# Tutorial SQL

## ¿Qué es SQL?

SQL es un lenguaje de programación diseñado para administrar datos almacenados en un sistema de administración de bases de datos relacionales (RDBMS).

SQL consta de un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos.

- El lenguaje de definición de datos se ocupa de la creación y modificación del esquema, por ejemplo, la instrucción CREATE TABLE le permite crear una nueva tabla en la base de datos y la instrucción ALTER TABLE cambia la estructura de una tabla existente.
- El lenguaje de manipulación de datos proporciona las construcciones para consultar datos como la instrucción SELECT y para actualizar los datos como INSERT , UPDATE y DELETE .
- El lenguaje de control de datos consta de las declaraciones que se ocupan de la autorización y la seguridad del usuario, como las declaraciones GRANT y REVOKE.

### Sintaxis SQL

SQL es un lenguaje declarativo, por lo tanto, su sintaxis se lee como un lenguaje natural. Una declaración SQL comienza con un verbo que describe la acción, por ejemplo, SELECT , INSERT , UPDATE o DELETE . Después del verbo están el sujeto y el predicado.

# SQL básico

## SELECT

Para consultar datos de una tabla, usa la instrucción SELECT de SQL. La declaración SELECT contiene la sintaxis para seleccionar columnas, seleccionar filas, agrupar datos, unir tablas y realizar cálculos simples.

A continuación, se ilustra la sintaxis básica de la declaración SELECT que recupera datos de una sola tabla.

```
SELECT nombre_de_columna
FROM nombre_de_tabla
```

En esta sintaxis:

- Primero, especifique una lista de columnas separadas por comas de las que desea consultar los datos en la cláusula SELECT.
- Luego, especifique el nombre de la tabla en la cláusula FROM.

Al evaluar la declaración SELECT, el sistema de base de datos evalúa primero la cláusula FROM y luego la cláusula SELECT.

En caso de que desee consultar datos de todas las columnas de una tabla, puede usar el operador asterisco (*), así:

```
SELECT *
FROM nombre_de_tabla
```

Tenga en cuenta que SQL no distingue entre mayúsculas y minúsculas. Significa que las palabras clave SELECTy selectson las mismas.

# ORDER BY

Cuando usa la instrucción SELECT para consultar datos de una tabla, el orden en que aparecen las filas en el conjunto de resultados puede no ser el esperado.

En algunos casos, las filas que aparecen en el conjunto de resultados están en el orden en que se almacenan físicamente en la tabla. Sin embargo, en caso de que el optimizador de consultas utilice un índice para procesar la consulta, las filas aparecerán tal como están almacenadas en el orden de la clave del índice. Por esta razón, el orden de las filas en el conjunto de resultados es indeterminado o impredecible.

Para especificar exactamente el orden de las filas en el conjunto de resultados, agregue una cláusula ORDER BY en la declaración del SELECT de la siguiente manera:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
ORDER BY nombre_de_columna ASC
```

En esta sintaxis, la cláusula ORDER BY aparece después de la cláusula FROM. En caso de que la declaración SELECT contenga una cláusula WHERE, la cláusula ORDER BY debe aparecer después de la cláusula WHERE.

Para ordenar el conjunto de resultados, especifique la columna en la que desea ordenar y el tipo de orden de clasificación:

- Ascendente (ASC)
- Descendente (DESC)

Si no especifica el orden de clasificación, el sistema de base de datos normalmente clasifica el conjunto de resultados en orden ascendente (ASC) de forma predeterminada.

Cuando incluye más de una columna en la cláusula ORDER BY, el sistema de base de datos primero clasifica el conjunto de resultados según la primera columna y luego clasifica el conjunto de resultados ordenados según la segunda columna, y así sucesivamente.

# DISTINCT

La clave principal asegura que la tabla no tenga filas duplicadas. Sin embargo, cuando se usa la declaración SELECT para consultar una parte de las columnas en una tabla, puede obtener duplicados.

Para eliminar duplicados de un conjunto de resultados, utilice el operador DISTINCT en la cláusula SELECT de la siguiente manera:

