BASE DE
DATOS 1º AÑO
2025

Clase Nº 4: Consultas y funciones SQL

Contenido:

- Consultas y mantenimiento avanzado en SQL.
- Uso de subconsultas en cláusulas complejas (WHERE, FROM).
- Aplicación de funciones y operadores de conjuntos (UNION, INTERSECT, EXCEPT).
- Consultas multitablas y operadores JOIN.
- Uso de referencias externas (EXISTS, NOT EXISTS).

En la clase de hoy trabajaremos los siguientes temas:

- Hacer consultas y mantenimiento más avanzados.
- Utilización de subconsultas en cláusulas más complejas.
- Realizar Funciones.
- Utilizar Operadores de conjuntos.
- Utilizar Consultas multitablas.
- Crear nuevas bases de datos más complejas.

1. **Presentación:**

¡Bienvenidos a la Clase N° 4 de Base de Datos!
Hoy trabajaremos con consultas SQL avanzadas que nos permitirán extraer, analizar y manipular datos de manera más eficiente. También exploramos el uso de funciones, operadores de conjuntos y consultas que combinan varias tablas.
Este conocimiento es esencial para cualquier administrador de bases de datos o desarrollador que trabaje con SQL.
Veremos ejemplos de operaciones más complejas y cláusulas más específicas, donde se utilizarán más operadores lógicos y consultas multitablas, donde se pondrá en práctica en ejercicios más dinámicos, de modo que el lector pueda avanzar progresivamente en el uso de SQL.

En este video vamos a ver conceptos de cómo llevar a cabo SUBCONSULTAS en SQL para poder trabajar con distintas Bases de datos.

Link:https://www.youtube.com/watch?v=xuASGBwNboU&ab_channel=TodoCode

2.-DESARROLLO

Funciones de

columna

En ocasiones es necesario contar datos: ¿cuántos clientes hay en Castellón?O también hacer cálculos sobre ellos: ¿a cuánto asciende el IVA cobrado en la factura 3752 SQL proporciona una serie de funciones que se pueden utilizar en la cláusula SELECT y que actúan sobre los valores de las columnas para realizar diversas operaciones como, por ejemplo, sumarlos y obtener el valor máximo el valor medio, entre otros. Las funciones de columna más habituales son las que se muestran a continuación:

Función	Descripción
`COUNT(*)`	Cuenta el número total de filas.
`COUNT(columna)`	Cuenta los valores **no nulos** en una columna.
`SUM(columna)`	Suma todos los valores de la columna.
`MAX(columna)`	Obtiene el valor máximo de la columna.
`MIN(columna)`	Obtiene el valor mínimo de la columna.
`AVG(columna)`	Calcula el promedio de los valores.

Ejemplo:

SELECT AVG(precio) AS precio_medio FROM productos WHERE categoria = 'Electrónica';

Ejemplo con COUNT():

SELECT COUNT(*) AS total_ventas FROM ventas WHERE fecha >= '2024-01-01';

A continuación, se muestran mas ejemplos:

-- cantidad media por línea de

factura SELECT AVG(cant) FROM

lineas_fac;

-- cantidad media por línea de factura del artículo

SELECT AVG(cant)

FROM lineas_fac

WHERE codart = 'TLFXK2';

-- se puede hacer varios cálculos

a la vez SELECT SUM(cant) AS

suma, COUNT(*) AS lineas FROM

lineas_fac;

La función COUNT() realiza operaciones distintas dependiendo de su argu- mento: COUNT(*) Cuenta filas.

COUNT(columna) Cuenta el número de valores no nulos en La

columna. COUNT(DISTINCT columna) Cuenta el número de

valores distintos y No nulos en la columna.

A continuación, se muestra su uso mediante un ejemplo. Se ha

creado una tabla P que contiene los datos de una serie de piezas:

SELECT * FROM P;

