



Tecnicatura Universitaria
en Programación

BASES DE DATOS I

Unidad Temática N° 1:
Recuperación de Datos

Teórico
1° Año – 2° Cuatrimestre



Índice

Recuperación de Datos	2
Consultas	2
Sentencia SELECT	2
La lista de selección: Cláusula SELECT	4
Ordenamiento de los resultados de una consulta	6
Condiciones de búsqueda en el WHERE	2
Condiciones de búsqueda compuestas.....	8
Condiciones de búsqueda compuestas.....	9
Combinación de Tablas.....	11
Composición de tablas en el FROM con JOIN	16
Combinación externa izquierda: LEFT JOIN	18
Combinación externa derecha: RIGHT JOIN	20
Funciones Incorporadas	25
Funciones para el manejo de cadenas.....	25
Funciones matemáticas	28
Funciones para el uso de fechas y horas	29
BIBLIOGRAFÍA	35

Recuperación de Datos

Consultas

Una consulta es una instrucción que permite recuperar datos contenidos en una base de datos. El motor de la base de datos la interpreta como una solicitud sobre el almacenamiento contenido en el servidor. Se recurre a ella cuando, por ejemplo, se quiere obtener salidas de información de tablas o de otros orígenes de datos como vistas, tablas temporales, subconsultas, funciones de tabla, etc. de una base de datos obteniendo un subconjunto de registros que cumplan con determinadas condiciones.

Probablemente consultar los datos, es una de las tareas que se llevan a cabo con mayor frecuencia. Cuando existen varios orígenes de datos con una cantidad considerable de registros y las relaciones entre estos orígenes son complejas, las consultas pueden llegar a ser complicadas, tardando en ejecutarse una cantidad de tiempo nada despreciable.

Esta es precisamente otra de las cuestiones que deben afrontarse, ya que no se trata únicamente de obtener los datos deseados, sino de hacerlo además en un tiempo razonable, optimizando recursos de red y procesamiento. El lenguaje SQL es muy flexible y permite que un mismo resultado pueda realizarse utilizando consultas de muy diversas formas.

Sentencia SELECT

La sentencia SELECT se utiliza para obtener un conjunto de registros que puede proceder de una o más orígenes de datos en formato de tablas.

Esta sentencia, que se utiliza para expresar consultas, es la más potente y compleja de las sentencias de SQL. La sentencia SELECT recupera datos de una base de datos y los devuelve en forma de tablas en memoria es decir quedan guardadas en el almacenamiento del sistema de cómputo. La sintaxis completa es la siguiente:

La sintaxis de la sentencia completa es:

```
SELECT [ DISTINCT | ALL ] lista_columnas  
[INTO nueva_tabla]  --Propio de SQL Server  
FROM lista_tablas  
[ WHERE condición ]  
[ GROUP BY lista_columnas ]
```

```
[ HAVING condición ]  
[ ORDER BY lista_columnas [ ASC | DESC ] ]
```

Se explicará en detalle cada una de las cláusulas de la sentencia SELECT a continuación

- **lista_columnas**: son las columnas que se desean mostrar en la tabla de resultado de la consulta en el orden en el que son especificadas, separadas por comas.
- **DISTINCT|ALL** : "DISTINCT" es una palabra clave se utiliza para eliminar filas duplicadas en el resultado de una consulta. Esto significa que solo se mostrarán las filas de la tabla de resultado que tengan contenido único. Por defecto, se asume "ALL", que muestra todas las filas del resultado de la consulta, incluyendo aquellas que tengan exactamente el mismo contenido que otra.
- **INTO nueva_tabla**: esta cláusula, propia de SQL Server, permite definir la creación de una nueva tabla a partir de la tabla de resultado de una consulta especificada. Se utiliza para crear tablas temporales o para hacer una copia de una tabla existente.
- **FROM lista_tablas**: esta cláusula especifica el origen de los datos con forma de tabla que pueden ser tablas bases, tablas temporales, vistas, subconsultas etc. . Si hay más de una tabla, se separan con comas y se debe prestar atención a la forma en que las tablas deben relacionarse o combinarse en la sentencia o bien puede utilizarse JOIN para establecer la combinación o composición de los distintos orígenes de datos.
- **WHERE condición**: esta cláusula se utiliza para aplicar algún tipo de condición que restrinja el número de registros devueltos por una consulta. Las condiciones de búsqueda suelen utilizarse para filtrar los resultados de una consulta en base a criterios específicos.
- **GROUP BY lista_columnas**: esta cláusula se utiliza para agrupar los resultados de una consulta en base a una o varias columnas.
- **HAVING condición**: esta cláusula se utiliza para filtrar los grupos generados por la cláusula "GROUP BY" en base a una condición específica.
- **ORDER BY lista_columnas**: se utiliza para ordenar los resultados de una consulta ordenados según uno o más criterios específicos. **ASC** se

utiliza para ordenar los resultados en orden ascendente la que se asume por defecto (de la A a la Z, de fecha más antiguas a fechas más recientes, o de menor a mayor según sean el tipo de dato de la columna) y **DESC** se utiliza para ordenar los resultados en orden contrario o sea descendente.

La lista de selección: Cláusula SELECT

- **lista_columnas:** se especifica el nombre de la/s columna/s de las cuales se quiere mostrar la información separadas por comas. Esta lista recupera y muestra las columnas en el orden especificado.

Por ejemplo: Teniendo las tablas del personas y tipos_dni, se quiere listar legajo, apellido, nombre y fecha de nacimiento desde la tabla personas

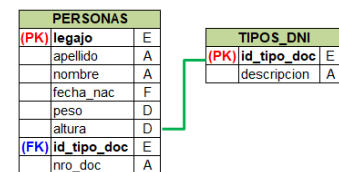


Ilustración 1: Elaboración propia.

```
select legajo, apellido, nombre, fecha_nac
from personas
```

La lista de columnas en la lista de selección va separada por comas, las columnas son los nombres de los campos que pertenecen a la tabla enunciada en el from. El resultado de esta consulta va a ser un listado con cuatro columnas.

legajo	apellido	nombre	fecha_nac
100	Rodríguez	Juan	1/5/1996
200	Pérez	Pedro	3/8/1984
300	Pérez	Ana	12/10/2010
400	Álvarez	Juana	3/9/1977
500	Sanchez	Juan	5/8/2002

Ilustración 2: Elaboración propia.

- **Uso del *:** Para emitir un listado de todos los campos de la/s tabla/s del from se incluye el * en la lista de selección. En el ejemplo se mostrarían todos los campos de la tabla personal

```
select * from personas
```

Se pueden crear columnas calculadas creando expresiones con operaciones matemáticas (+, -, *, / para sumar, restar, multiplicar y dividir) y funciones. También concatenar caracteres utilizando el signo más: '+'

Si se quiere el listado anterior con el nombre y apellido en una sola columna se puede usar el signo más para concatenar caracteres

```
select legajo, apellido + ', ' + nombre, fecha_nac
from personas
```

Lo que se va a lograr, en este caso, es ver en la 2da. columna el apellido del alumno sumado a una coma y un espacio y luego el nombre; la tabla de resultado va a contener tres columnas.

legajo	(no column name)	fecha_nac
100	Rodríguez, Juan	1/5/1996
200	Pérez, Pedro	3/8/1984
300	Pérez, Ana	12/10/2010
400	Álvarez, Juana	3/9/1977
500	Sanchez, Juan	5/8/2002

Ilustración 3: Elaboración propia.

- **Alias:** Si se presta atención al encabezado de cada columna de los ejemplos anteriores se verá que aparece en nombre del campo origen del dato o bien columna sin nombre.

Se puede agregar un alias a los campos que aparecerán como encabezados de columnas:

```
select legajo,
       apellido + ', ' + nombre as 'Nombre Completo',
       fecha_nac
from personas
```

legajo	Nombre Completo	fecha_nac
100	Rodríguez, Juan	1/5/1996
200	Pérez, Pedro	3/8/1984
300	Pérez, Ana	12/10/2010
400	Álvarez, Juana	3/9/1977
500	Sanchez, Juan	5/8/2002

Ilustración 4: Elaboración propia.

El alias va entre comillas salvo que no contenga caracteres especiales, y en el Transact-SQL, actualmente, no es necesaria la palabra reservada AS

```
select legajo,
       apellido + ', ' + nombre 'Nombre completo',
       fecha_nac Fecha_Nacimiento
from personas
```

Aquí la tercer columna también tiene alias que está sin comillas ya que pueden omitirse cuando el alias no tiene caracteres especiales.

- **DISTINCT|ALL.** Distinct elimina filas duplicadas en el resultado de la consulta. Por defecto está definida la cláusula ALL es decir muestra todo, incluido si existen filas con exactamente el mismo contenido.

Se puede ver ejecutando estas dos consultas y observando detenidamente la diferencia entre ambos resultados

```
select peso, altura
from personas
```

peso	altura
95	1.85
50	0.80
50	0.85
95	1.85
98	1.85

Elaboración propia

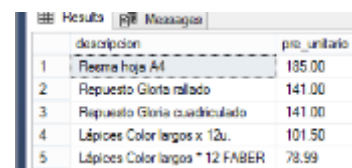
```
select distinct peso, altura
from personas
```

peso	altura
95	1.85
50	0.80
50	0.85
98	1.85

Elaboración propia

- **TOP n** Si se quiere limitar la cantidad de filas devueltas por la sentencia select, se puede utilizar combinada con Top seguido por el número de filas. Por ejemplo: se quiere listar los 5 artículos más caros, para ello emitimos el listado de los articulos ordenados por precio en forma descendente (de mayor a menor) y mostramos los 5 de más arriba:

```
select top 5 descripcion, pre_unitario
from articulos
order by 2 desc
```



	descripcion	pre_unitario
1	Pleuma hoja A4	135.00
2	Repuesto Gloria rallado	141.00
3	Repuesto Gloria cuadrado	141.00
4	Lápices Color largos x 12u.	101.50
5	Lápices Color largos * 12 FABER	79.99

Ordenamiento de los resultados de una consulta

ORDER BY: lista de columnas que debe utilizarse como criterio para ordenar los registros. La palabra reservada ASC indica que el orden ascendente es decir de mayor a menor para columnas numéricas; en order alfabético para columnas alfanuméricas según el código ASCII, y de la fecha más antigua a las más reciente si la columna es de tipo fecha. Si se indica DESC la columna se ordena en forma inversa a la anterior

Si la lista está compuesta por más de una columna, se separan por comas, y cada una con su respectivo ASC y DESC según corresponda

Se verá esta cláusula en un ejemplo: listar legajo apellido, nombre y fecha de nacimiento de personas ordenado por apellido y nombre

```
select legajo, apellido, nombre, fecha_nac
from personas
order by apellido, nombre
```

legajo	apellido	nombre	fecha_nac
400	Álvarez	Juana	3/9/1977
300	Pérez	Ana	12/10/2010
200	Pérez	Pedro	3/8/1984
100	Rodríguez	Juan	1/5/1996
500	Sanchez	Juan	5/8/2002

Ilustración 5: Elaboración propia.