```
SELECT DISTINCT nombre_de_columna
FROM nombre_de_tabla
ORDER BY nombre_de_columna ASC
```

Si usa una columna después del operador DISTINCT, el sistema de base de datos usa esa columna para evaluar el duplicado. En caso de que use dos o más columnas, el sistema de base de datos usará la combinación de valor en estas columnas para la verificación de duplicación.

# LIMIT

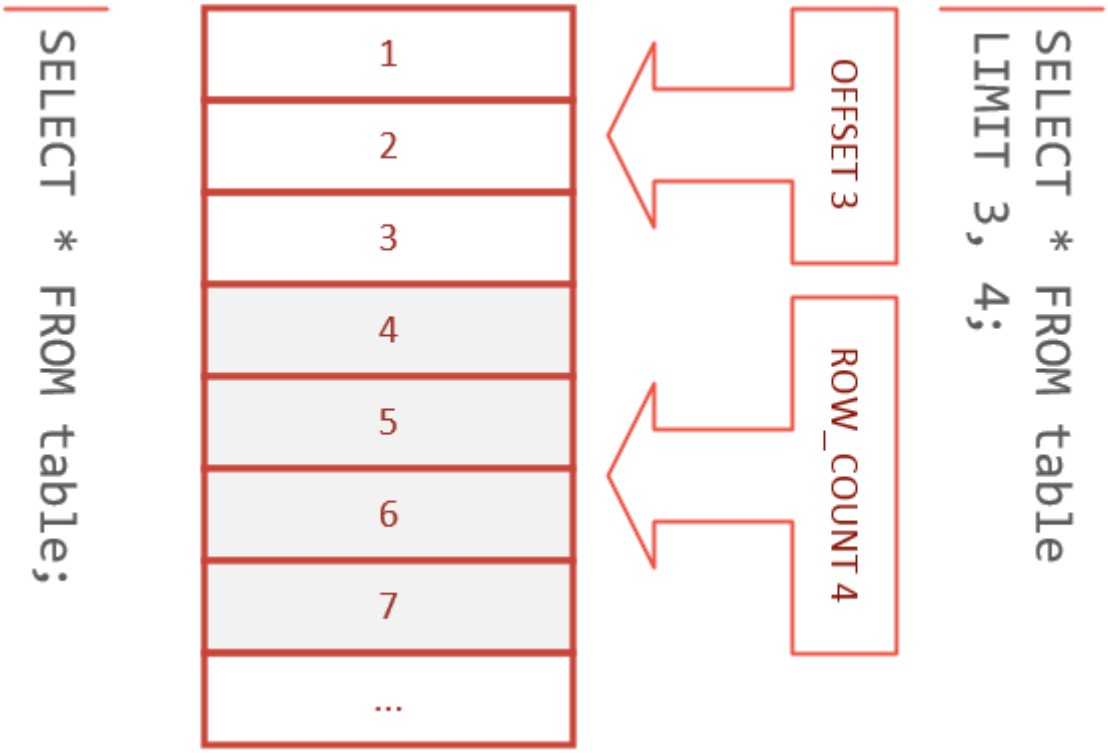
Para recuperar una parte de las filas devueltas por una consulta , use las cláusulas LIMIT y OFFSET. A continuación, se ilustra la sintaxis de estas cláusulas:

```
SELECT DISTINCT nombre_de_columna
FROM nombre_de_tabla
LIMIT cantidad_de_filas OFFSET filas_a_omitir
```

En esta sintaxis:

- La "cantidad_de_filas" determina el número de filas que se devolverán.
- La cláusula OFFSET omite las "filas_a_omitir" filas antes de comenzar a devolver las filas. La cláusula OFFSET es opcional, por lo que puede omitirla. Si usa ambas cláusulas LIMIT y OFFSET, primero se saltan las filas antes de restringir el número de filas.

Cuando usa la cláusula LIMIT, es importante usar una cláusula ORDER BY para asegurarse de que las filas de la devolución estén en un orden específico.




# WHERE

Para seleccionar ciertas filas de una tabla, usa una cláusula WHERE en la declaración SELECT. A continuación, se ilustra la sintaxis de la cláusula WHERE en la declaración SELECT:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE condición
```

La cláusula WHERE aparece inmediatamente después de la cláusula FROM. La cláusula WHERE contiene una o más expresiones lógicas que evalúan cada fila de la tabla. Si una fila que causa la condición se evalúa como verdadera, se incluirá en el conjunto de resultados; de lo contrario, será excluido.

Tenga en cuenta que SQL tiene una lógica de tres valores que es VERDADERO, FALSO y DESCONOCIDO. Significa que si una fila hace que la condición se evalúe como FALSE o NULL, la fila no se devolverá.

Tenga en cuenta que la expresión lógica que sigue a la cláusula WHERE también se conoce como predicado. Puede utilizar varios operadores para formar los criterios de selección de filas que se utilizan en la cláusula WHERE.

# OPERADORES DE COMPARACIÓN

La forma más básica de filtrar datos es utilizar operadores de comparación

La siguiente tabla muestra los operadores de comparación SQL :

| Operador | Significado |
|---|---|
| = | Igual |
| <> (!=) | Distinto |
| > | Mayor |
| >= | Mayor o igual |
| < | Menor |
| <= | Menor o igual |

Para formar una expresión simple, utilice uno de los operadores anteriores con dos operandos que pueden ser el nombre de la columna en un lado y un valor literal en el otro, por ejemplo:

```
columna_salario > 1000
```

Hace una pregunta: "¿El salario es superior a 1000?".

O puede usar nombres de columna en ambos lados de un operador como:

```
columna_salario_min < columna_salario_max
```

Esta expresión plantea otra pregunta: "¿El salario mínimo es menor que el salario máximo?".

# OPERADORES LÓGICOS

Los operadores de comparación permiten filtrar datos según una condición, aunque es probable que también desee filtrar datos mediante varias condiciones, posiblemente con más frecuencia de la que desea filtrar por una sola condición. Los operadores lógicos le permiten utilizar varios operadores de comparación en una consulta.