pnum | pnombre | color | peso |

ciudad P1 | tuerca | verde | 12 |

París

P2 | perno | rojo | |

Londres P3 | birlo | azul

| 17 | Roma P4 | birlo

| rojo | 14 | Londres P5 |

leva | | 12 | París

P6 | engrane | rojo | 19 | París

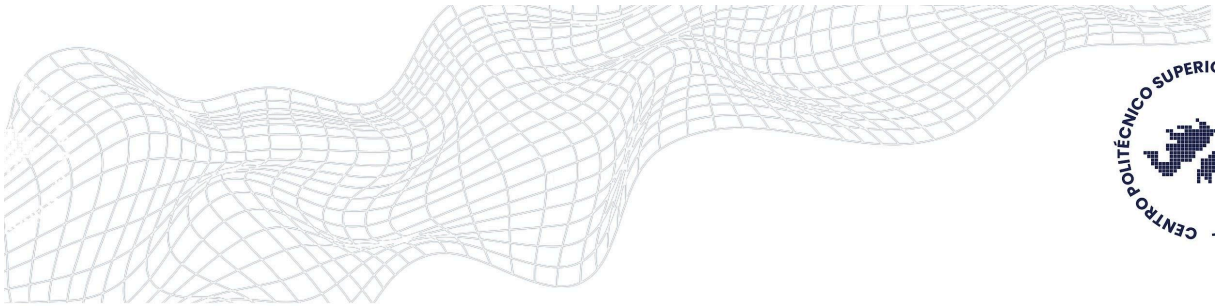
y se ha ejecutado la siguiente sentencia: SELECT COUNT(*) AS cuenta1, COUNT(color) AS cuenta2, COUNT(DISTINCT color) AS cuenta3

FROM P; El resultado de ejecutarla será el siguiente:

cuenta1 | cuenta2 | cuenta36 | 5 | 35

A la vista de los resultados se puede decir que cuenta1 contiene el número de piezas, cuenta2 contiene el número de piezas con color y cuenta3 contiene el número de colores de los que hay piezas.

Las funciones de columna (SUM, MAX, MIN, AVG) ignoran los nulos, es



decir, los nulos no son tenidos en cuenta en los cálculos. Según esto, se plantea la siguiente pregunta: ¿coincidirá siempre el valor de media1 y media2 al ejecutar la siguiente sentencia?

SELECT AVG(dto) AS media1, SUM(dto)/COUNT(*) AS media2 FROM lineas_fac;

La respuesta es negativa, ya que en media1 se devuelve el valor medio de los descuentos no nulos, mientras que en media2 lo que se devuelve es el valor medio de los descuentos (interpretándose los descuentos nulos como el descuento cero). Como se ha visto, la función AVG calcula la media de los valores no nulos de una columna. Si la tabla de la cláusula FROM es la de artículos, la media es por artículo; si la tabla de la cláusula FROM es la de facturas, la media espor factura. Cuando se quiere calcular otro tipo de media se debe hacer el cálculo mediante un cociente. Por ejemplo, el número medio de facturas por mes durante el año pasado se obtiene dividiendo el número de facturas del año pasado entre doce meses:

SELECT COUNT(*)/12 AS media_mensual FROM facturas

WHERE EXTRACT(year FROM fecha) = EXTRACT(year

FROM CURRENT_DATE)-1;

Es importante tener en cuenta que la función COUNT devuelve un entero y que las operaciones entre enteros devuelven resultados enteros. Es decir, la operación SELECT 2/4; devuelve el resultado cero. Por lo tanto, es conveniente multiplicar uno de los operados por 1.0 para asegurarse de que se opera con números reales. En este caso, será necesario redondear los decimales del resultado a lo que sea

preciso:

```
SELECT ROUND(COUNT(*)*1.0/12,2) AS media_mensual FROM facturas
WHERE EXTRACT(year FROM fecha) = EXTRACT(year FROM
CURRENT_DATE)-1;
```

Cláusula GROUP BY

La cláusula GROUP BY forma grupos con las filas que tienen en común los valores de una o varias columnas. Sobre cada grupo se pueden aplicar las fun- ciones de columna que se han estado utilizando hasta ahora (SUM, MAX, MIN, AVG, COUNT), que pasan a denominarse funciones de grupo. Estas funciones, utilizadas en la cláusula SELECT, se aplican una vez para cada grupo.

La siguiente sentencia cuenta cuántas facturas tiene cada cliente el año pasado: SELECT codcli, COUNT(*) FROM facturas WHERE EXTRACT(year FROM fecha) = EXTRACT(year FROM CURENT_DATE)-1 GROUP BY codcli;

El modo en que se ejecuta la sentencia se explica a continuación. Se toma la tabla de facturas (FROM) y se seleccionan las filas que cumplen la restricción (WHERE). A continuación, las facturas se separan en grupos, de modo que en un mismo grupo sólo hay facturas de un mismo cliente (GROUP BY codcli), con lo cual hay tantos grupos como clientes hay con facturas del año pasado. Finalmente, de cada grupo se muestra el código del cliente y el número de facturas que hay en el grupo (son las facturas de ese cliente): COUNT(*).

La cláusula GROUP BY agrupa filas con valores en común y permite aplicar funciones de agregación.

Ejemplo:

```
SELECT categoria, COUNT(*) AS total_productos
FROM productos
GROUP BY categoria;
```

- **Diferencia entre WHERE y HAVING**

- WHERE filtra filas **antes** de aplicar GROUP BY.
- HAVING filtra **después** de agrupar los datos.