Se puede observar cómo en primer lugar se ordena el listado por el apellido de las personas y en segundo lugar (cuando el apellido, el primer criterio es repetido) se ordena por el nombre.

Otro ejemplo: ordenar el listado por la segunda columna :

```
select legajo,
       apellido + ', ' + nombre 'Nombre Completo',
       fecha_nac
from personas
order by 2
```

o bien,

```
select legajo,
       apellido+', '+nombre 'Nombre Completo',
       fecha_nac fecha_nacimiento
from personas
order by apellido + ', ' + nombre
```

o bien,

```
select legajo,
       apellido + ', ' + nombre 'Nombre Completo',
       fecha_nac Fecha_nacimiento
from personas
order by 'Nombre Completo'
```


Si se quisiera ordenar por apellido en forma descendente (de la Z a la A) y en nombre en forma ascendente:

```
select legajo,
       apellido + ', ' + nombre 'Nombre Completo',
       fecha_nac Fecha_nacimiento
from   personas
order by apellido desc, nombre
```

o bien,

```
select legajo,
       apellido + ', ' + nombre 'Nombre Completo',
       fecha_nac Fecha_nacimiento
from   personas
order by 2 desc, 3
```

Condiciones de búsqueda en el WHERE

Se utiliza si solo se quiere seleccionar solo una parte de las filas de una tabla, las que cumplan con la condición especificada en esta cláusula.

Conceptualmente SQL recorre cada una de las filas de la tabla y aplica la condición. Esta fila se incluye en el resultado de la consulta si la condición es true (verdadero) de lo contrario se excluye. Se pueden resumir las condiciones de búsqueda en cinco diferentes.

Los ejemplos de aquí en más se van a basar en la base de datos librería cuyo diagrama es el que se muestra a continuación:

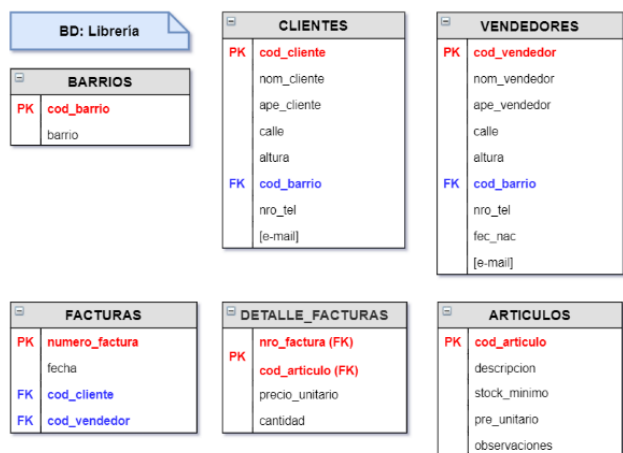


Ilustración 6: Elaboración propia

Condiciones de comparación o Test de comparación: =, <, >, !=, <=, >=, <>

SQL calcula y compara los valores de dos expresiones por cada fila de datos como puede ser un campo con una constante o expresiones más complejas. Se utilizan los operadores de comparación. En este tipo de condiciones se utilizan

operadores de comparación como el igual (=), distinto de (< > o !=), menor que (<), mayor que (>), menor o igual que (<=) y mayor o igual que (>=).

Por ejemplo, si se quiere mostrar los libros cuyo precio sea menor a \$ 150 podemos utilizar un operador de comparación en el WHERE.

Por ejemplo, listar las facturas emitidas antes del 10/7/2008

```
select *  
from facturas  
where fecha < '10/07/2008'
```

Condición de búsqueda (o Test) de correspondencia con un patrón de búsqueda

Se utiliza para comprobar si el valor de un dato en una columna coincide con un patrón específico. SQL comprueba si el valor de dato de una columna o expresión se ajusta a un patrón por cada fila. Si es así, el test devuelve verdadero y la fila se muestra.

Se utiliza el operador LIKE y el patrón es una cadena que puede incluir uno o más caracteres comodines:

- % : se utiliza para reemplazar cero o más caracteres. permiten buscar valores que contengan cierta cadena de caracteres, ya sea al inicio, al final o en cualquier posición dentro de la cadena. Por ejemplo, si se busca todos los artículos que contienen la palabra "libro", se puede usar el patrón "%libro%".
- _ (guión bajo): se utiliza para reemplazar un único carácter. Por ejemplo, si se busca todos los artículos que tienen un código de barras que comienza con "123", se puede usar el patrón "123_".
- []: se utiliza para reemplazar cualquier carácter que se encuentre dentro del rango especificado o en el conjunto especificado. Por ejemplo, si se busca todos los artículos cuyo código de barras comienza con una letra mayúscula, se puede usar el patrón "[A-Z]%".
- ^ : se utiliza para reemplazar cualquier carácter que NO se encuentre dentro del rango especificado o en el conjunto especificado. Por ejemplo, si se busca todos los artículos cuyo código de barras no comienza con una letra mayúscula, se puede usar el patrón "[^A-Z]%".

Por ejemplo: Si se quiere listar todos los artículos que comiencen con "L"

```
SELECT *  
FROM articulos  
WHERE descripcion LIKE 'L%'
```



	cod_articulo	descripcion	stock_minimo	pre_unitario	observaciones
1	1	Lápiz Evolution HB2 * 4 u	NULL	50.00	NULL
2	4	Lápices Color cortos * 12 u	54	120.50	NULL
3	6	Lápices Color largos * 12 u	NULL	127.90	NULL
4	22	Lápices con goma * 2 u	NULL	45.50	NULL
5	25	Lápices Color largos * 12 FABER	NULL	178.99	NULL
6	26	Lapicera Bic Azul trazo fino	NULL	24.99	NULL

Ilustración 10: Elaboración propia.

Algunos ejemplos utilizando los diferentes usos del comodín % en combinación con la palabra "libro":

- **Comience** con "libro": Se buscarán todos los artículos cuyo nombre comience con "libro", por ejemplo:

```
SELECT *  
FROM articulos  
WHERE nombre LIKE 'libro%'
```

Esto mostrará todos los artículos cuyo nombre empiece con "libro", como por ejemplo "Libro de matemáticas", "Libro de historia", etc.

- **Finalice** con "libro": Se buscarán todos los artículos cuyo nombre finalice con "libro", por ejemplo:

```
SELECT *  
FROM articulos  
WHERE nombre LIKE '%libro'
```

Esto mostrará todos los artículos cuyo nombre termine con "libro", como por ejemplo "Diccionario de inglés - español - libro", "El señor de los anillos - libro", etc.

- **Contenga** "libro": Se buscarán todos los artículos cuyo nombre contenga la palabra "libro" en cualquier posición, por ejemplo:

```
SELECT *  
FROM articulos  
WHERE nombre LIKE '%libro%'
```

Esto mostrará todos los artículos cuyo nombre contenga la palabra "libro" en cualquier posición, como por ejemplo "Libro de matemáticas", "Diccionario de inglés - español - libro", "El señor de los anillos - libro", etc.

Otros ejemplos:

- La cláusula where de todos los que terminen con N sería:

```
... WHERE descripcion LIKE '%N'
```

- La cláusula where de todos los que contengan lápiz sería:

```
... WHERE descripcion LIKE '%lapiz%'
```

- De los que comiencen con letras que van de la l a la m:

```
... WHERE descripcion LIKE '[l-m] %'
```

Select *

from articulos

Where descripcion like 'L%'

	cod_articulo	descripcion	stock_minimo	pre_unitario	observaciones
1	1	Lápiz Evolution HB2 * 4 u	NULL	50.00	NULL
2	4	Lápices Color cortos * 12 u	54	120.50	NULL
3	6	Lápices Color largos * 12 u	NULL	127.90	NULL
4	22	Lápices con goma * 2 u	NULL	45.50	NULL
5	25	Lápices Color largos * 12 FABER	NULL	178.99	NULL
6	26	Lapicera Bic Azul trazo fino	NULL	24.99	NULL

Elaboración propia

Condición de búsqueda (o Test) de rango

Es una forma de verificar si el valor de una columna se encuentra dentro de un rango de valores determinado por dos expresiones. Se utiliza el operador **BETWEEN** para especificar el rango. Para utilizar el operador BETWEEN se necesitan tres expresiones SQL: la primera es el valor por comprobar, y la segunda y tercera son los extremos del rango.

La sintaxis general de una consulta SQL que utiliza el operador BETWEEN es la siguiente:

```
SELECT columnas
FROM tabla
WHERE columna BETWEEN valor_inicio AND valor_fin;
```

El operador BETWEEN devuelve verdadero si el valor de la columna está dentro del rango especificado y falso en caso contrario. ***Es importante tener en cuenta que los valores de inicio y fin del rango también se incluyen en la búsqueda.***

Por ejemplo, si queremos encontrar todos los artículos cuyo precio unitario esté entre 50 y 100 pesos, se puede utilizar la siguiente consulta:

```
SELECT *
FROM articulos
WHERE pre_unitario BETWEEN 50 AND 100;
```

	cod_articulo	descripcion	stock_minimo	pre_unitario	observaciones
1	1	Lápiz Evolution HB2 * 4 u	NULL	50.00	NULL
2	7	Separadores tamaño rivadavia * 6 u	NULL	85.70	Motivos de Disney
3	11	Cuademo Tamaño rivadavia rayado	80	52.70	NULL
4	12	Cuademo Tamaño rivadavia cuadriculado	65	52.70	NULL
5	15	Conjunto Geométrico	NULL	85.90	Regla - escuadra - transportador
6	17	Cartuchera de Tela	55	62.60	NULL
7	18	Correctores Bic Lápiz * 1 u	NULL	71.40	NULL
8	19	Rótulos * 18 u	NULL	55.90	NULL
9	23	Goma para Lápiz * 10 u	NULL	87.50	NULL
10	24	Goma para lapicera * 10 u	NULL	99.50	NULL
11	27	Cuademo tapa dura rayado	20	92.99	NULL

Elaboración propia.

