**Teoría de Conjuntos**

* Al haber abarcado el tema de relaciones entre tablas y normalización, se debe considerar cómo es que los datos van a ser accesible a través de múltiples tablas. Para esto, es importante conocer conceptos básicos de la Teoría de Conjuntos

**¿Qué es la teoría de conjuntos?**

Es una rama de las matemáticas que se dedica al estudio de las propiedades y relaciones de los conjuntos. Un conjunto es una colección bien definida de objetos, conocidos como elementos.

Por ejemplo

Un conjunto puede ser Colores: {rojo, verde, azul, amarillo, naranja}

Otro conjunto puede ser Frutas: {manzana, piña, fresa, naranja, uva}

Con los conjuntos se pueden realizar varias operaciones, lo cual es fundamental para comprender cómo se manipularán las tablas para obtener la información deseada. Estas operaciones nos proporcionan una mejor noción sobre la estructura y relación de los datos en una base de datos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operación | Descripción | Simbolo | Ejemplo | Resultado |
| Unión | Combina los elementos de ambos conjuntos. | U | Colores U Frutas | {rojo, verde, azul, amarillo, naranja, manzana, piña, fresa, uva} |
| Intersección | Agrupa solo los elementos que están en ambos conjuntos. | ∩ | Colores ∩ Frutas | {naranja} |
| Diferencia | Resta los elementos de un conjunto del otro. | - | Colores - Frutas | {rojo, verde, azul, amarillo} |
| Complemento | Extrae todos los elementos de un conjunto que no están en otro. | C() | Primarios: {rojo, amarillo,azul} C(Primarios) | {verde, naranja} |

**Joins e intersecciones**

Al trabajar en SQL, utilizamos la operación JOIN para realizar la intersección entre dos tablas. De esta forma podemos “cruzar” los datos de dos tablas para realizar la operación que necesitemos.

Ejemplo

Existen tres principales tipos de JOINs y dos más que no se usan comúnmente, pero que abarcaremos por pura utilidad.   
Se utilizarán las siguientes tablas para crear el ejemplo.

Cars

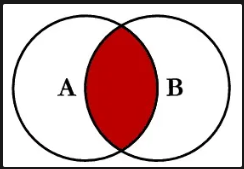
| **PlateNum** | **CustomerID** |
| --- | --- |
| BCJ953 | 1 |
| JKL123 | 2 |
| YRN395 | NULL |

Customers

| **CustomerID** | **Name** |
| --- | --- |
| 1 | John Doe |
| 2 | Jane Doe |
| 3 | Luke Skywalker |

**INNER JOIN**

* El INNER JOIN consiste en una intersección entre dos tablas, retornando solamente los datos que coinciden en la relación.
* Generalmente es el más utilizado por la forma en la que retorna los resultados, ya que evita los NULLs en la respuesta.



Ejemplo

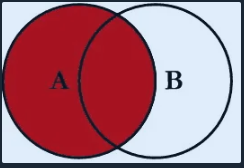
SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars  
INNER JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| BCJ953 | John Doe |
| JKL123 | Jane Doe |

LEFT JOIN

* El LEFT JOIN también realiza una intersección, pero en este caso, sol incluye los elementos de “la tabla de la izquierda” aunque no existan matches en la relación.
* Se le llama “tabla de la izquierda” a la tabla sobre la que está haciendo el SELECT



Ejemplo

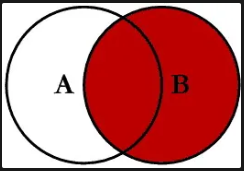
SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars  
LEFT JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| BCJ953 | John Doe |
| JKL123 | Jane Doe |
| YRN395 | NULL |

**RIGHT JOIN**

* El RIGHT JOIN funciona de la misma forma que el LEFT JOIN, pero incluyendo los valores NULL de la tabla de la derecha
* La tabla de la derecha es la tabla sobre la que se realiza el JOIN.



