



Universidad Tecnológica De Panamá  
Facultad de Ingeniería en Sistemas Computacionales  
2do Semestre

Profesor: Ronald Ponce

Integrantes: Jean Meléndez 8-985-955

Kevin Valdés 8-1021-301

Daniel Gonzales 8-1022-1099

Martin Liao 1-757-1706

Ricardo Rose

Asignatura: Base de datos

Taller No.2: El álgebra relacional y su relación con las bases  
de datos

Año Lectivo: 2024

1ra parte.

Se le proporcionará un script SQL que creará las tablas necesarias y añadirá los registros correspondientes para que puedas ejecutar los ejemplos prácticos de cada operación de álgebra relacional que mencionamos anteriormente. Este script te permitirá ejemplificar el uso de las instrucciones matemáticas relacionales y sus homólogas en SQL.

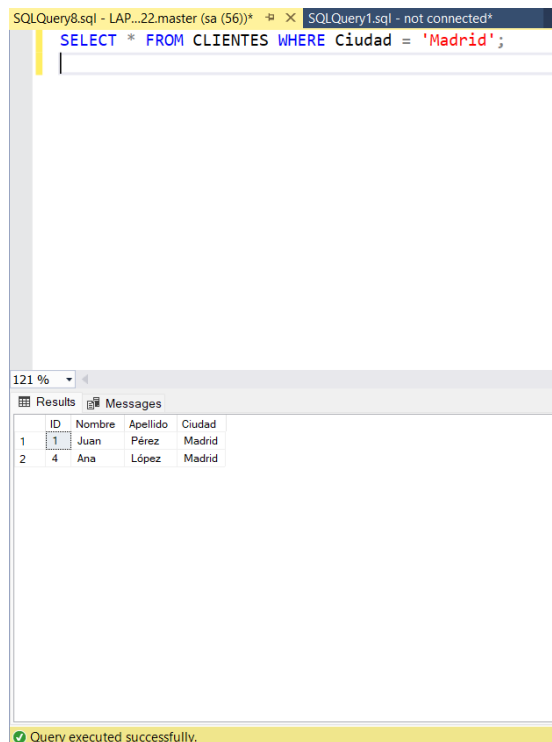
## 1. Selección ( $\sigma$ )

**Definición matemática:** La selección ( $\sigma$ ) filtra filas de una relación que cumplen una condición específica.

**Instrucción SQL equivalente:** SELECT ... WHERE ...

**Ejemplo práctico:**

SELECT \* FROM CLIENTES WHERE Ciudad = 'Madrid';



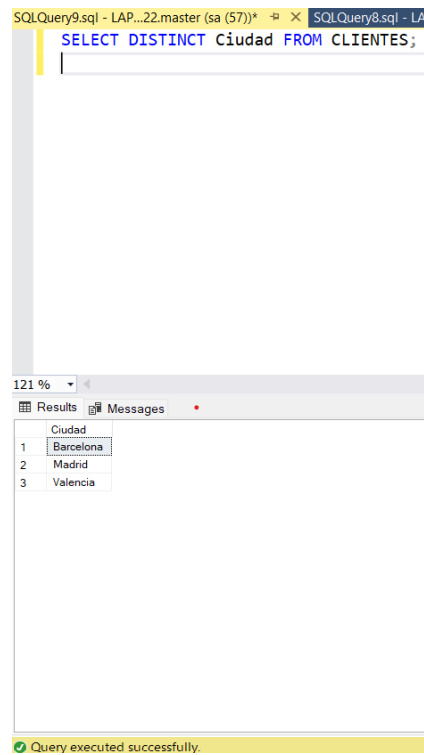
## 2. Proyección ( $\pi$ )

Definición matemática: La proyección ( $\pi$ ) obtiene columnas específicas de una relación, eliminando duplicados.

Instrucción SQL equivalente: SELECT DISTINCT ...

Ejemplo práctico:

SELECT DISTINCT Ciudad FROM CLIENTES;



The screenshot shows a SQL query editor with the query `SELECT DISTINCT Ciudad FROM CLIENTES;` entered. Below the editor, the 'Results' tab is active, displaying a table with the following data:

	Ciudad
1	Barcelona
2	Madrid
3	Valencia

A status bar at the bottom indicates 'Query executed successfully.'

### 3. Unión (U)

Definición matemática: La unión ( $\cup$ ) combina dos relaciones, eliminando duplicados.