Esta consulta devolverá todos los artículos cuyo precio unitario esté entre 50 y 100 pesos, incluyendo los valores 50 y 100.

Condición de búsqueda (o Test) de pertenencia a un conjunto

Es una cláusula que se utiliza en SQL para verificar si el valor de una columna se encuentra dentro de un conjunto de valores especificados. El operador utilizado para realizar esta verificación es el operador **IN**.

En la consulta SQL, después de la cláusula WHERE, se especifica la columna que se desea verificar y se utiliza el operador IN, seguido por una lista de valores separados por comas dentro de paréntesis. La consulta devolverá todas las filas donde el valor de la columna especificada esté presente en la lista de valores.

Un ejemplo de uso de la Condición de búsqueda de pertenencia a un conjunto sería listar todos los clientes cuyo código sea 1, 3, 7, 8 o 12. La consulta SQL para esto sería:

```
SELECT *
FROM clientes
```


WHERE cod_cliente **IN** (1, 3, 7, 8, 12)

	cod_cliente	nom_cliente	ape_cliente	calle	altura	cod_barrio	nro_tel	e-mail
1	1	Rodolfo	Perez	San Martin	120	1	NULL	NULL
2	3	Héctor	Abarca	Luis Gongora	160	12	4701314	habarca@hotmail.com
3	7	Roque	Paez	Humberto Primo	79	1	4262630	NULL
4	8	Elvira Josefa	Luque	Mariano Usandivaras	360	3	4502829	NULL
5	12	Adriana	Gonzalez	San Jerónimo	763	1	NULL	NULL

Ilustración 7: Elaboración propia.

Esta condición es muy útil para simplificar la escritura de consultas, ya que permite verificar si una columna se encuentra dentro de un conjunto de valores sin necesidad de escribir múltiples condiciones utilizando el **operador OR**.

Condición de búsqueda (o Test) de valor nulo

Se utiliza para verificar si un valor en una columna de una tabla de base de datos es nulo o no. El valor nulo representa la ausencia de un valor o un valor desconocido en una columna determinada.

El operador **"IS NULL"** se utiliza para comparar el valor de una columna con el valor nulo. Si la columna tiene un valor nulo, la expresión será verdadera y el registro se incluirá en los resultados de la consulta. Si la columna tiene un valor distinto de nulo, la expresión será falsa y el registro no se incluirá en los resultados.

Por ejemplo, listar los artículos para los que no existan observaciones, La sintaxis utilizada en esta consulta es la siguiente:

```
SELECT *
FROM articulos
WHERE observaciones IS NULL
```

En la consulta de ejemplo, se utiliza la condición de búsqueda de valor nulo para seleccionar todos los artículos de una tabla de base de datos en la que la columna "observaciones" no tiene ningún valor (es nula). Esto significa que la consulta devolverá todos los registros que no tienen observaciones asociadas, es decir, que la columna "observaciones" no tiene ningún dato almacenado en la tabla.

La condición "observaciones IS NULL" se utiliza para seleccionar todos los registros en la tabla "articulos" donde la columna "observaciones" es nula. El operador IS se utiliza para comparar el valor de la columna "observaciones" con el

valor nulo. Si la columna tiene un valor nulo, la condición devuelve TRUE y el registro se selecciona en la consulta.

Es importante tener en cuenta que el operador IS NULL solo puede utilizarse para comparar un valor de columna con el valor nulo. Si se intenta comparar un valor con una cadena vacía (""), se debe utilizar la condición "observaciones = ''" en lugar de "observaciones IS NULL".

Operadores lógicos NOT, OR y AND

Operador NOT

Se puede utilizar para seleccionar filas en donde la condición de búsqueda es falsa. Por ejemplo listar los artículos cuyo precio no esté entre 10 y 100

```
Select *
from articulos
Where pre_unitario not between 10 and 100
```

	cod_articulo	descripcion	stock_minimo	pre_unitario	observaciones
1	2	Papel p/forrar Fantasia * 10 u	130	220.00	NULL
2	3	Papel p/forrar Araña * 10 u	150	250.00	Color rojo - azul - verde
3	4	Lápices Color cortos * 12 u	54	120.50	NULL
4	5	Fibras cortas * 6	40	125.30	NULL
5	6	Lápices Color largos * 12 u	NULL	127.90	NULL
6	8	Carpeta fibra negra - Tamaño rivadavia	NULL	102.90	3 anillos
7	9	Carpeta Fantasia - Tamaño Rivadavia	60	150.90	3 anillos

Ilustración 8: Elaboración propia

Condiciones de búsqueda compuestas

Utilizando las reglas de la lógica, se pueden combinar estas condiciones de búsqueda simples para formar otras más complejas utilizando los operadores lógicos OR y AND

El operador **NOT** es un operador lógico en SQL que se utiliza para negar una condición de búsqueda. En otras palabras, el operador NOT se utiliza para seleccionar las filas que no cumplen una determinada condición.

En un ejemplo de uso del operador NOT en SQL, se puede utilizar para seleccionar todos los artículos de una tabla de base de datos en los que el precio unitario no se encuentra entre 60 y 600. La consulta correspondiente tendría la siguiente estructura:

```
SELECT *
FROM articulos
WHERE pre_unitario NOT BETWEEN 60 AND 600
```


	cod_articulo	descripcion	stock_minimo	stock	pre_unitario	observaciones
1	10	Adhesivo sintético 30 gr	NULL	20	56.00	NULL
2	11	Cuaderno Tamaño rivadavia rayado	150	180	52.70	NULL
3	12	Cuaderno Tamaño rivadavia cuadriculado	50	65	52.70	NULL
4	13	Repuesto Gloria rallado	150	120	641.00	400 hojas
5	14	Repuesto Gloria cuadriculado	20	90	641.00	400 hojas
6	26	Lapicera Bic Azul trazo fino	NULL	100	54.99	NULL

Ilustración 9: Elaboración propia.

La condición "pre_unitario NOT BETWEEN 60 AND 600" se utiliza para seleccionar todos los registros en la tabla "articulos" en los que el precio unitario no se encuentra entre 60 y 600. Si un registro cumple la condición, se seleccionará en la consulta.

Es importante destacar que la cláusula "**NOT BETWEEN**" también se puede utilizar para negar otros operadores de comparación, como "**NOT IN**" o "**NOT LIKE**".

Condiciones de búsqueda compuestas

Las condiciones de búsqueda compuestas son expresiones lógicas que se construyen a partir de condiciones de búsqueda simples mediante el uso de los operadores lógicos OR y AND. Estos operadores permiten combinar múltiples condiciones de búsqueda y establecer relaciones lógicas entre ellas para obtener resultados más precisos y específicos.

Operador AND

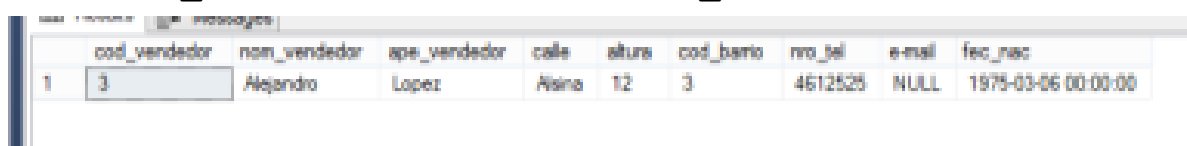
La expresión lógica AND se utiliza para combinar dos o más condiciones de búsqueda y **devuelve únicamente aquellos registros que cumplen todas las condiciones**.

Como es el caso del listado de vendedores cuyo nombre comience con A y nacidos antes de 1980:

Select *

from vendedores

Where nom_vendedor like 'A%' **and** fec_nac < '1/1/1980'



	cod_vendedor	nom_vendedor	ape_vendedor	calle	altura	cod_banio	nro_tel	e-mail	fec_nac
1	3	Alejandro	Lopez	Alina	12	3	4612525	NULL	1975-03-06 00:00:00

Ilustración 10: Elaboración propia

En este caso solo se muestra el registro que cumple con ambas condiciones

La siguiente consulta SQL devuelve todas las filas de la tabla 'ventas' donde el valor de la columna 'monto' sea mayor que 500 y la columna 'fecha' sea posterior a '01/01/2022':

```
SELECT *
FROM ventas
WHERE monto > 500 AND fecha > '01/01/2022'
```

Operador OR

La expresión lógica OR se utiliza para combinar dos o más condiciones de búsqueda y **devuelve aquellos registros que cumplan al menos una de las condiciones**.

```
Select *
from articulos
Where stock_minimo > 100 or pre_unitario > 80
```



	cod_articulo	descripcion	stock_minimo	pre_unitario	observaciones
1	2	Papel p-fonar FontaTe * 10 u	130	22.30	NULL
2	3	Papel p-fonar AnaTe * 10 u	150	25.50	Color rojo - azul - verde
3	13	Repuesto Glorie rollado	120	141.00	400 hojas
4	14	Repuesto Glorie cuadriculado	90	141.00	400 hojas
5	20	Resma hoja A4	40	185.00	NULL
6	28	Lápices Color largos x 12u	NULL	101.50	NULL

Ilustración 11: Elaboración propia

En el ejemplo se listarán los registros cuyo stock mínimo sea mayor a 100 o cuyo precio mayor a 80. Preste atención al 1er. registro, cumple con la condición del stock mínimo mayor a 100 pero no cumple con la del precio unitario mayor a 80; el registro 3 cumple con ambas condiciones y el registro cuatro solo cumple la condición del precio unitario.

La siguiente consulta SQL devuelve todas las filas de la tabla 'clientes' donde el valor de la columna 'nombre' sea 'María' o 'Juan':

```
SELECT *
FROM clientes
WHERE nombre = 'Maria' OR nombre = 'Juan'
```

Además, se pueden utilizar paréntesis para establecer el orden de las operaciones y crear expresiones lógicas más complejas. Por ejemplo, la siguiente consulta SQL devuelve todas las filas de la tabla 'productos' donde el valor de la columna 'precio' sea mayor que 50 y la columna 'stock' sea mayor que 10 o el valor

de la columna 'precio' sea menor o igual que 50 y la columna 'stock' sea mayor o igual que 20:

```
SELECT *  
FROM productos  
WHERE (precio > 50 AND stock > 10)  
OR (precio <= 50 AND stock >= 20)
```

En este caso, la consulta busca todos los registros de la tabla 'productos' que cumplan alguna de las dos siguientes condiciones:

1. El precio es mayor que 50 y el stock es mayor que 10.
2. El precio es menor o igual que 50 y el stock es mayor o igual que 20.