Ejemplo

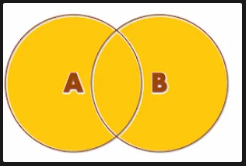
SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars  
RIGHT JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID

Result

| **PlateNum** | **Name** |
| --- | --- |
| BCJ953 | John Doe |
| JKL123 | Jane Doe |
| NULL | Luke Skywalker |

**FULL OUTER JOIN**

* El FULL OUTER JOIN es uno de los menos utilizados, ya que este incluye a todas las filas en los resultados, sin importar si los datos son NULL



Ejemplo

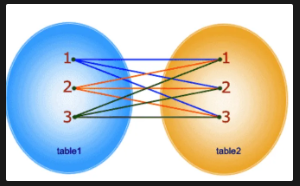
SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars  
FULL OUTER JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| BCJ953 | John Doe |
| JKL123 | Jane Doe |
| YRN395 | NULL |
| NULL | Luke Skywalker |

**CROSS JOIN**

* El CROSS JOIN es una de las operaciones menos usadas en SQL, pero es util en algunos casos.
* Esta operación realiza el producto cartesiano sobre dos tablas. Para simplificar el ejemplo, veamos las siguientes tablas:



* Socks

| **Brand** | **Color** |
| --- | --- |
| Nike | White |
| Puma | Black |

* Sandals

| **Brand** | **Color** |
| --- | --- |
| Birks | Tan |
| Crocs | Blue |

Ejemplo

SELECT Socks.Brand as sock\_brand,   
Socks.Color as sock\_color,   
Sandals.Brand as sandal\_brand,   
Sandals.Color as sandal\_color  
FROM Socks  
CROSS JOIN Sandals

Resultado

| **sock\_brand** | **sock\_color** | **sandal\_brand** | **sandal\_color** |
| --- | --- | --- | --- |
| Nike | White | Birks | Tan |
| Puma | Black | Crocs | Blue |
| Nike | White | Crocs | Blue |
| Puma | Black | Birks | Tan |

El CROSS JOIN realizó todas las posibles combinaciones entre las filas de la tabla, resultando en cuatro combinaciones diferentes de medias y sandalias.

🤔 ¿Por qué nos interesan los JOINs que devuelven NULLs?

Muchas veces cuando estamos diseñando un query, nos vamos directamente al uso del INNER JOIN para evitar los datos NULL en el resultado. Es importante que también conozcamos la utilidad de los JOINs que retornan datos NULL.

Por ejemplo: Consideremos el caso en el que se quiere obtener todos los carros que no tienen un cliente asignado.   
Partamos del ejemplo del LEFT JOIN

Query

SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars  
LEFT JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| BCJ953 | John Doe |
| JKL123 | Jane Doe |
| YRN395 | NULL |

En el resultado se puede observa ver que se obtiene un carro que no tiene cliente asignado.   
Agregando una simple condición se pueden eliminar todas las filas que no se necesitan, en este caso los carros asociados a un cliente

Query

SELECT cars.PlateNum, customers.Name   
FROM Cars AS cars LEFT JOIN Customers AS customers  
ON cars.CustomerID = customers.CustomerID WHERE customers.Name IS NULL

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| YRN395 | NULL |

Con este resultado ahora sí cumplimos la especificación de listar los carros que no tienen un cliente asociado.   
De la misma forma, pero con los clientes, se puede utilizar el RIGHT JOIN para obtener todos los clientes que no tienen un carro asignado:

Query

SELECT cars.PlateNum, customers.Name FROM Cars AS cars

RIGHT JOIN Customers AS customers ON cars.CustomerID = customers.CustomerID

WHERE cars.PlateNum IS NULL

Resultado

| **PlateNum** | **Name** |
| --- | --- |
| NULL | Luke Skywalker |

Por lo tanto, siempre es importante conocer y tratar de buscar todas las posibles utilidades de una función, antes de centrarnos en utilizar una única opción.

**Conclusión**

La teoría de conjuntos es fundamental para entender cómo se manipulan y relacionan grupos de datos.

* Nos ayudan a utilizar eficazmente las relaciones de datos en bases de datos.
* En SQL, utilizamos funciones de JOIN para relacionar datos entre tablas

Existen varios tipos de JOINs que sirven para diferentes propósitos:

* INNER JOIN: Retorna registros que tienen coincidencias en ambas tablas.
* LEFT JOIN: Retorna todos los registros de la tabla izquierda y los registros coincidentes de la derecha.
* RIGHT JOIN: Retorna todos los registros de la tabla derecha y los registros coincidentes de la izquierda.
* FULL OUTER JOIN: Retorna todos los registros cuando hay coincidencias en cualquiera de las tablas.
* CROSS JOIN: Retorna el producto cartesiano de ambas tablas.

Es una buena práctica seleccionar el tipo de JOIN adecuado para cada operación, ya que esto afecta la integridad y eficiencia de las consultas. Utilizar el JOIN incorrecto puede complicar y ralentizar las consultas SQL, afectando negativamente el rendimiento de la base de datos.

Los JOINs son herramientas poderosas para relacionar datos de manera eficiente y efectiva en SQL, permitiendo realizar análisis complejos y obtener información significativa de múltiples fuentes de datos.

