta) AS total
4 FROM ventas
5 WHERE venta > 1000
6 GROUP BY ID_local
7 ORDER BY ID_local DESC
```

Query 1 x empleados

Limit to 1000 rows

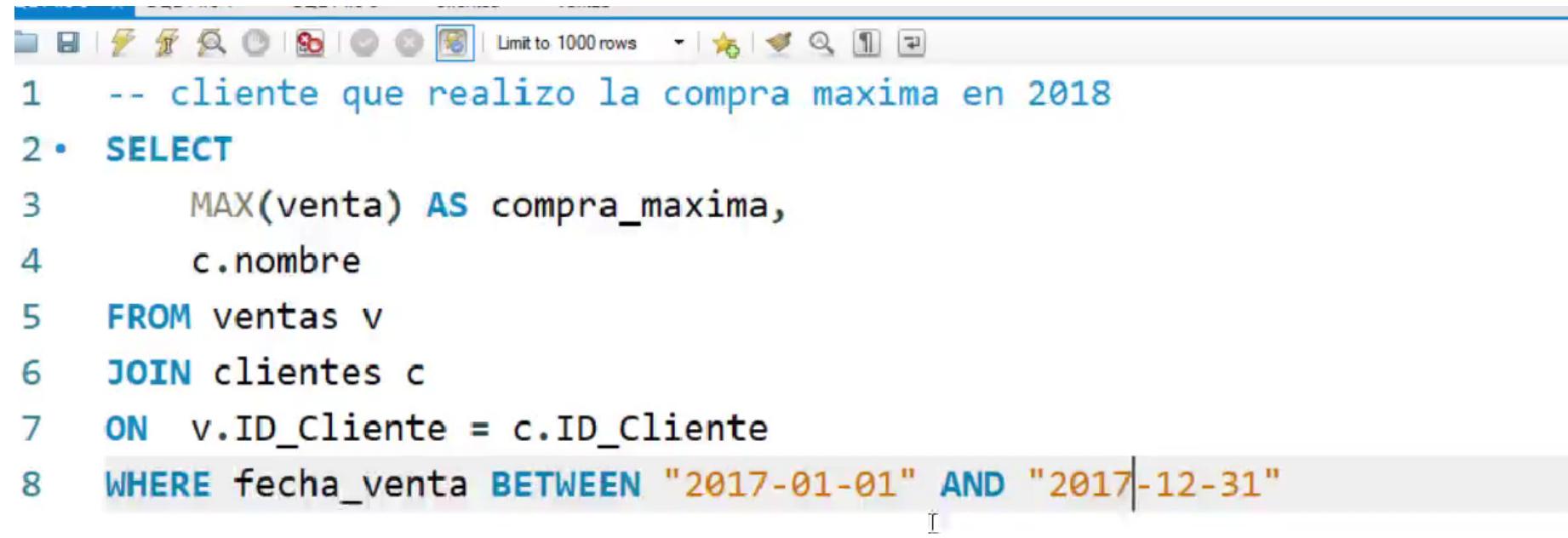
```
1 SELECT ID_local, SUM(venta) AS venta_total
2 FROM ventas
3 WHERE clave_producto = "pzz"
4 GROUP BY ID_local, clave_producto
5 HAVING venta_total > 1500
6 ORDER BY venta_total ASC
7
8 -- HAVING
```

```
1 • SELECT ID_local, clave_producto, SUM(venta) AS venta_total  
2   FROM ventas  
3   GROUP BY ID_local, clave_producto WITH ROLLUP  
4
```

	ID_local	clave_producto	venta_total
▶	1	brr	3202
	1	pzz	4628
	1	qsd	1698
	1	NULL	9528
	2	cz	2275
	2	pzz	1349
	2	NULL	3624
	3	L...	1067

EJERCICIOS 5.5

1-



The screenshot shows a MySQL Workbench interface with a query editor window. The window title is 'SQL' and the tab bar shows 'SQL'. The toolbar includes icons for file operations, search, and refresh. A status bar at the bottom indicates 'Limit to 1000 rows'. The query itself is a SELECT statement:

```
1 -- cliente que realizo la compra maxima en 2018
2 • SELECT
3     MAX(venta) AS compra_maxima,
4     c.nombre
5 FROM ventas v
6 JOIN clientes c
7 ON v.ID_Cliente = c.ID_Cliente
8 WHERE fecha_venta BETWEEN "2017-01-01" AND "2017-12-31"
```

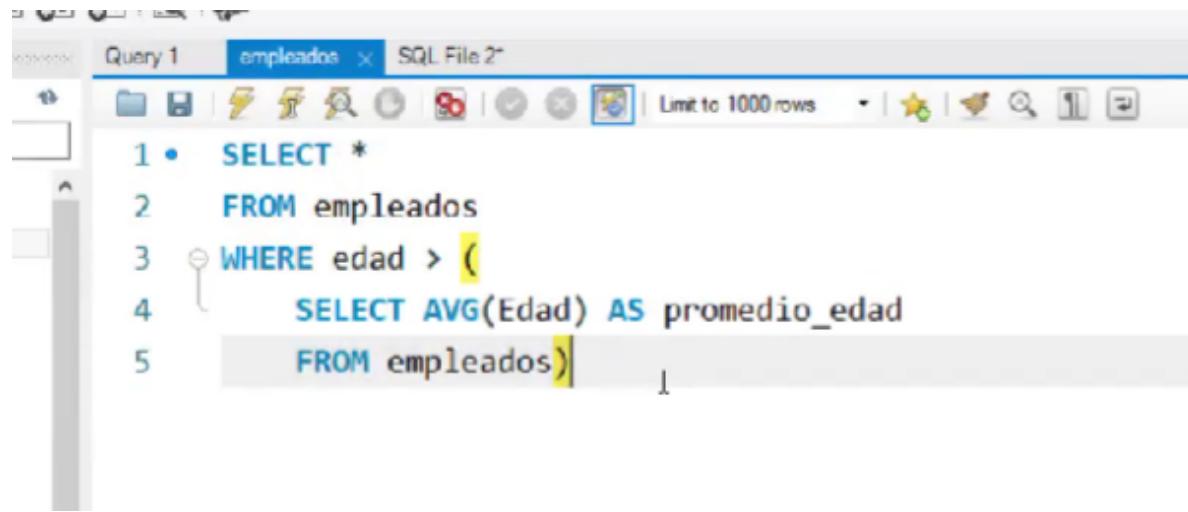
2-

```
-- cual fue el pago menor que recibimos en 2018
SELECT
    MIN(pago) AS pago_minimo,
    ID_pago
FROM pagos
WHERE fecha_pago BETWEEN "2018-01-01" AND "2018-12-31"
```

3-

```
1 -- cuantas ventas hubo en el 2ndo semestre del 2018
2
3 • SELECT
4     COUNT(venta) AS CONTEO
5 FROM ventas
6 WHERE fecha_venta BETWEEN "2018-07-01" AND "2018-12-31"
```

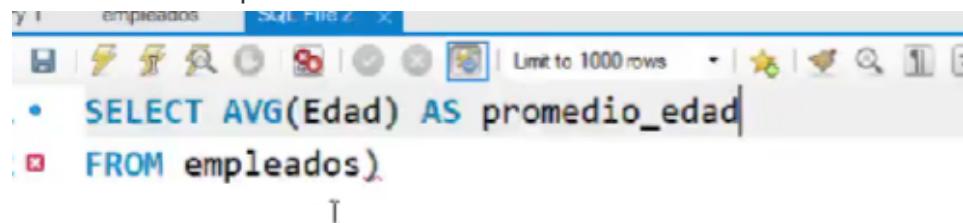
- SUBQUERIS



A screenshot of the MySQL Workbench interface. The title bar shows "Query 1" and "empleados". The main area contains the following SQL code:

```
1 • SELECT *
2   FROM empleados
3   WHERE edad > (
4       SELECT AVG(Edad) AS promedio_edad
5       FROM empleados)
```

visualizar el subqueries



A screenshot of the MySQL Workbench interface, showing the expanded subquery visualization. The title bar shows "empleados" and "SQL File 2". The main area contains the following SQL code:

```
• SELECT AVG(Edad) AS promedio_edad
□ FROM empleados)
```

EJERICICOS CAP 7

```
1 -- vista con las ventas totales para los últimos 3 años
2
3 • SELECT "2017" AS AÑO,
4     SUM(venta) AS total_año
5 FROM ventas
6 WHERE YEAR(fecha_venta) = YEAR(NOW()) - 3
7 UNION
8 SELECT "2018" AS AÑO,
9     SUM(venta) AS total_año
10 FROM ventas
11 WHERE YEAR(fecha_venta) = YEAR(NOW()) - 2
```