Para ello, se utiliza la expresión lógica OR para combinar ambas condiciones. Es decir, la consulta devolverá todos los registros que cumplan la primera condición o la segunda condición o ambas.

Además, para cada condición se utilizan operadores lógicos AND para combinar dos condiciones de búsqueda simples. En la primera condición, la columna 'precio' debe ser mayor que 50 y la columna 'stock' debe ser mayor que 10. En la segunda condición, la columna 'precio' debe ser menor o igual que 50 y la columna 'stock' debe ser mayor o igual que 20.

Combinación de Tablas

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina componer (joining) las tablas. La tabla resultante (que contiene datos de las dos tablas originales) se denomina una composición entre las dos tablas. Una composición basada en una coincidencia exacta entre dos columnas se denomina más precisamente una equicomposición.

Para realizar en SQL composiciones multitabla se puede utilizar la sentencia SELECT con una condición de búsqueda que especifique la comparación de columnas de tablas diferentes; previamente la cláusula FROM lista las dos tablas intervinientes separadas con comas. El resultado obtenido es una composición "Interna", es decir que se mostrarán los registros de la primera tabla que tengan registros relacionados en la segunda tabla y los registros de la segunda que tengan registros relacionados con la primera, si existe algún registro de una tabla que no tiene relación con la otra no se mostrará.

Por ejemplo, se quiere listar los vendedores y el barrio donde viven.

Listando los vendedores y los barrios por separado se puede observar que el vender código 1 vive en el barrio con código 2, y el vendedor 2 vive en el barrio 5

cod_vendedor	nombre	cod_barrio
1	Camizo Mart?n	2
2	Ledesma Mariela	5
3	Lopez Alejandro	3
4	Miranda Marcelo	1
5	Monti Gabriel	4
6	Juarez Susana	9

cod_barrio	barrio
1	CENTRO
2	ALTO ALBERDI
3	OBSERVATORIO
4	JARDÍN
5	GENERAL PAZ
6	PUEYRREDÓN
7	PARQUE HORIZONTE
8	SAN MARTÍN
9	SAN VICENTE
10	NUEVA CÓRDOBA
11	MAIPÚ
12	PANAMERICANO

Ilustración 12: Elaboración propia

Entonces, si se realiza la composición de las tablas a través del campo que tienen en común cod_barrio, la sentencia SELECT, sería la siguiente:

```
select ape_vendedor+' '+nom_vendedor Vendedor, barrio
from vendedores, barrios
where vendedores.cod_barrio=barrios.cod_barrio
```

Ilustración 13: Elaboración propia

Genera resultados solo para los pares de filas en los que el número de barrio (cod_barrio) de la tabla barrios coincide con el número de barrio de la tabla vendedores.

Entre la tabla barrios y vendedores existe una relación de uno a varios entonces se ha especificado en la condición de búsqueda que compare la clave foránea y la clave primaria de ambas tablas.

Results		Messages
	Vendedor	barrio
1	Camizo Martín	ALTO ALBERDI
2	Ledesma Mariela	GENERAL PAZ
3	Lopez Alejandro	OBSERVATORIO
4	Miranda Marcelo	CENTRO
5	Monti Gabriel	JARDIN
6	Juarez Susana	SAN VICENTE
7	Ortega Ana	AL TO ALBERDI

Ilustración 14: Elaboración propia

Hay que tener en cuenta que si un campo tiene el mismo nombre en dos tablas diferentes se debe anteponer al mismo el nombre de la tabla a la cual pertenece separado por un punto.

Es importante tener en cuenta que, si un campo tiene el mismo nombre en dos tablas diferentes, se debe anteponer el nombre de la tabla a la que pertenece, separado por un punto. Para evitar escribir repetidamente el nombre de la tabla, se puede dar un alias a las tablas en la cláusula FROM.

Alias en la cláusula FROM: se refiere a dar un nombre corto o abreviado a una tabla en una consulta SQL, para facilitar la lectura y escritura del código. Esto permite hacer referencia a ellas de manera más sencilla y clara en el resto de la consulta, especialmente cuando se está trabajando con varias tablas y/o con nombres de tablas largos.

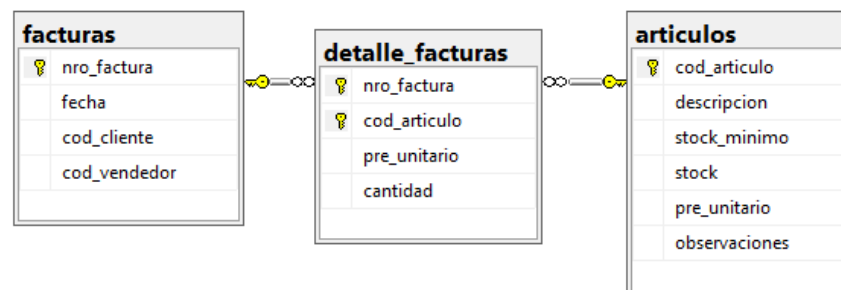
En el ejemplo proporcionado, se utilizan los alias "v" y "b" para las tablas "vendedores" y "barrios", respectivamente. Estos alias se definen en la cláusula FROM mediante la sintaxis "nombre_de_tabla alias", y luego se utilizan en la cláusula WHERE para especificar la relación entre las tablas mediante la comparación de las claves primarias y foráneas.

```
SELECT v.ape_vendedor + ', ' + v.nom_vendedor 'Vendedor',  
       b.barrio  
FROM   vendedores v, barrios b  
WHERE  v.cod_barrio = b.cod_barrio
```

Ilustración 155: Elaboración propia.

Es importante destacar que los alias deben ser únicos dentro de una consulta y que su uso puede mejorar significativamente la legibilidad y la eficiencia de la misma. Además, pueden ser especialmente útiles cuando se combinan varias tablas o cuando se trabaja con nombres de tabla largos o complejos.

En SQL, es posible combinar tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas. A través de la utilización de esta técnica, se pueden crear consultas que relacionen múltiples tablas y proporcionen información más detallada y precisa. Un ejemplo de esto es la siguiente consulta, que lista los artículos facturados con sus respectivas facturas.



```
SELECT f.nro_factura, fecha, descripcion, cantidad,  
       d.pre_unitario, cantidad * d.pre_unitario 'Subtotal'  
FROM   facturas f, detalle_facturas d, articulos a  
WHERE  f.nro_factura = d.nro_factura  
AND    a.cod_articulo = d.cod_articulo
```

Ilustración 166: Elaboración propia.

SQL puede combinar tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas. La siguiente consulta lista los artículos facturados con sus respectivas facturas

```
Select f.nro_factura, fecha, descripcion, cantidad, d.pre_unitario,
       cantidad*d.pre_unitario Subtotal
from facturas f, detalle_facturas d, articulos a
where f.nro_factura = d.nro_factura
and a.cod_articulo = d.cod_articulo
```

Ilustración 17: Elaboración propia

Composiciones con criterios de selección de fila.

Las composiciones con criterios de selección de fila son una técnica muy útil en consultas de múltiples tablas, que permiten combinar la especificación de las columnas de emparejamiento con otras condiciones de búsqueda para obtener resultados más precisos y relevantes.

Por ejemplo, supongamos que deseamos *'listar todas las facturas del mes de agosto de 2019 con sus respectivos clientes'*. En este caso, podemos utilizar una composición entre las tablas de "facturas" y "clientes", utilizando como columna de emparejamiento la clave primaria y foránea que relaciona ambas tablas. Luego, podemos **agregar una condición adicional** en la cláusula WHERE que especifique que solo se deben incluir las facturas emitidas en el mes de agosto de 2019.

La consulta SQL resultante podría ser la siguiente:

```
SELECT f.nro_factura, f.fecha, c.ape_cliente + ', ' + c.nom_cliente 'Clientes'
FROM facturas f , clientes c
WHERE f.cod_cliente = c.cod_cliente
AND MONTH(f.fecha) = 8 AND YEAR(f.fecha) = 2019
```

Ilustración 187: Elaboración propia.

En esta consulta, se están combinando dos tablas: "facturas" y "clientes". La condición de búsqueda utilizada en la cláusula WHERE establece que solo se deben seleccionar aquellos registros donde el mes y el año de la fecha de emisión en la tabla "facturas" sean iguales a mayo y 2015, respectivamente. De esta manera, la consulta devuelve únicamente las facturas del mes de mayo de 2015 junto con los nombres de sus respectivos clientes. En esta consulta se utilizan las funciones MONTH() y YEAR() de SQL para extraer el mes y el año de una fecha determinada. Estas funciones se trabajarán en detalle más adelante.

	nro_factura	fecha	Cientes
1	192	2019-08-04 00:00:00.000	Castillo, Marta Analía
2	193	2019-08-03 00:00:00.000	Morales, Santiago
3	194	2019-08-01 00:00:00.000	Paez, Roque
4	195	2019-08-02 00:00:00.000	Morales, Santiago

Ilustración 19: Elaboración propia.

Multiplicación de tablas.

Una composición es un caso especial de una combinación más general de datos procedentes de dos tablas, conocida como el producto cartesiano de dos tablas. El producto de dos tablas es otra tabla que consta de todos los pares posibles de filas de las dos tablas. Las columnas de la tabla producto son todas las columnas de la primera tabla, seguidas de todas las columnas de la segunda tabla. Si se especifica una consulta de dos tablas sin una cláusula WHERE, SQL produce el producto de las dos tablas como resultado. El siguiente ejemplo muestra todas las posibles de vendedores y barrios.

El producto cartesiano, también conocido como multiplicación de tablas, es una operación que se realiza en SQL al combinar dos tablas (o varias tablas) **sin una condición de unión**. Al realizar esta operación, se creará una nueva tabla que **contiene todas las combinaciones posibles entre las filas** de las dos tablas (o varias tablas).

Por ejemplo, si se tienen dos tablas: "vendedores" y "barrios", ambas con dos filas, el producto cartesiano resultante tendría 4 filas, cada una representando una combinación de vendedor y barrio:

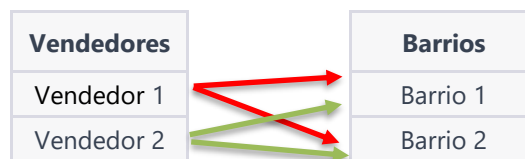


Ilustración 20: Elaboración propia.

Tabla resultante con el producto cartesiano:

Vendedores	Barrios
Vendedor 1	Barrio 1
Vendedor 1	Barrio 2
Vendedor 2	Barrio 1
Vendedor 2	Barrio 2

Ilustración 21: Elaboración propia.