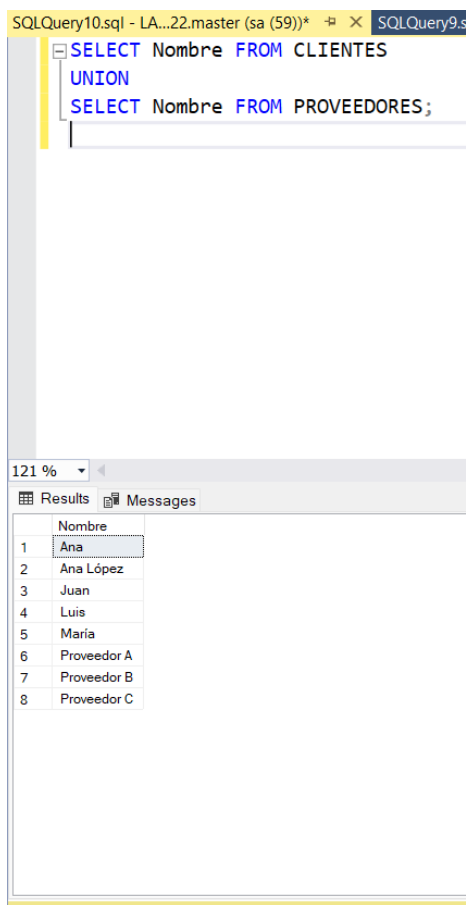
Instrucción SQL equivalente: `SELECT ... UNION SELECT ...`

Ejemplo práctico:

`SELECT Nombre FROM CLIENTES`

`UNION`

`SELECT Nombre FROM PROVEEDORES;`



#### 4. Intersección ( $\cap$ )

Definición matemática: La intersección ( $\cap$ ) devuelve las filas que están en ambas relaciones.

Instrucción SQL equivalente: No hay un operador de intersección directo en SQL, pero se puede simular Usando GROUP BY con HAVING se pueden contar los nombres y filtrar aquellos que aparecen en ambas tablas.

```
SELECT Nombre
```

```
FROM (
```

```
    SELECT Nombre FROM CLIENTES
```

```
    UNION ALL
```

```
    SELECT Nombre FROM PROVEEDORES
```

```
) AS Combined
```

```
GROUP BY Nombre
```

```
HAVING COUNT(*) > 1;
```

## 5. Diferencia (-)

Álgebra Relacional: Devuelve filas de una tabla que no están en otra.  
Ejemplo:

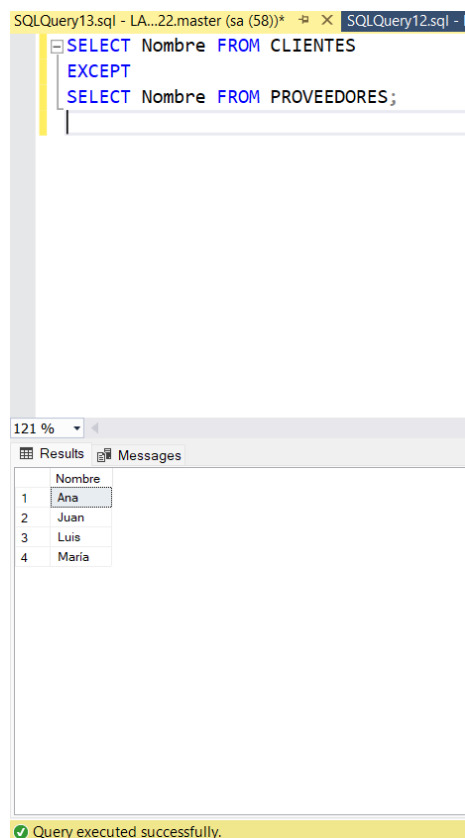
Obtener los nombres de los clientes que no son proveedores.

Ejemplo:

```
SELECT Nombre FROM CLIENTES
```

```
EXCEPT
```

```
SELECT Nombre FROM PROVEEDORES;
```



Como en esta imagen se ve, no muestra los proveedores designados como: proveedor A, proveedor B, Proveedor C, etc.