Cada operador lógico es un copo de nieve especial, los cuales veremos a continuación:

| Operador | Significado |
|---|---|
| AND | El operador AND muestra un registro si tanto la primera condición como la la segunda condición es verdadera. |
| OR | El operador OR muestra un registro si la primera condición o la la segunda condición es verdadera. |
| BETWEEN | El operador BETWEEN le permite seleccionar solo filas dentro de un cierto rango. |
| IN | El operador IN le permite especificar una lista de valores que le gustaría incluir. |
| LIKE | El operador LIKE le permite hacer coincidir valores similares, en lugar de valores exactos. |
| NOT | El operador NOT le permite seleccionar filas que no coinciden con una determinada condición. |
| IS NULL | El operador IS NULL le permite seleccionar filas que no contienen datos en una columna determinada. |

## AND

El operador AND le permite construir múltiples condiciones de la cláusula WHERE de una instrucción SQL como SELECT, UPDATE y DELETE:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE condición AND condición
```

El operador AND devuelve verdadero si ambas expresiones se evalúan como verdaderas. Si una de las dos expresiones es falsa, el operador AND devuelve falso incluso si una de las expresiones es NULL.

La siguiente tabla ilustra los resultados del operador AND al comparar valores verdaderos, falsos y NULL:

| AND | VERDADERO | FALSO | NULL |
|---|---|---|---|
| **VERDADERO** | VERDADERO | FALSO | NULL |
| **FALSO** | FALSO | FALSO | FALSO |
| **NULL** | NULL | FALSO | NULL |

## OR

De manera similar al operador AND, el operador OR combina múltiples condiciones en una cláusula WHERE de declaración SQL:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE condición OR condición
```

Sin embargo, el operador OR devuelve verdadero si al menos una expresión se evalúa como verdadera.

La siguiente tabla muestra el resultado del operador OR cuando comparamos los valores verdadero, falso y NULO.

| OR | VERDADERO | FALSO | NULL |
|---|---|---|---|
| **VERDADERO** | VERDADERO | VERDADERO | VERDADERO |
| **FALSO** | VERDADERO | FALSO | NULL |
| **NULL** | VERDADERO | NULL | NULL |

## BETWEEN

El operador BETWEEN es un operador lógico. Devuelve un valor de verdadero, falso o desconocido. El operador BETWEEN se utiliza en la cláusula WHERE de la declaración SELECT, DELETE o UPDATE, para encontrar valores dentro de un rango.

El operador BETWEEN selecciona un rango de datos entre dos valores. Los valores pueden ser números o fechas.

A continuación, se ilustra la sintaxis del operador BETWEEN:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE expresión BETWEEN valor_inferior AND valor_superior
```

En esta sintaxis:

- expresión: la expresión para probar en el rango definido por valor_inferior y valor_superior.

- valor_inferior y valor_superior: pueden ser expresiones o valores literales con el requisito de que el valor de valor_inferior sea menor que el valor de valor_superior.

## IN

El operador IN es un operador lógico que le permite comparar un valor con un conjunto de valores. El operador IN devuelve verdadero si el valor está dentro del conjunto de valores. De lo contrario, devuelve falso o desconocido.

A continuación, se ilustra la sintaxis del INoperador:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE expresión IN (valor_1, valor_2, ...)
```

## LIKE

A veces, es útil probar si una expresión coincide con un patrón específico, por ejemplo, para encontrar todos los empleados cuyos nombres comienzan con "J". En estos casos, debe utilizar el operador LIKE.

El operador LIKE prueba si una expresión coincide con un patrón específico. Consulte la siguiente sintaxis:

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE expresión LIKE patrón
```

Si la expresión coincide con el patrón, el operador LIKE devuelve verdadero. De lo contrario, devuelve falso.

Para construir un patrón, usa dos caracteres comodín SQL:

- % el signo de porcentaje coincide con cero, uno o más caracteres
- _ El signo de subrayado coincide con un solo carácter.

La siguiente tabla ilustra algunos patrones y sus significados:

| **Expresión** | **Sentido** |
|---|---|
| LIKE 'Kim%' | Comienza con 'Kim' |
| LIKE '%er' | Termina con 'er' |
| LIKE '%ch%' | Contiene 'ch' |
| LIKE 'Le_' | Comineza con 'Le' y le sigue solo un carácter |
| LIKE '_uy' | Termina con 'uy' y comienza con un solo carácter |
| LIKE '%are_' | Contiene 'are', comienza con cualquier número de caracteres y termina con un carácter |
| LIKE '_are%' | Contiene 'are', comienza con un carácter y termina con cualquier número de caracteres |

## NOT

Para invertir el resultado de cualquier expresión booleana, utilice el operador NOT. A continuación se ilustra cómo utilizar el operador NOT.