Es importante tener en cuenta que el producto cartesiano puede ser útil en algunas situaciones, pero generalmente no se desea obtener todas las combinaciones posibles entre las filas de dos tablas. Por eso, es común utilizar una cláusula WHERE para establecer una condición y obtener resultados más precisos

Composición de tablas en el FROM con JOIN

Un **JOIN** es una operación que se utiliza para combinar dos o más tablas en una sola tabla, de manera que podamos obtener una vista integrada de los datos contenidos en ellas.

Cuando combinamos tablas usando un Join, se utiliza uno o varios campos en común que sirven como clave para relacionar los registros de las tablas.

Hay tres tipos de combinaciones:

- Combinaciones internas (**INNER JOIN** o **JOIN**)
- Combinaciones externas: Existen dos tipos: **LEFT JOIN** y **RIGHT JOIN**.
- Combinaciones cruzadas (producto cartesiano)

Composición interna Inner Join

Inner Join es una operación que se utiliza para combinar dos o más tablas en una sola tabla, seleccionando únicamente las filas que tienen valores coincidentes en ambas tablas.

La combinación interna Inner Join se utiliza para obtener información de dos o más tablas y combinar dicha información en una salida. Esta operación se realiza mediante la utilización de la cláusula "**JOIN**" en la consulta SQL. La sintaxis básica de una consulta Inner Join es la siguiente:

```
SELECT campos
FROM tabla1
JOIN tabla2 ON condición de combinación;
```

En esta consulta, "tabla1" y "tabla2" son los nombres de las dos tablas que se desean combinar, y "campos" son los campos de la tabla que se desean mostrar en la salida. La condición de combinación se especifica mediante la cláusula "**ON**", y determina cómo se relacionan las filas de las dos tablas.

Por ejemplo, si se desea listar los datos de los clientes con sus facturas (cada factura con su respectivo cliente), se puede utilizar una consulta Inner Join de la siguiente manera:

```
SELECT ape_cliente + ', ' + nom_cliente 'CLIENTE',
       nro_factura 'FACTURA', fecha 'FECHA'
FROM facturas f
JOIN clientes c ON f.cod_cliente = c.cod_cliente;
```

En esta consulta, "facturas" y "clientes" son las dos tablas que se desean combinar, y se relacionan mediante el campo "cod_cliente". La salida de esta consulta mostrará los datos de los clientes junto con sus facturas correspondientes.

Si se desea realizar una consulta con más de dos tablas, como listar los datos de los clientes con sus facturas y vendedores, se pueden utilizar varias cláusulas Join en la consulta. Por ejemplo:

```
SELECT ape_cliente + ', ' + nom_cliente 'CLIENTE',
       nro_factura 'FACTURA',
       fecha 'FECHA',
       ape_vendedor + ', ' + nom_vendedor 'VENDEDOR'
FROM facturas f
JOIN clientes c ON f.cod_cliente = c.cod_cliente
JOIN vendedores v ON f.cod_vendedor = v.cod_vendedor;
```

En esta consulta, se utilizan tres tablas ("facturas", "clientes" y "vendedores"), y se relacionan mediante los campos "cod_cliente" y "cod_vendedor". La salida de esta consulta mostrará los datos de los clientes junto con sus facturas y vendedores correspondientes. Alguno de los datos resultantes de la consulta:

	CLIENTE	FACTURA	FECHA	VENDEDOR
527	Morales, Pilar	527	2022-05-14 00:00:00.000	Monti, Gabriel
528	Luque, Elvira Josefa	528	2022-05-15 00:00:00.000	Lopez, Alejandro
529	Ruiz, Marcos	529	2022-05-17 00:00:00.000	Monti, Gabriel
530	Perez, Rodolfo	530	2022-05-17 00:00:00.000	Lopez, Alejandro
531	Ruiz, Marcos	531	2022-05-18 00:00:00.000	Ledesma, Mariela
532	Paéz, Roque	532	2022-05-19 00:00:00.000	Miranda, Marcelo
533	Castillo, Marta Analía	533	2022-05-20 00:00:00.000	Monti, Gabriel
534	Ruiz, Marcos	534	2022-05-20 00:00:00.000	Miranda, Marcelo
535	Luque, Elvira Josefa	535	2022-05-21 00:00:00.000	Carrizo, Martín
536	Perez, Rodolfo	536	2022-05-22 00:00:00.000	Ledesma, Mariela
537	Abarca, Héctor	537	2022-05-23 00:00:00.000	Ledesma, Mariela
538	Moriel, Roberto	538	2022-05-23 00:00:00.000	Lopez, Alejandro

Ilustración 22: Elaboración propia

Usar **"JOIN"** o **"INNER JOIN"** en una consulta SQL da el mismo resultado que realizar la composición como una condición de búsqueda en el **"WHERE"**. Ambas opciones muestran únicamente los registros donde coinciden los valores de las claves primarias y foráneas de las tablas involucradas en la combinación.

Esto significa que, si una tabla no tiene una correspondencia en la otra tabla, no se mostrarán los registros de esa tabla en la salida. Por ejemplo, si se realiza una consulta para listar los datos de los clientes con sus facturas utilizando Inner Join, no se mostrarán los registros de aquellos clientes que no tengan facturas, ni las facturas que no tengan un cliente asociado. Lo mismo sucede si se desea listar los datos de los vendedores con sus facturas utilizando Inner Join, donde no se mostrarán los registros de aquellos vendedores que no tengan facturas registradas.

En resumen, Inner Join se utiliza para combinar dos o más tablas en una sola tabla y mostrar únicamente los registros que tienen valores coincidentes en ambas tablas. Si una tabla no tiene una correspondencia en la otra tabla, no se mostrarán los registros de esa tabla en la salida.

Combinación externa izquierda: **LEFT JOIN**

La combinación externa izquierda, también conocida como **LEFT JOIN**, es una operación de unión de tablas en la que se seleccionan todos los registros de la tabla de la izquierda (primera tabla) y aquellos registros de la tabla de la derecha (segunda tabla) que coincidan con los registros de la tabla de la izquierda en función de la condición de unión especificada. Si un registro de la tabla de la izquierda no tiene una coincidencia en la tabla de la derecha, se muestra en los resultados de la consulta con los campos de la tabla de la derecha establecidos en valores nulos **"NULL"**. De esta manera, se pueden incluir registros que no tengan una correspondencia exacta en ambas tablas.

La sintaxis básica para un **LEFT JOIN** es la siguiente:

```
SELECT campos
FROM tabla_izquierda
LEFT JOIN tabla_derecha ON condicion_de_union;
```

Por ejemplo, si tenemos dos tablas llamadas "Clientes" y "Pedidos", donde "Clientes" tiene información sobre los clientes y "Pedidos" almacena información sobre los pedidos realizados por los clientes, podemos usar un LEFT JOIN para mostrar todos los clientes, independientemente de si han realizado algún pedido o no, junto con sus pedidos correspondientes:

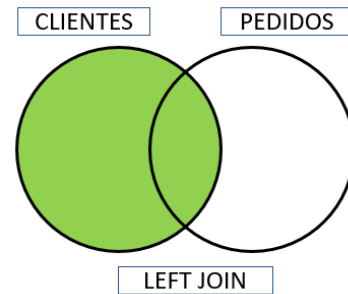


Tabla "Clientes":

id_cliente	nombre	apellido
1	Juan	Pérez
2	María	González
3	Carlos	Sánchez
4	Ana	López
5	Francisco	Martínez

Tabla "Pedidos":

nro_pedido	id_cliente	fecha
1	1	01/01/2022
2	2	02/01/2022
3	1	03/01/2022
4	4	04/01/2022
5	3	05/01/2022

Tabla 1: Elaboración propia.

Con estas tablas, podemos usar un LEFT JOIN para mostrar todos los clientes y sus pedidos correspondientes, incluso si no han realizado ningún pedido:

```
SELECT c.nombre, p.nro_pedido, p.fecha
FROM clientes c
LEFT JOIN pedidos p ON c.id_cliente = p.id_cliente;
```

El resultado de esta consulta sería:

nombre	nro_pedido	fecha
Juan	1	01/01/2022
Juan	3	03/01/2022
María	2	02/01/2022
Carlos	NULL	NULL
Ana	4	04/01/2022
Francisco	NULL	NULL

Tabla 1: Elaboración propia.

Como se puede ver, en la columna de "nro_pedido" y "fecha" aparecen valores nulos "NULL" para aquellos clientes que no han realizado ningún pedido.

Combinación externa derecha: RIGHT JOIN

La combinación externa derecha, también conocida como **RIGHT JOIN** es una operación de unión de tablas en la que se seleccionan todos los registros de la tabla de la derecha (segunda tabla) y aquellos registros de la tabla de la izquierda (primera tabla) que coincidan con los registros de la tabla de la derecha en función de la condición de unión especificada. Si un registro de la tabla de la derecha no tiene una coincidencia en la tabla de la izquierda, se muestra en los resultados de la consulta con los campos de la tabla de la izquierda establecidos en valores nulos "**NULL**". De esta manera, se pueden incluir registros que no tengan una correspondencia exacta en ambas tablas.

En SQL, la sintaxis básica para un RIGHT JOIN es:

```
SELECT campos
FROM tabla_izquierda
RIGHT JOIN tabla_derecha ON condicion_de_union;
```

Por ejemplo, tenemos dos tablas en una base de datos: "Empleados" y "Departamentos". La tabla "Empleados" contiene información sobre los empleados de una empresa, incluyendo su número de identificación (id_emp), apellido y nombre. La tabla "Departamentos" contiene información sobre los departamentos de la empresa, incluyendo su número de identificación (id_depto), la identificación del empleado a cargo (id_emp) y la fecha de alta del departamento (fecha_alta).

Queremos realizar una consulta que muestre todos los departamentos de la empresa, junto con el nombre y apellido de su jefe (el empleado a cargo), aunque algunos departamentos no tengan un jefe asignado todavía. Para lograr esto, utilizaremos un RIGHT JOIN entre las dos tablas.

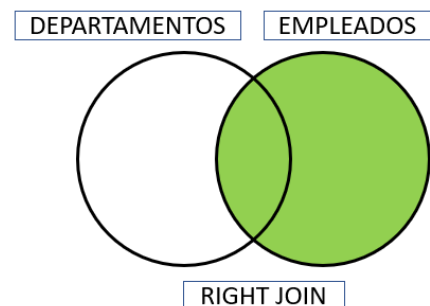


Tabla "Empleados":

id_emp	apellido	nombre
1	García	Juan
2	Rodríguez	Ana
3	Pérez	Luis
4	Sánchez	Mariana
5	González	Santiago

Tabla "Departamentos":

id_depto	id_emp	fecha_alta
1	1	01/01/2021
1	2	15/02/2021
2	3	10/03/2021
2	4	15/03/2021

Tabla 1: Elaboración propia.