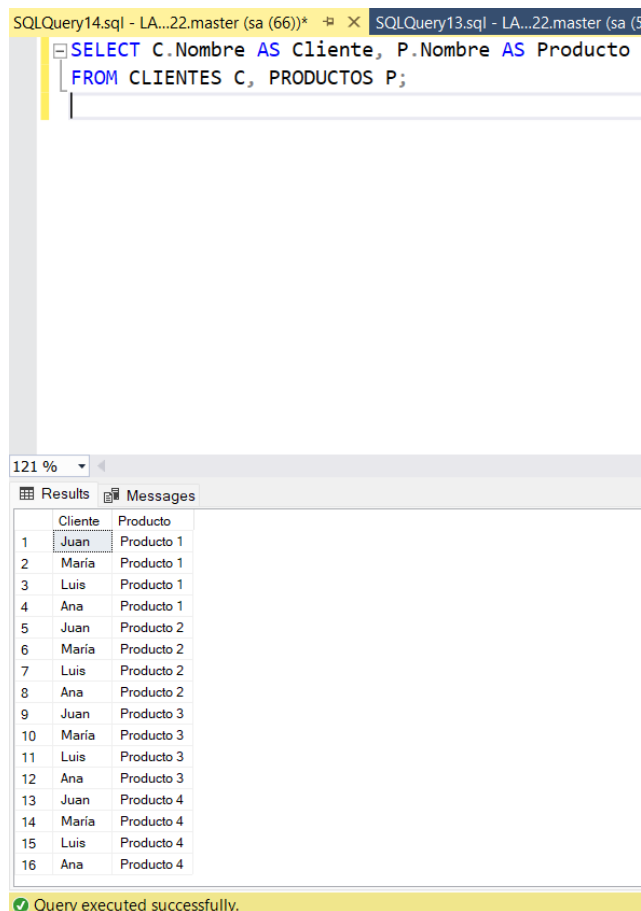
## 6. Producto Cartesiano (x)

Álgebra Relacional: Combina todas las filas de dos tablas. Ejemplo:

Obtener todos los pares de clientes y productos.

Ejemplo:

```
SELECT C.Nombre AS Cliente, P.Nombre AS Producto  
FROM CLIENTES C, PRODUCTOS P;
```



SQLQuery14.sql - LA...22.master (sa (66))\* X SQLQuery13.sql - LA...22.master (sa (5

```
SELECT C.Nombre AS Cliente, P.Nombre AS Producto  
FROM CLIENTES C, PRODUCTOS P;
```

121 %

Results Messages

	Cliente	Producto
1	Juan	Producto 1
2	Maria	Producto 1
3	Luis	Producto 1
4	Ana	Producto 1
5	Juan	Producto 2
6	Maria	Producto 2
7	Luis	Producto 2
8	Ana	Producto 2
9	Juan	Producto 3
10	Maria	Producto 3
11	Luis	Producto 3
12	Ana	Producto 3
13	Juan	Producto 4
14	Maria	Producto 4
15	Luis	Producto 4
16	Ana	Producto 4

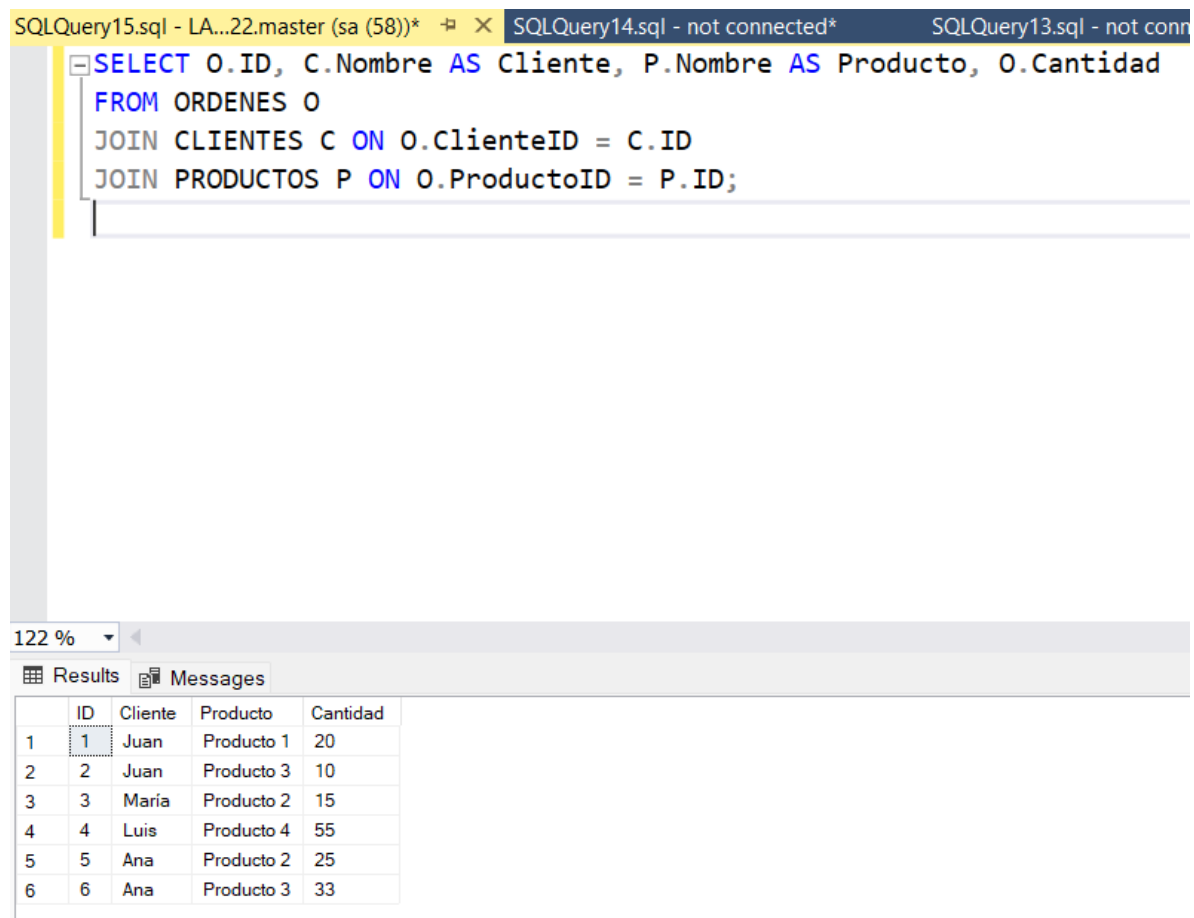
Query executed successfully.