```
SELECT nombre_de_columna
FROM nombre_de_tabla
WHERE NOT expresión_boleana
```

La siguiente tabla muestra el resultado del operador NOT.

| | NOT |
|---|---|
| **VERDADERO** | FALSO |
| **FALSO** | VERDADERO |
| **NULL** | NULL |

### IS NULL

La idea de Null se introdujo en SQL para manejar la información faltante en el modelo relacional. La introducción de Null (o Unknown) junto con verdadero y falso es la base de la lógica de tres valores. Nulo no tiene un valor (y no es miembro de ningún dominio de datos) sino más bien un marcador de posición o "marca" para la información que falta. Por lo tanto comparaciones con Null nunca puede resultar en verdadero o falso, pero siempre en el tercer resultado lógico.

Si una columna en una tabla es opcional, podemos insertar un nuevo registro o actualizar un registro existente sin agregar un valor a esta columna. Esto significa que el campo se guardará con un valor nulo.

Los valores nulos se tratan de forma diferente a otros valores. Null se utiliza como marcador de posición para valores desconocidos o inaplicables.

Como consecuencia:

- Nulo <> Nulo o Nulo <> X? Desconocido.
- Nulo = Nulo o Nulo = X? Desconocido.

Dado que los operadores SQL devuelven desconocido al comparar cualquier cosa con Nulo, SQL proporciona dos predicados de comparación específicos de nulo: IS NULL y IS NOT prueba si los datos son o no nulos.

En consecuencia, NULL es especial en SQL. NULL indica que los datos son desconocidos, inaplicables o incluso que no existen. En otras palabras, NULL representa que faltan datos en la base de datos.

Por ejemplo, si un empleado no tiene ningún número de teléfono, puede almacenarlo como una cadena vacía. Sin embargo, si no sabemos su número de teléfono al momento de insertar el registro del empleado, usaremos el valor NULL para los números de teléfono desconocidos.

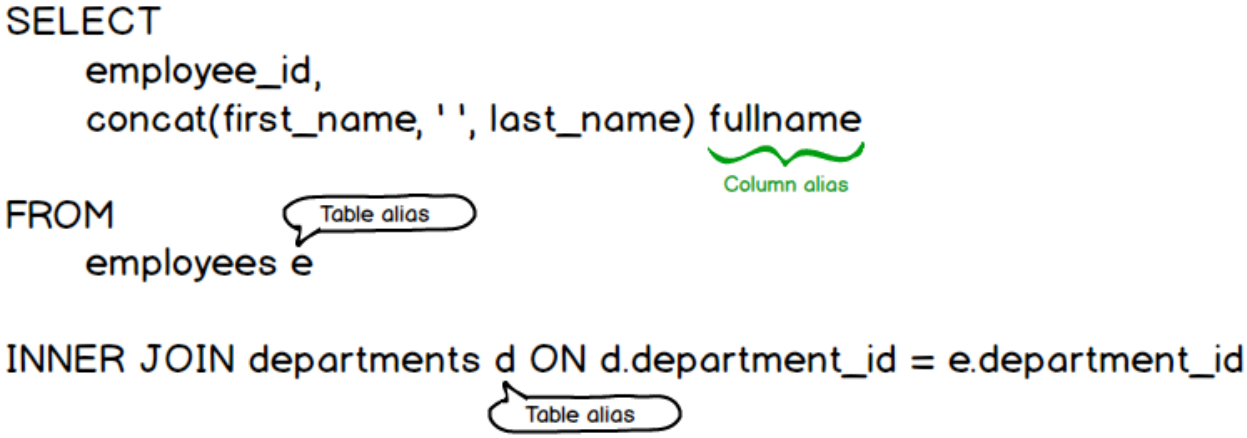
El valor NULL es especial porque cualquier comparación con un valor NULL nunca puede dar como resultado verdadero o falso, sino un tercer resultado lógico, desconocido.

# SQL intermedio

## ALIAS

El alias SQL le permite asignar un nombre temporal a una tabla o columna durante la ejecución de una consulta . Hay dos tipos de alias: alias de tabla y alias de columna.

Casi todos los sistemas de administración de bases de datos relacionales admiten tanto alias de columna como de tabla.




# PRODUCTO CARTESIANO

Para hacer un producto de cartesion, lo único que debe hacer es agregar más de una tabla en la instrucción FROM.

Sintaxis del producto cartesiano