Si queremos mostrar todos los empleados, incluyendo aquellos que no tienen un departamento asignado, podemos utilizar un RIGHT JOIN de la siguiente manera:

```
SELECT e.id_emp, e.apellido, d.id_depto, d.fecha_alta  
FROM departamentos d  
RIGHT JOIN empleados e ON d.id_emp = e.id_emp;
```

Este código mostrará una tabla con todos los empleados, incluyendo aquellos que no tienen un departamento asignado. En esos casos, el valor de "id_depto" y "fecha_alta" serán nulos (NULL). El resultado de esta consulta sería:

id_emp	apellido	id_depto	fecha_alta
1	García	1	01/01/2021
2	Rodríguez	1	15/02/2021
3	Pérez	2	10/03/2021
4	Sánchez	2	15/03/2021
5	González	NULL	NULL

Tabla 1: Elaboración propia.

Como se puede observar, el empleado con "id_emp" 5 no tiene un departamento asignado, por lo que los valores de "id_depto" y "fecha_alta" son nulos (NULL).

Combinación externa completa: FULL JOIN

La combinación externa completa o "**FULL JOIN**" es una operación de unión en bases de datos que se utiliza para unir dos tablas y mostrar todos los registros de ambas tablas, incluso aquellos que no tienen coincidencias en la otra tabla. Esto significa que si hay registros en una tabla que no tienen una coincidencia en la otra tabla, se mostrarán en los resultados de la consulta con los campos correspondientes a la otra tabla establecidos como "**NULL**".

La estructura de una combinación externa completa o "full join" en SQL es la siguiente:

```
SELECT *  
FROM tabla1  
FULL OUTER JOIN tabla2 ON tabla1.campo = tabla2.campo;
```

Donde "tabla1" y "tabla2" son los nombres de las tablas que se quieren unir y "campo" es el campo común en ambas tablas por el que se quiere unir.

Veamos un ejemplo, se desea obtener una lista completa de todos los empleados y sus respectivos salarios. Incluso si un empleado no tiene un registro de salario en la tabla de salarios, se debe mostrar el registro de empleado con un valor "null" en el campo de salario. De manera similar, si hay un registro de salario sin un correspondiente registro de empleado en la tabla de empleados, se debe mostrar el registro de salario con un valor "null" en el campo de empleado.

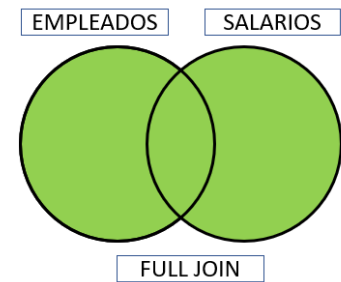


Tabla "Empleados":

id_empleado	nombre	departamento
1	Juan	Ventas
2	Ana	Finanzas
3	Pedro	Marketing
4	Catalina	Recursos Humanos

Tabla "Salarios":

id_salario	id_empleado	monto
1	1	4000
2	2	5000
3	3	4500
4	5	3500

Tabla 1: Elaboración propia.

Ahora, si queremos hacer una "full join" entre las tablas "empleados" y "salarios":

```
SELECT *
FROM empleados e
FULL OUTER JOIN salarios s ON e.id_empleado = s.id_empleado;
```

El resultado sería:

id_empleado	nombre	departamento	id_salario	monto
1	Juan	Ventas	1	4000
2	Ana	Finanzas	2	5000
3	Pedro	Marketing	3	4500
4	Catalina	Recursos Humanos	NULL	NULL
NULL	NULL	NULL	4	3500

Tabla 1: Elaboración propia.

Como se puede ver, la consulta muestra todos los registros de ambas tablas, incluso aquellos que no tienen una coincidencia en la otra tabla, y los campos correspondientes son "NULL".

Es importante tener en cuenta que la "full join" no es la operación más eficiente y, en algunos casos, puede producir una cantidad excesiva de resultados si se tienen tablas con una gran cantidad de registros. Por lo tanto, es importante evaluar cuidadosamente si es necesaria una "full join" o si se puede utilizar otra operación de unión más eficiente en su lugar.

Combinaciones cruzadas: CROSS JOIN

Correcto, una combinación cruzada (**CROSS JOIN**) en SQL es una operación que muestra todas las combinaciones de todos los registros de las tablas combinadas, sin incluir una condición de enlace (JOIN condition). El resultado es el producto cartesiano de todas las filas de ambas tablas, lo que significa que cada fila de la primera tabla se combina con todas las filas de la segunda tabla, y viceversa.

Por lo tanto, el número de filas del resultado es igual al número de registros de la primera tabla multiplicado por el número de registros de la segunda tabla. Si una tabla tiene m registros y la otra tiene n registros, el resultado de la combinación cruzada tendrá $m * n$ filas.

Es importante tener en cuenta que la combinación cruzada puede generar un gran número de filas en el resultado, lo que puede ser muy costoso en términos de rendimiento. Por lo tanto, es recomendable utilizarla con precaución y solo cuando sea necesario, en tablas pequeñas o cuando se necesite generar un conjunto completo de combinaciones para un propósito específico.

Por ejemplo, Supongamos que tenemos dos tablas, "tabla1" y "tabla2", con los siguientes datos:

Tabla 1:		Tabla 2:	
id	nombre	id	ciudad
1	Juan	1	Madrid
2	Ana	2	Barcelona
		3	Valencia

Tabla 1: Elaboración propia.

Para obtener todas las combinaciones de registros de ambas tablas, podemos realizar una combinación cruzada de la siguiente manera:

```
SELECT * FROM tabla1 CROSS JOIN tabla2;
```

El resultado de esta consulta sería:

tabla1.id	tabla1.nombre	tabla2.id	tabla2.ciudad
1	Juan	1	Madrid
1	Juan	2	Barcelona
1	Juan	3	Valencia
2	Ana	1	Madrid
2	Ana	2	Barcelona
2	Ana	3	Valencia

Tabla 1: Elaboración propia.

Como se puede observar, el resultado incluye todas las combinaciones de registros de ambas tablas, sin importar si hay o no una relación lógica entre ellas.

Combinar varios tipos de join en una misma sentencia.

es posible combinar varios tipos de join en una misma sentencia SQL para obtener información de varias tablas relacionadas. Sin embargo, es importante tener en cuenta el orden de las tablas y los tipos de join que se utilizan, ya que esto afectará el resultado final de la consulta.

Cuando se utilizan varios joins en una misma sentencia, la tabla resultante del primer join se combina con la siguiente tabla, y así sucesivamente, hasta que se hayan combinado todas las tablas especificadas en la sentencia.

Por ejemplo, supongamos que tenemos tres tablas relacionadas: "ventas", "productos" y "proveedores". La tabla "ventas" tiene una clave foránea (FK) "producto_id" que se relaciona con la tabla "productos", y la tabla "productos" tiene una clave foránea "proveedor_id" que se relaciona con la tabla "proveedores".

Si queremos obtener información sobre las ventas, los productos y los proveedores relacionados, podemos utilizar varios tipos de join en una misma sentencia SQL. Por ejemplo:

```
SELECT ven.*, prod.*, prov.*
```

```
FROM ventas ven
```

```
INNER JOIN productos prod ON ven.producto_id = prod.id
```

```
LEFT JOIN proveedores prov ON prod.proveedor_id = prov.id;
```

En este ejemplo, se utiliza un inner join para unir las tablas "ventas" y "productos" por la clave foránea "producto_id". Luego, se utiliza un left join para unir la tabla "productos" con la tabla "proveedores" por la clave foránea "proveedor_id".

Es importante recordar que el orden de las tablas y los tipos de join utilizados pueden afectar el resultado de la consulta. Por lo tanto, es recomendable entender bien las relaciones entre las tablas y planificar cuidadosamente la sentencia SQL antes de ejecutarla.

Funciones Incorporadas

Funciones para el manejo de cadenas

Las funciones para el manejo de cadenas en SQL Server son un conjunto de herramientas integradas que permiten trabajar con cadenas de caracteres de manera eficiente. Estas funciones facilitan la manipulación, extracción, búsqueda y comparación de cadenas, lo que resulta esencial para diversas tareas relacionadas con bases de datos. A continuación, se presentan algunas de las funciones más comunes utilizadas en SQL Server para el manejo de cadenas.

- **SUBSTRING (cadena, inicio, longitud):** se utiliza para obtener una parte específica de una cadena de caracteres. Se especifica la cadena de entrada, la posición de inicio y la longitud de la subcadena deseada. La función devuelve la subcadena resultante.

Por ejemplo, si tenemos la cadena "Hola mundo" y queremos obtener la subcadena "mundo", podemos utilizar la función SUBSTRING de la siguiente manera:

```
SELECT substring('Hola mundo', 6, 5) --retorna: 'mundo'.
```

El resultado sería "mundo", ya que la función comienza a partir del sexto carácter de la cadena "Hola mundo" (que es la letra "m") y toma los siguientes 5 caracteres.

- **STR (numero, longitud, cantidaddecimales):** en SQL convierte un número en una cadena de caracteres. Esta función toma tres parámetros: el número que se quiere convertir, la longitud de la cadena resultante y la cantidad de decimales que se quieren mostrar.

Por ejemplo, supongamos que queremos mostrar el número 3.1416 como una cadena de caracteres con una longitud de 6 y solo mostrar dos decimales. Podríamos utilizar la función str() de la siguiente manera:

```
SELECT str(3.1416, 6, 2); --retorna: '3.14'.
```

El resultado de este query sería la cadena de caracteres ' 3.14', con un espacio antes del número para completar la longitud de 6 caracteres.

- **STUFF (cadena1, inicio, cantidad, cadena2):** en SQL permite modificar una cadena (cadena1) insertando la cadena (cadena2) en una posición específica (inicio), reemplazando un número de caracteres determinado (cantidad). La función devuelve la cadena resultante después de realizar la modificación.

Por ejemplo, Supongamos que tenemos la siguiente cadena: "Hola Mundo". Si queremos reemplazar los caracteres "la " (es decir, "la" seguido de un espacio en blanco) en la cadena original con la cadena "mundo ", podemos usar la función STUFF de la siguiente manera:

```
SELECT stuff('Hola Mundo', 3, 3, 'mundo') --retorna: 'Homundo Mundo'
```

La función STUFF tomará la cadena original "Hola Mundo", comenzará en la posición 3, reemplazará 3 caracteres ("la ") con la cadena "mundo ", y devolverá el resultado final: "Homundo Mundo".