## 7. Join (⋈)

Álgebra Relacional: Combina filas de dos tablas basándose en una condición de relación. Ejemplo:

Obtener información de las órdenes junto con el nombre del cliente y del producto.

```
SELECT O.ID, C.Nombre AS Cliente, P.Nombre AS Producto, O.Cantidad
FROM ORDENES O
JOIN CLIENTES C ON O.ClienteID = C.ID
JOIN PRODUCTOS P ON O.ProductoID = P.ID;
```



The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query editor contains the following SQL code:

```
SELECT O.ID, C.Nombre AS Cliente, P.Nombre AS Producto, O.Cantidad
FROM ORDENES O
JOIN CLIENTES C ON O.ClienteID = C.ID
JOIN PRODUCTOS P ON O.ProductoID = P.ID;
```

The results pane shows the following data:

	ID	Cliente	Producto	Cantidad
1	1	Juan	Producto 1	20
2	2	Juan	Producto 3	10
3	3	María	Producto 2	15
4	4	Luis	Producto 4	55
5	5	Ana	Producto 2	25
6	6	Ana	Producto 3	33



## 2da Parte:

1. ¿En base al siguiente script suministrado interprete que realiza la siguiente consulta y que relación guarda con las instrucciones vista en las evaluadas a la matemática relacional?

### Interpretación de la Consulta

La consulta SQL selecciona los **ClienteID** y **Nombre** de la tabla **CLIENTES\_NEW** que han hecho órdenes en **ORDENES\_NEW**. La relación se establece a través de un **INNER JOIN**, donde se conectan las órdenes con los clientes mediante **ClienteID**.

Objetivo de la consulta: Identificar a aquellos clientes que han comprado todos los productos disponibles en la tabla **ORDENES\_NEW**. Esto se logra agrupando los resultados por cliente y aplicando una condición que filtra a los clientes que han adquirido un número de productos únicos igual al total de productos únicos en la tabla de órdenes.

### Relación con Álgebra Relacional:

**Selección y Proyección:** Se está seleccionando información específica (**ClienteID** y **Nombre**) de las tablas.

**Agrupación:** El uso de **GROUP BY** representa una forma de agrupación similar a las operaciones de agrupación en álgebra relacional.

**Condición de Filtrado:** La cláusula **HAVING** se usa para filtrar resultados después de la agrupación, lo que es análogo a las condiciones en álgebra relacional.

2. Explique que hace cada una de las funciones de domino agregado a fin poder explicar con más claridad lo solicitado En el punto 1.