```
1 -- Dar seguimiento a ventas de 2019 mayores a 2019
2
3 • SELECT ID_venta,
4     SUM(venta),
5     IF(SUM(venta)>2900000,"Dar seguimiento","no") AS estado
6 FROM ventas
7 WHERE Fecha_venta BETWEEN "2019-01-01" AND "2019-12-31"
8 GROUP BY ID_venta
9 HAVING estado = "Dar seguimiento"
```

CAP 8

- VISTAS

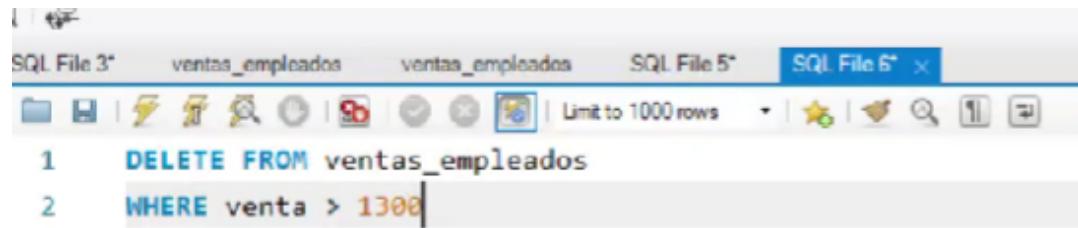
supongamos que tenemos un código y queremos reutilizarlo colocamos el create view as y debajo todas mis líneas de código

```
1 • USE datos;
2
3 • CREATE VIEW ventas_empleados AS
4 SELECT
5     L.LETRA_ZONA,
6     L.TELEFONO,
7     v.venta,
8     v.venta_empleado,
9     e.Nombre
10    FROM local l
11    LEFT JOIN ventas v
12        ON l.ID_Local = v.ID_Local
13    LEFT JOIN empleados e
14        ON v.venta_empleado = e.ID_empleado
15
```

- Reemplazar una vista

```
1 • USE datos;
2
3 • CREATE OR REPLACE VIEW ventas_empleados AS
4 SELECT
5     L.LETRA_ZONA,
6     L.TELEFONO,
7     v.venta,
8     v.venta_empleado,
9     e.Nombre
10    FROM local l
11    LEFT JOIN ventas v
12        ON l.ID_Local = v.ID_Local
13    LEFT JOIN empleados e
14        ON v.venta_empleado = e.ID_empleado
15    WHERE venta > 1000
16
```

ELIMINAR UNA VISTA



The screenshot shows a MySQL Workbench interface with multiple tabs at the top: SQL File 3*, ventas_empleados, ventas_empleados, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active. Below the tabs is a toolbar with various icons. The main area contains two numbered lines of SQL code:

```
1  DELETE FROM ventas_empleados
2  WHERE venta > 1300
```

TRIGGER

es un bloque de código o instrucciones de SQL que se ejecutan automáticamente (antes o después de un insert o delete (basicamente los DML, comando de manipulacion)), nos pueden ayudar para tener consistencia en nuestros datos. Basicamente modifica en el momento de la ejecución de una tabla.

```
DELIMITER $$
```

```
CREATE TRIGGER pago_after_insert
    AFTER INSERT ON pago_orden
    FOR EACH ROW
BEGIN
    UPDATE orden
    SET Balance = Balance + NEW.cantidad
    WHERE Orden = NEW.Orden;
END $$
```

```
DELIMITER ;
```

VER TRIGGER

The screenshot shows the MySQL Workbench interface. At the top, there is a menu bar with 'Server', 'Tools', 'Scripting', and 'Help'. Below the menu is a toolbar with various icons. Two tabs are open: 'SQL File 8*' and 'SQL File 7*'. The 'SQL File 7*' tab is active and contains the following SQL command:

```
1 SHOW TRIGGERS LIKE "pago%"
```

ELIMINAR TRIGGER

SQL File 8* SQL File 7* X

Limit to 1000 rows