- **LEN (cadena):** en SQL devuelve la longitud de una cadena dada como argumento. Es decir, retorna el número de caracteres que tiene la cadena.

Por ejemplo, si tenemos la cadena "abcde", la función "len('abcde')" devolverá el valor 5, que es el número de caracteres en la cadena.

```
SELECT len('abcde') --retorna: 5.
```

- **CHAR (x):** en SQL toma un número entero (x) como argumento y devuelve el caracter correspondiente en código ASCII. El número entero debe estar en el rango de 0 a 255, que es el rango válido de valores de código ASCII.

Por ejemplo, si queremos obtener el caracter en código ASCII correspondiente al número 65, que representa la letra "A", podemos usar la función "char(65)" y el resultado será la letra "A".

```
SELECT char(65) --retorna: 'A'.
```

- **LEFT (cadena, longitud):** en SQL devuelve la cantidad de caracteres especificados (longitud) de una cadena dada (cadena), comenzando desde el primer caracter en la izquierda. Es decir, esta función devuelve una subcadena que comienza desde el inicio de la cadena y tiene una longitud específica.

Por ejemplo, si tenemos la cadena "Hola Mundo" y queremos obtener los primeros cuatro caracteres, podemos utilizar la función "left('Hola Mundo', 4)" y el resultado será "Hola".

```
SELECT left('Hola Mundo', 4) --retorna: 'Hola'.
```

- **RIGHT (cadena, longitud):** en SQL devuelve la cantidad de caracteres especificados (longitud) de una cadena dada (cadena), comenzando desde el último carácter en la derecha. Es decir, esta función devuelve una subcadena que comienza desde el final de la cadena y tiene una longitud específica.

Por ejemplo, si tenemos la cadena "Hola Mundo" y queremos obtener los últimos cuatro caracteres, podemos utilizar la función "right('Hola Mundo', 5)" y el resultado será "Mundo".

```
SELECT right('Hola Mundo', 5) --retorna: 'Mundo'.
```

- **LOWER (cadena):** en SQL convierte todos los caracteres de la cadena dada (cadena) a minúsculas y devuelve la nueva cadena como resultado.

Por ejemplo, si tenemos la cadena "HOLA MUNDO" y queremos convertirla a minúsculas, podemos utilizar la función "lower('HOLA MUNDO')" y el resultado será "hola mundo".

```
SELECT lower('HOLA MUNDO') --retorna: 'hola mundo'.
```

- **UPPER (cadena):** en SQL convierte todos los caracteres de la cadena dada (cadena) a mayúsculas y devuelve la nueva cadena como resultado.

Por ejemplo, si tenemos la cadena "Hola Mundo" y queremos convertirla a mayúsculas, podemos utilizar la función "upper('Hola Mundo')" y el resultado será "HOLA MUNDO".

```
SELECT upper('Hola Mundo') --retorna: 'HOLA MUNDO'.
```

- **LTRIM (cadena):** en SQL elimina los espacios en blanco al principio de la cadena dada (cadena) y devuelve la nueva cadena sin esos espacios.

Por ejemplo, si tenemos la cadena " Hola Mundo" y queremos eliminar los espacios en blanco al principio, podemos utilizar la función "ltrim(' Hola Mundo')" y el resultado será "Hola Mundo".

```
SELECT ltrim(' Hola Mundo') --retorna: 'Hola Mundo'.
```

- **RTRIM (cadena):** en SQL elimina los espacios en blanco al final de la cadena dada (cadena) y devuelve la nueva cadena sin esos espacios.

Por ejemplo, si tenemos la cadena "Hola Mundo " y queremos eliminar los espacios en blanco al final, podemos utilizar la función "rtrim('Hola Mundo ')" y el resultado será "Hola Mundo".

```
SELECT rtrim('Hola Mundo ') --retorna: 'Hola Mundo'.
```

- **REPLACE (cadena, cadenareemplazo, cadenareemplazar):** en SQL busca todas las ocurrencias de la subcadena "cadenareemplazo" en la cadena "cadena" y las reemplaza por la subcadena "cadenareemplazar".

Por ejemplo, si tenemos la cadena "Hola Mundo, Hola Amigos" y queremos reemplazar todas las ocurrencias de "Hola" por "Adiós", podemos utilizar la función "replace('Hola Mundo, Hola Amigos', 'Hola', 'Adiós')" y el resultado será "Adiós Mundo, Adiós Amigos".

```
SELECT replace('Hola Mundo, Hola Amigos', 'Hola', 'Adiós')
--retorna: 'Adiós Mundo, Adiós Amigos'.
```

- **REPLICATE (cadena, cantidad):** en SQL toma una cadena y la repite "cantidad" veces, devolviendo la nueva cadena resultante.

Por ejemplo, si tenemos la cadena "Hola" y queremos repetirla tres veces, podemos utilizar la función "replicate('Hola', 3)" y el resultado será "HolaHolaHola".

```
SELECT replicate('Hola', 3) --retorna: 'HolaHolaHola'.
```

- **SPACE (cantidad):** en SQL crea una cadena de espacios en blanco de una longitud especificada por el argumento "cantidad". El valor de "cantidad" debe ser un número entero positivo.

Por ejemplo, si queremos crear una cadena de 5 espacios en blanco, podemos utilizar la función "space(5)" y el resultado será una cadena de 5 espacios.

```
SELECT 'Hola' + space(5) + 'Mundo' --retorna: 'Hola      Mundo'.
```

Se quiere mostrar el nombre del vendedor seguido por el apellido en mayúscula separados por 10 espacios todo en una misma columna

```
select
nom_vendedor+SPACE(10)+UPPER(ape_vendedor)
from vendedores
```



	(No column name)
1	Martin CARRIZO
2	Marcela LEDESMA
3	Alejandro LOPEZ
4	Marcelo MIRANDA
5	Gabriel MONTE
6	Susana JUAREZ
7	Ana ORTEGA
8	Juan MONTE
9	Ana ORTEGA

Ilustración 23: Elaboración propia

Funciones matemáticas

Las funciones matemáticas en SQL operan con valores numéricos y retornan un resultado numérico. Estas funciones se utilizan para realizar operaciones aritméticas, redondear números, obtener valores absolutos, entre otras operaciones.

A continuación, veremos algunas de las funciones matemáticas más comunes en SQL:

Función	Definición	Ejemplo
ABS	Retorna el valor absoluto de un número.	ABS (-5) retorna 5
CEILING	Redondea un número hacia arriba al entero más cercano.	CEILING (3.1416) retorna 4
FLOOR	Redondea un número hacia abajo al entero más cercano.	FLOOR (3.1416) retorna 3
ROUND	Redondea un número al número de decimales especificado.	ROUND (3.1416, 2) retorna 3.14
SQRT	Retorna la raíz cuadrada de un número.	SQRT (25) retorna 5
POWER	Eleva un número a la potencia especificada por el exponente.	POWER (2, 3) retorna 8
RAND	Retorna un número aleatorio entre 0 y 1.	RAND () retorna un número aleatorio entre 0 y 1
SIGN	Retorna el signo de un número. (-1 para negativo, 0 para cero, 1 para positivo)	SIGN (-5) retorna -1, SIGN (0) retorna 0, SIGN (5) retorna 1
TRUNCATE	Trunca un número al número de decimales especificado.	TRUNCATE (3.1416, 2) retorna 3.14
LOG	Retorna el logaritmo natural de un número.	LOG (10) retorna 2.30258509299405

Tabla 1: Elaboración propia.

Funciones para el uso de fechas y horas

Las funciones para el uso de fechas y horas son de gran utilidad en el mundo de la gestión de bases de datos, ya que permiten realizar operaciones específicas con este tipo de datos y obtener información precisa y relevante en función de nuestras necesidades. Microsoft SQL Server ofrece una variedad de funciones para trabajar con fechas y horas, lo que nos permite realizar cálculos, comparaciones y manipulaciones con este tipo de datos de manera eficiente y precisa.

A continuación, te presento algunas de las funciones más utilizadas en SQL Server para trabajar con fechas y horas.

- **GETDATE():** en SQL Server retorna la fecha y hora actuales del sistema donde se está ejecutando el servidor de base de datos. Esta función no necesita argumentos y retorna un valor de tipo datetime.


```
SELECT GETDATE ()
```

Donde se muestra la fecha actual del sistema junto con la hora y los milisegundos.

- **YEAR(fecha):** Esta función toma como entrada una fecha y devuelve un número entero de 4 dígitos que representa el año correspondiente. ***El tipo de dato de entrada debe ser fecha y la salida es un entero.***

Por ejemplo, si se le pasa la fecha '15/03/2023', la función retornará el valor 2023, que corresponde al año de la fecha.

```
SELECT YEAR('15/03/2023') --retorna:2023
```

Para obtener el año actual, se puede utilizar la función YEAR() junto con GETDATE(). Esto retornará el año actual en un número entero de cuatro dígitos. Por ejemplo, la siguiente consulta SQL devolverá el año actual:

```
SELECT YEAR(GETDATE ())
```

- **MONTH(fecha):** Esta función acepta una fecha como entrada y devuelve un entero que representa el mes de la fecha. ***Toma como argumento una fecha y retorna el mes correspondiente como un número entero entre 1 y 12.*** El tipo de dato de entrada debe ser de fecha y la salida es un entero.

Por ejemplo, si se le pasa la fecha '15/03/2022', la función retornará el valor 3.

```
SELECT MONTH('15/03/2022') --retorna:3
```

Para obtener el mes actual, se puede utilizar la función MONTH() junto con GETDATE(). Esto retornará el mes actual en un número entero entre 1 y 12. Por ejemplo, la siguiente consulta SQL devolverá el mes actual:

```
SELECT MONTH(GETDATE ())
```

- **DAY(fecha):** Esta función acepta una fecha como entrada y devuelve un entero que representa el día del mes de la fecha. ***Toma como argumento una fecha y retorna el día correspondiente como un número entero entre 1 y 31.*** El tipo de entrada debe ser un tipo de dato de fecha y la salida es un entero que representa el día del mes de la fecha.