Explicación de las Funciones de Dominio Agregado:

a. `COUNT(DISTINCT Producto)` en el `HAVING`

Esta función cuenta cuántos productos únicos ha comprado cada cliente. En el contexto del `HAVING`, se utiliza para filtrar aquellos clientes que han comprado una cantidad de productos únicos igual al total de productos disponibles.

Así, solo se seleccionan los clientes que han comprado cada tipo de producto al menos una vez.

b. La subconsulta `(SELECT COUNT(DISTINCT Producto) FROM ORDENES_NEW)`

Esta subconsulta determina cuántos productos distintos están disponibles en la tabla `ORDENES_NEW`. Su resultado se compara con el conteo de productos únicos que cada cliente ha comprado en la consulta principal.

Esto asegura que solo se incluyan aquellos clientes que han adquirido la misma cantidad de productos únicos que el total disponible.

3. Explique que hacen las siguientes dos instrucciones:

a. COUNT(DISTINCT Producto) en el HAVING:

b. La subconsulta (SELECT COUNT(DISTINCT Producto) FROM ORDENES\_NEW):

Explicación de las Instrucciones:

a. COUNT(DISTINCT Producto) en el HAVING

Función: Cuenta el número de productos únicos comprados por cada cliente.

Propósito: Filtrar en la cláusula HAVING para asegurar que solo se incluyan clientes que han comprado todos los productos disponibles.

b. La subconsulta (SELECT COUNT(DISTINCT Producto) FROM ORDENES\_NEW)

Función: Calcula el número total de productos distintos en la tabla ORDENES\_NEW.

Propósito: Proporcionar un valor de referencia que se compara con el conteo de productos únicos de cada cliente en la cláusula HAVING, ayudando a determinar si el cliente ha adquirido todos los productos.

## **Scripts Proporcionados**

### **Primera parte:**

-- Creación de la tabla Clientes

```
CREATE TABLE CLIENTES (  
  ID INT CONSTRAINT PK_ID_CLI PRIMARY KEY,  
  Nombre VARCHAR(50),  
  Apellido VARCHAR(50),  
  Ciudad VARCHAR(50)  
);
```

-- Creación de la tabla Proveedores

```
CREATE TABLE PROVEEDORES (  
  ID INT CONSTRAINT PK_ID_PRO PRIMARY KEY,  
  Nombre VARCHAR(50),  
  Ciudad VARCHAR(50)  
);
```

-- Creación de la tabla Productos

```
CREATE TABLE PRODUCTOS (  
  ID INT CONSTRAINT PK_ID_PROD PRIMARY KEY,  
  Nombre VARCHAR(50),  
  Precio DECIMAL(10,2)  
);
```

-- Creación de la tabla Ordenes

```
CREATE TABLE ORDENES (  
  ID INT CONSTRAINT PK_ID_ORD PRIMARY KEY,  
  ClienteID INT,  
  ProductoID INT,  
  Cantidad INT,  
  CONSTRAINT FK_CLIENTES_ClienteID FOREIGN KEY (ClienteID)  
  REFERENCES CLIENTES(ID),  
  CONSTRAINT FK_PRODUCTOS_ProductoID FOREIGN KEY (ProductoID)  
  REFERENCES
```

Productos(ID)

);

-- Inserción de datos en la tabla Clientes

INSERT INTO CLIENTES (ID, Nombre, Apellido, Ciudad) VALUES

(1, 'Juan', 'Pérez', 'Madrid'),

(2, 'María', 'González', 'Barcelona'),

(3, 'Luis', 'Martínez', 'Valencia'),

(4, 'Ana', 'López', 'Madrid');

-- Inserción de datos en la tabla Proveedores

INSERT INTO Proveedores (ID, Nombre, Ciudad) VALUES

(1, 'Proveedor A', 'Madrid'),

(2, 'Proveedor B', 'Barcelona'),

(3, 'Proveedor C', 'Valencia'),

(4, 'Ana López', 'Sevilla'); -- Nombre compartido con un cliente para ejemplos de intersección.