```
SELECT nombre_tabla_1.nombre_de_columna, ..., nombre_tabla_2.nombre_de_columna, ...
FROM nombre_tabla_1, nombre_tabla_2
```

Supongamos que tenemos las tablas A, con las filas 1 y 2, y la tabla B, con las filas 2 y 3; el resultado del producto cartesiano será

| Tabla A | Tabla B |
|---|---|
| 1 | 2 |
| 2 | 3 |

| Producto Carteciano | |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 2 | 2 |
| 2 | 3 |

# JOIN

El proceso de vincular tablas se llama JOIN. SQL proporciona muchos tipos de combinaciones, como combinación interna (INNER JOIN), combinación izquierda (LEFT OUTER JOIN), combinación derecha (RIGHT OUTER JOIN), combinación externa completa (FULL OUTER JOIN), etc. Este tutorial se centra en la combinación interna.

Diferentes JOINs SQL

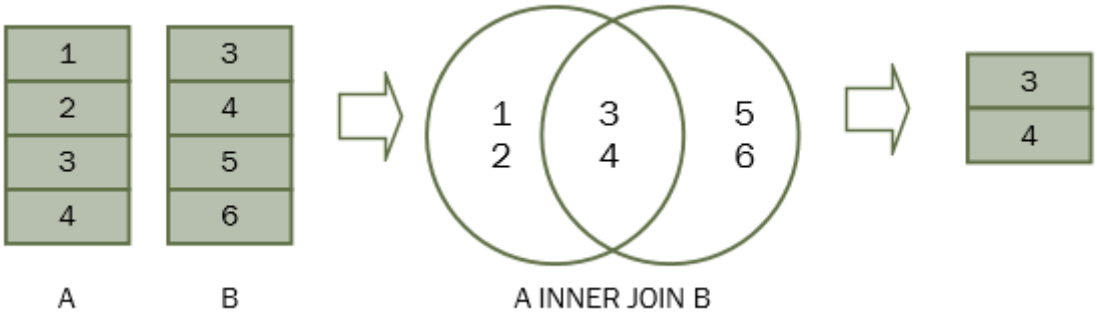
- INNER JOIN: Devuelve filas cuando hay al menos una coincidencia en ambos tablas
- OUTER JOIN: Devuelve filas cuando hay una coincidencia en una de las tablas
    - LEFT OUTER JOIN: Devuelve todas las filas de la tabla de la izquierda, incluso si no hay coincidencias en la tabla de la derecha.
    - RIGHT OUTER JOIN: Devuelve todas las filas de la tabla derecha, incluso si no hay coincidencias en la tabla de la izquierda.
    - FULL OUTER JOIN: Devuelve filas cuando hay una coincidencia en una de las tablas.

## INNER JOIN

La cláusula de INNER JOIN vincula dos (o más) tablas mediante una relación entre dos columnas. Siempre que usa la cláusula de combinación interna, normalmente piensa en la intersección .

Es mucho más fácil comprender el concepto de combinación interna a través de un ejemplo simple.

Supongamos que tenemos dos tablas A, con las filas 1, 2, 3 y 4, y B, con las filas 3, 4, 5 y 6.




La cláusula INNER JOIN aparece después de la cláusula FROM. La condición para coincidir entre la tabla A y la tabla B se especifica después de la palabra clave ON. Esta condición se llama condición de unión, es decir, B.n = A.n

La cláusula INNER JOIN puede unir tres o más tablas siempre que tengan relaciones, normalmente relaciones de clave externa.

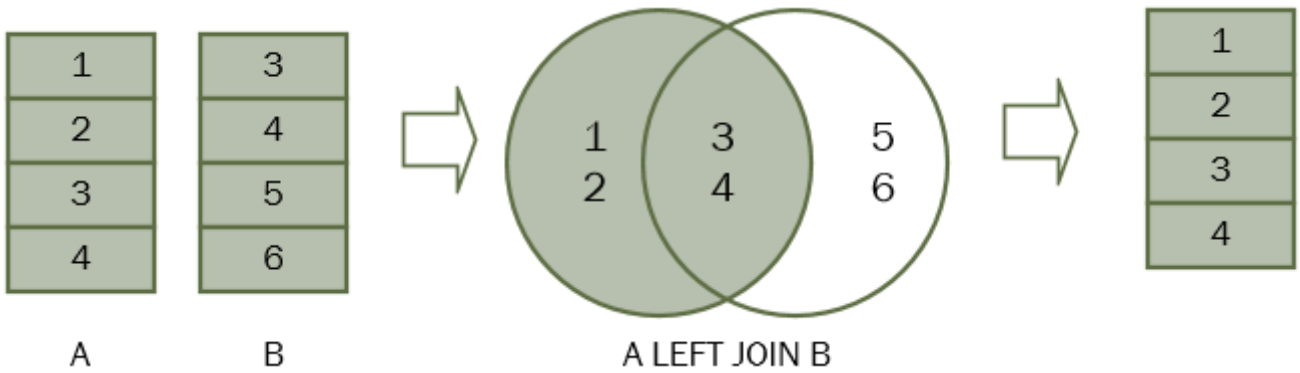
```
SELECT A.n, B.n
FROM A
     JOIN B ON A.x = B.x
```

## LEFT OUTER JOIN

La combinación a izquierda (LEFT OUTER JOIN) devuelve todas las filas de la tabla de la izquierda, haya o no una fila coincidente en la tabla de la derecha.

Suponga que tenemos dos tablas A y B. La tabla A tiene cuatro filas 1, 2, 3 y 4. La tabla B también tiene cuatro filas 3, 4, 5, 6.

Cuando unimos la tabla A con la tabla B, todas las filas de la tabla A (la tabla de la izquierda) se incluyen en el conjunto de resultados, ya sea que haya una fila coincidente en la tabla B o no.




En SQL, usamos la siguiente sintaxis para unir la tabla A con la tabla B.

```
SELECT A.n, B.n
FROM A
     LEFT OUTER JOIN B ON A.x = B.x
```

La cláusula LEFT JOIN aparece después de la cláusula FROM. La condición que sigue a la palabra clave ON se llama condición de uniónB.n = A.n

## RIGHT OUTER JOIN

La combinación a derecha (RIGHT OUTER JOIN) devuelve todas las filas de la tabla de la derecha, haya o no una fila coincidente en la tabla de la izquierda.

### FULL OUTER JOIN

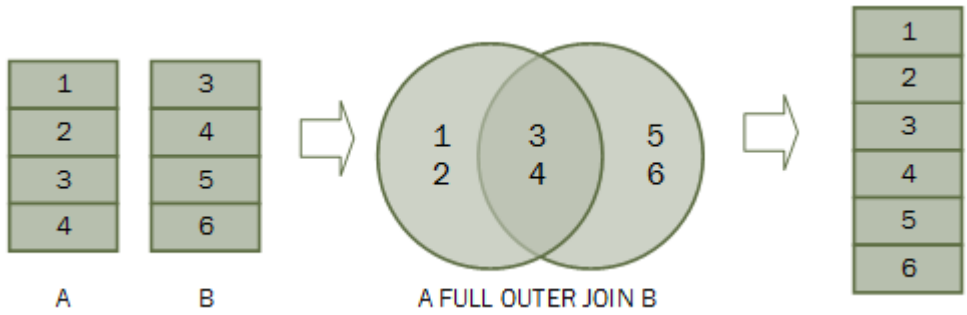
En teoría, una combinación externa completa (FULL OUTER JOIN) es la combinación de una combinación a izquierda y a derecha. La combinación externa completa incluye todas las filas de las tablas unidas, independientemente de que la otra tabla tenga la fila correspondiente o no.

Si las filas de las tablas unidas no coinciden, el conjunto de resultados de la combinación externa completa contiene valores NULL para cada columna de la tabla que carece de una fila coincidente. Para las filas coincidentes, se incluye en el conjunto de resultados una sola fila que tiene las columnas pobladas de la tabla unida.

La siguiente declaración ilustra la sintaxis de la combinación externa completa de dos tablas:

```
SELECT A.n, B.n
FROM A
     FULL OUTER JOIN B ON A.x = B.x
```

El siguiente diagrama de Venn ilustra la unión externa completa de dos tablas.




# FUNCIONES DE AGREGACIÓN

Las funciones agregadas de SQL devuelven un solo valor, calculado a partir de los valores en un columna.

Funciones agregadas útiles:

- AVG(): devuelve el valor promedio
- COUNT(): devuelve el número de filas
- FIST(): devuelve el primer valor
- LAST(): devuelve el último valor
- MAX(): devuelve el valor más grande
- MIN(): devuelve el valor más pequeño
- SUM(): devuelve la suma de los valores

# GROUP BY

La funciones de agregación de SQL como COUNT, AVG y SUM tienen algo en común: todas se agregan en toda la tabla. Pero, ¿qué sucede si desea agregar solo una parte de una tabla? Por ejemplo, es posible que desee contar el número de entradas de cada año.

En situaciones como esta, necesitaría usar la cláusula GROUP BY. GROUP BY le permite separar los datos en grupos, que se pueden agregar independientemente unos de otros.

Por tanto, un conjunto de agrupación es un conjunto de columnas por las que se agrupa mediante la cláusula GROUP BY. Normalmente, una sola consulta agregada define un solo conjunto de agrupaciones.

La sintaxis del GROUP BY es

```
SELECT nombre_de_columna, función_de_agregación()
FROM nombre_de_tabla
GROUP BY nombre_columna_agrupamiento
```

### HAVING

La cláusula HAVING se agregó a SQL porque la palabra clave WHERE no se pudo usar con funciones agregadas.

Para especificar una condición para grupos, use la cláusula HAVING.

La sintaxis del HAVING es

```
SELECT nombre_de_columna, función_de_agregación()
FROM nombre_de_tabla
GROUP BY nombre_columna_agrupamiento
         HAVING función_de_agrupación operador valor
```

## UNION

El operador UNION combina conjuntos de resultados de dos o más sentencias SELECT en un único conjunto de resultados. La siguiente declaración ilustra cómo utilizar el operador UNION para combinar conjuntos de resultados de dos consultas:

```
SELECT A.n, A.s
FROM A
    UNION
SELECT B.n, B.s
FROM B
```

Observe que cada instrucción SELECT dentro de UNION debe tener la mismo número de columnas. Las columnas también deben tener tipos de datos similares. Además, las columnas de cada instrucción SELECT deben estar en el mismo orden.

# SQL AVANZADO

## SUBCONSULTA

Por definición, una sub consulta es una consulta anidada dentro de otra consulta como SELECT, INSERT, UPDATE, o DELETE

```
SELECT A.n
FROM A
WHERE A.x in (SELECT B.X
              FROM B)
```

La consulta colocada entre paréntesis se denomina subconsulta. También se conoce como consulta interna o selección interna. La consulta que contiene la subconsulta se denomina consulta externa o selección externa.

Para ejecutar la consulta, primero, el sistema de base de datos tiene que ejecutar la subconsulta y sustituir la subconsulta entre paréntesis con su resultado y luego ejecuta la consulta externa.

Puede usar una subconsulta en muchos lugares, como:

- Con el operador IN o NOT IN

- Con operadores de comparación
- Con el operador EXISTS o NOT EXISTS
- En la cláusula FROM
- En la cláusula SELECT

# EXISTS

El operador EXISTS le permite especificar una subconsulta para probar la existencia de filas. A continuación, se ilustra la sintaxis del operador EXISTS:

```
SELECT A.n
FROM A
WHERE EXISTS (subconsulta)
```

El Soperador EXIST devuelve verdadero si la subconsulta contiene filas. De lo contrario, devuelve falso.

El operador EXISTS finaliza el procesamiento de la consulta inmediatamente una vez que encuentra una fila, por lo tanto, puede aprovechar esta función del operador EXISTS para mejorar el rendimiento de la consulta.

Si la subconsulta regresa NULL, el operador EXISTS aún devuelve el conjunto de resultados. Esto se debe a que el operador EXISTS solo verifica la existencia de una fila devuelta por la subconsulta. No importa si la fila lo es NULL o no.

El operador EXISTS es el único operador que permite utilizar la tabla de la selección externa en la selección interna (subconsulta).