Por ejemplo, si se le pasa la fecha '15/03/2022', la función retornará el valor 15.

```
SELECT DAY('15/03/2022') --retorna:15
```

Para obtener el mes actual, se puede utilizar la función DAY() junto con GETDATE(). Esto retornará el mes actual en un número entero entre 1 y 31. Por ejemplo, la siguiente consulta SQL devolverá el día actual:

```
SELECT DAY (GETDATE ( ))
```

- **DATEPART (partedefecha, fecha):** es una función de fecha y hora en SQL que se utiliza para obtener una parte específica de una fecha. "partedefecha" es una cadena que especifica la parte de la fecha que se desea obtener, como año, mes, día, hora, minuto, segundo, etc. "fecha" es la fecha de la que se desea obtener la parte. Los valores disponibles para utilizar en la variable 'partedefecha' se pueden visualizar en la siguiente tabla:

Intervalo	Definición	Ejemplo de consulta	Resultado
year (año)	Devuelve el año de la fecha	SELECT DATEPART (year, '09/03/2022 14:30:00.000')	2022
quarter (cuarto)	Devuelve el cuarto (trimestre) de la fecha	SELECT DATEPART (quarter, '09/03/2022 14:30:00.000')	1
month (mes)	Devuelve el mes de la fecha	SELECT DATEPART (month, '09/03/2022 14:30:00.000')	3
day (día)	Devuelve el día del mes de la fecha	SELECT DATEPART (day, '09/03/2022 14:30:00.000')	9
week (semana)	Devuelve el número de la semana del año de la fecha	SELECT DATEPART (week, '09/03/2022 14:30:00.000')	10
hour (hora)	Devuelve la hora de la fecha	SELECT DATEPART (hour, '09/03/2022 14:30:00.000')	14
minute (minuto)	Devuelve el minuto de la fecha	SELECT DATEPART (minute, '09/03/2022 14:30:00.000')	30
second (segundo)	Devuelve el segundo de la fecha	SELECT DATEPART (second, '09/03/2022 14:30:00.000')	0
millisecond (milisegundo)	Devuelve el milisegundo de la fecha	SELECT DATEPART (millisecond, '09/03/2022 14:30:00.000')	0

Tabla 1: Elaboración propia.

DATEPART es una función útil para realizar cálculos con fechas, como ordenar registros por fecha o calcular la edad de una persona a partir de su fecha de nacimiento. Por ejemplo *la edad de los vendedores*. Podemos usar la función DATEPART para obtener el año actual y restar el año de nacimiento de cada empleado para obtener su edad en años:

```
SELECT ape_vendedor + ', ' + nom_vendedor 'vendedor',
       DATEPART (year, GETDATE ( )) - DATEPART (year, fec_nac) 'edad'
FROM vendedores
```

- **DATENAME (partedefecha, fecha):** en SQL Server retorna el nombre de una parte específica de una fecha. Los valores posibles para la variable "partedefecha". Los valores disponibles para utilizar en la variable 'partedefecha' se pueden visualizar en la siguiente tabla:

Intervalo	Definición	Ejemplo de consulta	Resultado
year (año)	Retorna el año de la fecha	SELECT DATENAME(year, '09/03/2022')	"2023"
quarter (cuarto)	Retorna el cuarto (trimestre) de la fecha	SELECT DATENAME(quarter, '09/03/2022')	"1"
month (mes)	Retorna el nombre del mes de la fecha	SELECT DATENAME(month, '09/03/2022')	"Marzo"
day (día)	Retorna el día del mes de la fecha	SELECT DATENAME(day, '09/03/2022')	"9"
week (semana)	Retorna el número de la semana del año de la fecha	SELECT DATENAME(week, '09/03/2022')	"10"
weekday (día de la semana)	Retorna el nombre del día de la semana de la fecha	SELECT DATENAME(weekday, '09/03/2022')	"Miércoles"
hour (hora)	Retorna la hora de la fecha	SELECT DATENAME(hour, '09/03/2022 13:45:30')	"13"
minute (minuto)	Retorna el minuto de la fecha	SELECT DATENAME(minute, '09/03/2022 13:45:30')	"45"
second (segundo)	Retorna el segundo de la fecha	SELECT DATENAME(second, '09/03/2022 13:45:30')	"30"
millisecond (milisegundo)	Retorna el milisegundo de la fecha	SELECT DATENAME(millisecond, '09/03/2022 13:45:30.123')	"123"

Tabla 1: Elaboración propia.

Tenga en cuenta que el resultado de la función "DATENAME()" puede variar según el idioma configurado en el servidor. Y es una función útil para formatear fechas y presentarlas de manera más legible para los usuarios finales

- **DATEADD (partedelafecha, cantidad, fecha)** es una función de fecha en SQL que permite agregar una cantidad específica de tiempo a una fecha determinada. La función devuelve una nueva fecha y hora que resulta de agregar la cantidad específica de tiempo a la fecha dada. El parámetro "partedelafecha" indica la unidad de tiempo que se desea agregar (como año,

mes, día, hora, etc.), mientras que "cantidad" es la cantidad de unidades de tiempo que se desea agregar. Por último, "fecha" es la fecha a la que se desea agregar la cantidad de tiempo especificada. Los valores disponibles para utilizar en la variable 'partedefecha' se pueden visualizar en la siguiente tabla:

Intervalo	Definición	Ejemplo de consulta	Resultado
year (año)	Agrega o resta un número determinado de años a una fecha	SELECT DATEADD (year , 2, '09/03/2022')	09/03/2024
quarter (cuarto)	Agrega o resta un número determinado de cuartos (trimestres) a una fecha	SELECT DATEADD (quarter , -1, '09/03/2022')	09/12/2021
month (mes)	Agrega o resta un número determinado de meses a una fecha	SELECT DATEADD (month , 3, '09/03/2022')	09/06/2022
day (día)	Agrega o resta un número determinado de días a una fecha	SELECT DATEADD (day , -7, '09/03/2022')	02/03/2022
week (semana)	Agrega o resta un número determinado de semanas a una fecha	SELECT DATEADD (week , 2, '09/03/2022')	23/03/2022
hour (hora)	Agrega o resta un número determinado de horas a una fecha	SELECT DATEADD (hour , 4, '09/03/2022 08:30:00')	09/03/2022 12:30
minute (minuto)	Agrega o resta un número determinado de minutos a una fecha	SELECT DATEADD (minute , 15, '09/03/2022 08:30:00')	09/03/2022 8:45
second (segundo)	Agrega o resta un número determinado de segundos a una fecha	SELECT DATEADD (second , 30, '09/03/2022 08:30:00')	09/03/2022 8:30
millisecond (milisegundo)	Agrega o resta un número determinado de milisegundos a una fecha	SELECT DATEADD (millisecond , 500, '09/03/2022 08:30:00.000')	2022-03-09 08:30:00.500

Tabla 1: Elaboración propia.

- **DATEDIFF(partedelfecha, fecha1, fecha2):** es una función de fecha y hora en SQL que se utiliza para calcular la diferencia entre dos fechas. Calcula la diferencia entre las dos fechas en función de la unidad de tiempo especificada y devuelve el resultado como un número entero. El primer argumento, "partedelfecha", especifica la unidad de tiempo en la que se desea calcular la diferencia (como días, meses, años, etc.). "fecha1" y "fecha2" son las dos fechas que se comparan.

Intervalo	Definición	Ejemplo	Resultado
year (año)	Diferencia entre los años de las dos fechas	SELECT DATEDIFF (year , '01/01/2021', '09/03/2022')	2
quarter (cuarto)	Diferencia entre los trimestres de las dos fechas	SELECT DATEDIFF (quarter , '01/01/2021', '09/03/2022')	9
month (mes)	Diferencia entre los meses de las dos fechas	SELECT DATEDIFF (month , '01/01/2021', '09/03/2022')	26
dayofyear (día del año)	Diferencia entre los días del año de las dos fechas	SELECT DATEDIFF (dayofyear , '01/03/2022', '31/12/2022')	305
day (día)	Diferencia entre los días de las dos fechas	SELECT DATEDIFF (day , '01/01/2021', '09/03/2022')	798
week (semana)	Diferencia entre las semanas de las dos fechas	SELECT DATEDIFF (week , '01/01/2021', '09/03/2022')	114
hour (hora)	Diferencia entre las horas de las dos fechas	SELECT DATEDIFF (hour , '01/01/2021 00:00:00', '2021-01-02 12:00:00')	36
minute (minuto)	Diferencia entre los minutos de las dos fechas	SELECT DATEDIFF (minute , '01/01/2021 00:00:00', '01/01/2021 01:30:00')	90
second (segundo)	Diferencia entre los segundos de las dos fechas	SELECT DATEDIFF (second , '01/01/2021 00:00:00', '01/01/2021 00:01:15')	75
millisecond (milisegundo)	Diferencia entre los milisegundos de las dos fechas	SELECT DATEDIFF (millisecond , '01/01/2021 00:00:00', '01/01/2021 00:00:00.500')	500

Tabla 1: Elaboración propia.

Un ejemplo más sobre funciones DAY(), MONTH() y YEAR(), sería: obtener el día, mes y año de emisión de todas las facturas en columnas separadas:

```
SELECT nro_factura,
       DAY (fecha)   AS Dia,
       MONTH (fecha) AS Mes,
       YEAR (fecha)  AS Año
FROM facturas;
```

nro_factura	Dia	Mes	Año
200	9	1	2020
201	12	1	2020
202	15	1	2020
203	17	1	2020
204	18	1	2020
205	20	1	2020
206	21	1	2020
207	22	1	2020
208	25	1	2020
209	27	1	2020
210	31	1	2020

- **datepart**(partedefecha, fecha): retorna la parte específica de una fecha, el año, trimestre, día, hora, etc. Los valores para "partedefecha" pueden ser: year (año), quarter (cuarto), month (mes), day (día), week (semana), hour (hora), minute (minuto), second (segundo) y millisecond (milisegundo).

BIBLIOGRAFÍA

Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D. & Woody B (2019) Introducing Microsoft SQL Server 2019. Reliability, scalability, and security both on premises and in the cloud. Packt Publishing Ltd. Birmingham UK

Microsoft. SQL Server 2016. Disponible en: <https://www.microsoft.com/es-es/sql-server/sql-server-2016>

Opel, A. & Sheldon, R. (2010). Fundamentos de SQL. Madrid. Editorial Mc Graw Hill

Varga S., Cherry D., D'Antoni J. (2016). Introducing Microsoft SQL Server 2016 Mission-Critical Applications, Deeper Insights, Hyperscale Cloud. Washington. Microsoft Press



Atribución-No Comercial-Sin Derivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Referenciarlo de la siguiente manera: Universidad Tecnológica Nacional Facultad Regional Córdoba (S/D). Material para la Tecnicatura Universitaria en Programación, modalidad virtual, Córdoba, Argentina.