```
1  DROP TRIGGER IF EXISTS pago_after_insert
```

EVENTOS

```
1  DELIMITER $$  
2  
3  • CREATE EVENT anual_borrar_filas_auditoria  
4  
5  ON SCHEDULE  
6  
7      EVERY 1 YEAR STARTS "2020-12-25" ENDS "2023-12-25"  
8  
9  DO BEGIN  
10     DELETE FROM auditoria_pagos  
11     WHERE Fecha < NOW() - INTERVAL 1 YEAR;  
12  
13  
14 END $$  
15  
16 DELIMITER ;
```

TRANSACCIONES

Query 1 x orden pago_orden proveedores

The screenshot shows a MySQL Workbench interface with a query editor titled 'Query 1'. The editor contains the following SQL code:

```
1 • USE proveedores;
2
3 • START TRANSACTION;
4
5 • INSERT INTO orden
6     VALUES (DEFAULT, curdate(), 9999, 51344, "Verdura mix", 150, null);
7
8 • INSERT INTO pago_orden
9     VALUES (DEFAULT, curdate(), 4444, 9999, 150);
10
11 • COMMIT
```

The code performs the following steps:

- USE proveedores;
- START TRANSACTION;
- INSERT INTO orden (column1, column2, column3, column4, column5, column6, column7) VALUES (DEFAULT, curdate(), 9999, 51344, "Verdura mix", 150, null);
- INSERT INTO pago_orden (column1, column2, column3, column4, column5) VALUES (DEFAULT, curdate(), 4444, 9999, 150);
- COMMIT;

TOPOS DE DATOS

STRING

- **CHAR (n)** = cuando tienen longitud fija como códigos de 5 letras
- **VARCHAR (n)** = para guardar nombres, usuarios, emails, direcciones, **código postal, celular**

Normalmente se usa VARCHAR (n), por ejemplo con VARCHAR(30) tengo 30 caracteres para guardar casi cualquier nombre de pila que exista en español.

VARCHAR(n) guarda un máximo de 65,535 caracteres (64KB)

MEDIUMTEXT 16 Millones de caracteres (16MB)

LONGTEXT hasta 4GB, para guardar un libro

TINYTEXT hasta 255 bytes

TEXT hasta 64KB

Según el idioma se usa los bytes por carácter:

Inglés = 1 bytes

Europa Middle East = 2 bytes

Asian = 3 bytes



INTEGER

- TINYINT desde -128 hasta 127 con 1b
- UNSIGNED TINYINT desde 0 hasta 255
- SMALLINT desde -32K hasta 32K con 2b
- MEDIUMINT desde -8M hasta 8M con 3b
- INT desde -2B hasta 2B con 4b
- BIGINT desde -9Z hasta 9Z con 8b

ZEROFILL rellena según los dígitos que le dices, como ejemplo:

INT(3) = 001

DECIMALES

DECIMAL (p,e)

p = precisión es el número de dígitos

e = escala es el número de dígitos a la derecha del punto decimal

DECIMAL (5,2) = 123.45

DEC

NUMERIC

FIXED

FLOAT usa 4b

DOUBLE usa 8b

Boolenaos

Son valores de Sí y NO o también True or False

TRUE = 1

FALSE = 0

- **BOOL**
- **BOOLEAN**

FECHA Y HORA

DATE = sólo fechas como 07/07/2019

TIME = para tiempo ➔

DATETIME = combinado de fecha y tiempo

TIMESTAMP = normalmente se usa para mantener un control sobre cuándo se modificó un registro (estampa)

YEAR = año literal

BLOB

TINYBLOB = guarda datos binarios hasta 255b, normalmente es un archivo, imagen, video, etc.

BLOB = guarda datos hasta 65KB



MEDIUMBLOB = guarda datos hasta 16MB

LONGBLOB = 4GB

DISEÑO DE UNA BASE DE DATOS

Data Modeling

- a) Entender el negocio, sus necesidades y características, para qué será usada la base de datos.
- b) Construir un modelo conceptual (Conceptual Model), nos ayuda a identificar las variables.
- c) Modelo lógico (Logical model) – este modelo nos muestra las tablas y columnas que vamos a necesitar para poder usar los datos.
- d) Modelo físico (Physical model) – ya que tenemos el modelo lógico es hora de plasmarlo en un modelo físico en donde tenemos que tener en cuenta las características del sistema y la tecnología en donde se implementará nuestro modelo.

Esta respuesta es correcta.

Truquito

Ten en cuenta que cuando queremos decir que hemos o no hemos estado **EN** algún sitio, en inglés siempre usamos have been/haven't been **TO** a place, nunca **IN**.

A bucket list - la lista de cosas que hacer antes de morir