```
SELECT A.n
FROM A
WHERE EXISTS (SELECT 1
              FROM B
              WHERE A.x = B.x)
```

# CASE

La expresión SQL CASE le permite evaluar una lista de condiciones y devuelve uno de los posibles resultados.

Puede usar la expresión CASE en una cláusula o declaración que permita una expresión válida. Por ejemplo, puede utilizar la expresión CASE en instrucciones como SELECT , DELETE y UPDATE o en cláusulas como SELECT, ORDER BY y HAVING.

En conclusión, la declaración CASE es la forma en que SQL maneja la lógica si / entonces (IF/THEN). La declaración CASE es seguido por al menos un par de declaraciones WHEN y THEN

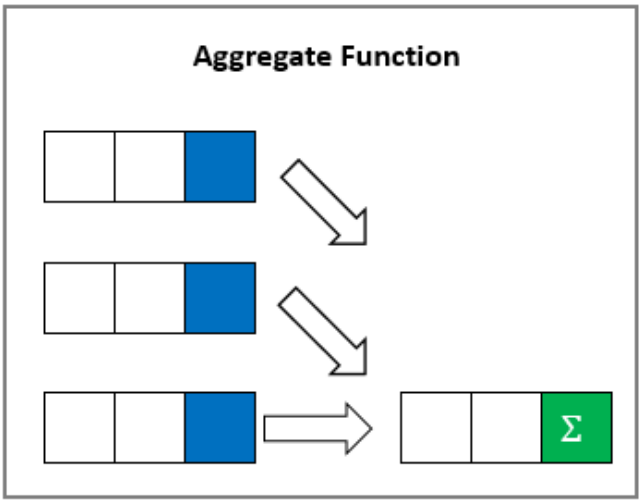
Cada declaración CASE debe terminar con la declaración END. La declaración ELSE es opcional y proporciona una forma de capturar valores no especificados en las declaraciones WHEN/THEN

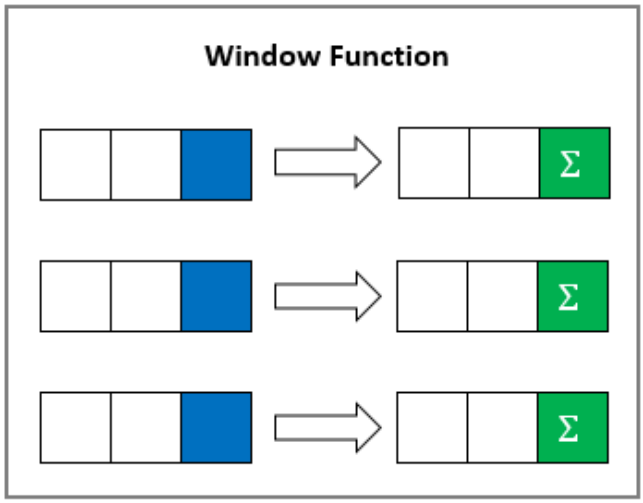
```
CASE expresión
     WHEN when_expresión_1 THEN resultado_1
     WHEN when_expresión_2 THEN resultado_2
     ...
     ELSE resultado_else
END
```

# FUNCIONES DE VENTANA

Las funciones agregadas realizan cálculos en un conjunto de filas y devuelven una única fila de salida.

Similar a una función agregada, una función de ventana calcula en un conjunto de filas. Sin embargo, una función de ventana no hace que las filas se agrupen en una sola fila de salida.




La sintaxis de las funciones de la ventana es la siguiente:

```
función_de_agrupación ( expresión ) OVER (
    cláusula_de_partición
    cláusula_de_orden
    cláusula_de_ventana
) nombre_de_columna_resultante
```

El nombre de la función de ventana compatible como ROW_NUMBER(), RANK(), y SUM() (AVG, MAX, MIN, ...).

La expresión o columna de destino en la que opera la función de ventana.

```
OVER cláusula
```

La cláusula OVER define las particiones de las ventanas para formar los grupos de filas y especifica el orden de las filas en una partición. La OVERcláusula consta de tres cláusulas: cláusulas de partición, orden y marco.

La cláusula de partición divide las filas en particiones a las que se aplica la función de ventana. Tiene la siguiente sintaxis:

```
PARTITION BY expr1, expr2, ...
```

Si la cláusula PARTITION BY no se especifica, todo el conjunto de resultados se trata como una sola partición.

La cláusula ORDER BY especifica los órdenes de las filas en una partición en la que opera la función de ventana:

```
ORDER BY expression [ASC | DESC]  [NULL {FIRST| LAST}]
```

Un frame es el subconjunto de la partición actual. Para definir el frame, utilice una de las siguientes sintaxis:

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

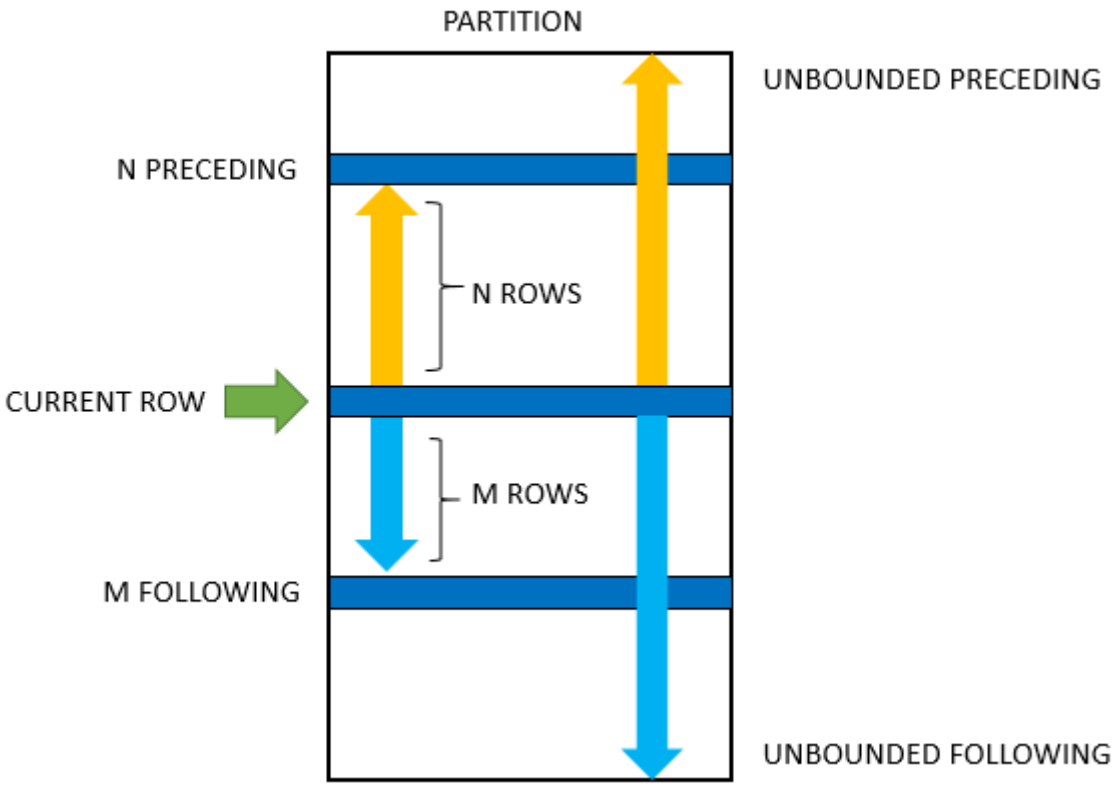
donde frame_startes es una de las siguientes opciones:

```
N PRECEDING
UNBOUNDED PRECEDING
CURRENT ROW
```

y frame_endes una de las siguientes opciones:

```
CURRENT ROW
UNBOUNDED FOLLOWING
N FOLLOWING
```

La siguiente imagen ilustra un marco y sus opciones:




- UNBOUNDED PRECEDING: el frame comienza en la primera fila de la partición.
- N PRECEDING: el frame comienza en enésimas filas antes de la fila actual.
- CURRENT ROW: significa la fila actual que se está evaluando.
- UNBOUNDED FOLLOWING: el frame termina en la última fila de la partición.
- N FOLLOWING: el frame termina en la fila Nh después de la fila actual.

ROWS o RANGE especifica el tipo de relación entre la fila actual y la filas del frame.

- ROWS: las compensaciones de la fila actual y las filas del frame son el números de filas.
- RANGE: el desplazamiento de la fila actual y las filas del frame son los valores de fila.

Las funciones de ventana se dividen en tres tipos de funciones de ventana de valor, funciones de ventana de agregación y funciones de ventana de clasificación:

Funciones de la ventana de valor

- FIRST_VALUE(): devuelve el primer valor en un conjunto ordenado de valores
- LAG(): proporciona acceso a una fila en un desplazamiento físico específico que viene antes de la fila actual
- LAST_VALUE(): devuelve el último valor en un conjunto ordenado de valores.
- LEAD(): proporciona acceso a una fila en un desplazamiento físico específico que sigue a la fila actual.

Funciones de la ventana de clasificación

- CUME_DIST(): calcula la distribución acumulada de valor dentro de un conjunto de valores.
- DENSE_RANK(): asigna filas a las filas de las particiones sin huecos en los valores de clasificación. Si dos o más filas en cada partición tienen los mismos valores, reciben el mismo rango. La siguiente fila tiene el rango aumentado en uno.
- NTILE(): permite dividir el conjunto de resultados en un número específico de grupos aproximadamente iguales o buckets. Asigna a cada grupo un número de bucket empezando por uno. Para cada fila de un grupo, la función NTILE() asigna un número de bucket que representa el grupo al que pertenece la fila.
- PERCENT_RANK(): calcula el percentil rango de filas de un conjunto de resultados.
- RANK(): asigna un rango a cada fila en la partición de un conjunto de resultados. El rango de una fila está determinado por uno más el número de rangos anteriores.
- ROW_NUMBER(): asigna un número entero secuencial para cada fila de conjunto de resultados de la consulta.

Funciones de ventana agregadas

- AVG(): calcula el promedio dentro de los valores de la ventana.
- COUNT(): cuenta la cantidad de elementos de la ventana.
- MAX(): retorna el valor máximo dentro de los valores de la ventana.
- MIN(): retorna el valor mínimo dentro de los valores de la ventana.
- SUM(): calcula la suma de los valores de la ventana.