-- Inserción de datos en la tabla Productos

INSERT INTO PRODUCTOS (ID, Nombre, Precio) VALUES

(1, 'Producto 1', 10.00),

(2, 'Producto 2', 20.00),

(3, 'Producto 3', 15.00),

(4, 'Producto 4', 25.00);

-- Inserción de datos en la tabla Ordenes

INSERT INTO ORDENES (ID, ClienteID, ProductoID, Cantidad) VALUES

(1, 1, 1, 20),

(2, 1, 3, 10),

(3, 2, 2, 15),

(4, 3, 4, 55),

(5, 4, 2, 25),

(6, 4, 3, 33);

## Segunda parte:

```
SELECT ClienteID, Nombre
FROM ORDENES_NEW ORN INNER JOIN CLIENTES_NEW CLN ON
ORN.ClienteID= CLN.ID
GROUP BY ClienteID, Nombre
HAVING COUNT(DISTINCT Producto) = (SELECT COUNT(DISTINCT
Producto) FROM ORDENES_NEW);
```

-- Script suministrado: Crear la tabla Clientes

```
CREATE TABLE CLIENTES_NEW (
  ID INT CONSTRAINT PK_ID_CLINEW PRIMARY KEY,
  Nombre VARCHAR (50) NOT NULL,
  Ciudad VARCHAR (50) NULL,
  Correo VARCHAR (100) UNIQUE
);-- Insertar datos en la tabla Clientes
INSERT INTO CLIENTES_NEW (ID, Nombre, Ciudad, Correo) VALUES
(1, 'Juan', 'Madrid','a@'),
(2, 'Ana', 'Barcelona','b@'),
(3, 'Luis', 'Sevilla','c@'),
(4, 'María', 'Valencia','d@'),
(5, 'Pedro', 'Valencia','e@'),
(6, 'Carlos', 'Madrid','f@'),
(7, 'Lucía', 'Sevilla','g@');
```

- Crear la tabla Ordenes

```
CREATE TABLE ORDENES_NEW (
  ID INT PRIMARY KEY,
  ClienteID INT,
  Fecha DATE,
  Producto VARCHAR(50),
  CONSTRAINT FK_ORDENES_NEW_ClienteID FOREIGN KEY (ClienteID)
REFERENCES
CLIENTES_NEW (ID)
);
```

-- Insertar datos en la tabla Ordenes (incluyendo productos repetidos)

INSERT INTO ORDENES\_NEW (ID, ClienteID, Fecha, Producto) VALUES

(1, 1, '2023-09-06', 'Laptop'),  
(2, 1, '2023-09-07', 'Tablet'),  
(3, 2, '2023-09-01', 'Laptop'),  
(4, 2, '2023-09-02', 'Tablet'),  
(5, 2, '2023-09-03', 'Monitor'),  
(6, 2, '2023-09-04', 'Teclado'),  
(7, 2, '2023-09-05', 'Ratón'),  
(8, 3, '2023-09-08', 'Laptop'),  
(9, 3, '2023-09-09', 'Monitor'),  
(10, 4, '2023-09-10', 'Teclado'),  
(11, 4, '2023-09-11', 'Ratón'),  
(12, 5, '2023-09-12', 'Laptop'),  
(13, 5, '2023-09-13', 'Tablet'),  
(14, 6, '2023-09-14', 'Laptop'),  
(15, 6, '2023-09-15', 'Tablet'),  
(16, 6, '2023-09-16', 'Monitor'),  
(17, 7, '2023-09-17', 'Laptop'),  
(18, 7, '2023-09-18', 'Tablet'),  
(19, 7, '2023-09-19', 'Monitor'),  
(20, 7, '2023-09-20', 'Teclado');

## **Conclusión del tema**

Este taller ha permitido explorar la intersección entre las bases de datos y el álgebra relacionales, destacando cómo los conceptos matemáticos subyacentes se implementan en la práctica a través de consultas SQL.

Las bases de datos relacionales utilizan estructuras tabulares para almacenar datos, permitiendo relaciones entre diferentes entidades. El álgebra relacional proporciona un marco teórico que describe cómo manipular y consultar estos datos de manera efectiva.

Cada operación de álgebra relacional —como selección, proyección, unión, intersección y agrupación— tiene su contraparte en SQL, lo que permite a los usuarios ejecutar consultas que transforman y extraen información relevante.

En este taller, se ha ilustrado cómo la utilización de funciones de agregado y cláusulas como **GROUP BY** y **HAVING** en SQL se relaciona directamente con las operaciones de agrupación y filtrado en álgebra relacional.

A través de ejemplos concretos, hemos visto cómo se pueden aplicar estos principios para resolver problemas prácticos, como identificar clientes que han adquirido todos los productos disponibles.

Comprender la relación entre las bases de datos y el álgebra relacionales no solo fortalece el conocimiento teórico, sino que también mejora la capacidad para diseñar y ejecutar consultas efectivas en sistemas de gestión de bases de datos, optimizando así el manejo de la información en entornos reales.