Ejemplo con HAVING:

```
SELECT categoria, COUNT(*) AS total_productos
FROM productos
GROUP BY categoria
HAVING COUNT(*) > 10;
```

Cláusula HAVING

En la cláusula HAVING, que puede aparecer tras GROUP BY, se utilizan las funciones de grupo para hacer restricciones sobre los grupos que se han formado. La sintaxis de la sentencia SELECT, tal y como se ha visto hasta el momento, es la siguiente: SELECT [DISTINCT] { * | columna [, columna] } FROM tabla[WHERE condición_de_búsqueda] [GROUP BY columna [,columna][HAVING condición_para_el_grupo]] [ORDER BY columna [ASC|DESC] [,columna [ASC | DESC]];

En las consultas que utilizan GROUP BY se obtiene una fila por cada uno de los grupos producidos. Para ejecutar la cláusula GROUP BY se parte de las filas de la tabla que cumplen el predicado establecido en la cláusula WHERE y se agrupan en función de los valores comunes en la columna o columnas especificadas. Mediante la

cláusula HAVING se realiza una restricción sobre los grupos obtenidos por la cláusula GROUP BY, y se seleccionan aquellos que cumplen el predicado establecido en la condición.

Evidentemente, en la condición de la cláusula HAVING sólo pueden aparecer restricciones sobre columnas por las que se ha agrupado y también funciones de grupo sobre cualquier otra columna de la tabla. Lo mismo sucede en la cláusula SELECT: sólo es posible especificar de manera directa columnas que aparecen en la cláusula GROUP BY y también funciones de grupo sobre cualquier otra columna. Cuando en las cláusulas SELECT o HAVING aparecen columnas que no se han especificado en la cláusula GROUP BY y que tampoco están afectadas por una función de grupo, se produce un error.

Subconsultas en WHERE, FROM y HAVING

Una subconsulta es una consulta dentro de otra consulta.

Ejemplo en WHERE:

```
SELECT * FROM empleados
WHERE salario > (SELECT AVG(salario) FROM empleados);
```

Ejemplo en HAVING:

```
SELECT departamento, COUNT(*) AS total_empleados
FROM empleados GROUP BY departamento  HAVING COUNT(*) >
(SELECT COUNT(*)FROM empleados WHERE cargo = 'Gerente');
```

Subconsultas

Una subconsulta es una sentencia SELECT anidada en otra sentencia SQL, que puede ser otra SELECT o bien cualquier sentencia de manejo de datos (INSERT, UPDATE, DELETE). Las subconsultas pueden anidarse unas dentro de otras tanto como sea necesario (cada SGBD puede tener un nivel máximo de anidamiento, que difícilmente se alcanzará). En este apartado se muestra cómo el uso de subconsultas en las cláusulas WHERE y HAVING otorga mayor potencia para la realización de restricciones. Además, en este apartado se introduce el uso de subconsultas en la cláusula FROM.

Subconsultas en la cláusula WHERE

La cláusula WHERE se utiliza para realizar restricciones a nivel de filas. El

predicado que se evalúa para realizar una restricción está formado por comparaciones unidas por los operadores AND/OR. Cada comparación involucra dos operando que pueden ser:

(a) Dos columnas de la tabla sobre la que se realiza la consulta.

-- artículos cuyo stock es el mínimo deseado

SELECT * FROM articulos WHERE stock = stock_min;

(b) Una columna de la tabla de la consulta y una constante.

-- artículos cuya descripción empieza como se indica

SELECT * FROM articulos WHERE UPPER(descrip) LIKE 'PROLONG%';

(c) Una columna o una constante y una subconsulta

sobre alguna tabla dela base de datos.

-- artículos vendidos con descuento mayor del 45%

SELECT * FROM artículos WHERE codart IN (SELECT codart FROM lineas_fac WHERE dto > 45);

Subconsultas en la cláusula HAVING

La cláusula HAVING permite hacer restricciones sobre grupos y necesariamente va precedida de una cláusula GROUP BY. Para hacer este

tipo de restricciones también es posible incluir subconsultas cuando sea necesario. La siguiente consulta obtiene el código del pueblo que tiene más clientes:

```
SELECT codpue FROM clientes GROUP BY codpue HAVING COUNT(*) >= ALL (
SELECT COUNT(*) FROM clientes GROUP BY
codpue );
```

En primer lugar se ejecuta la subconsulta, obteniéndose una columna de números en donde cada uno indica el número de clientes en cada pueblo. La subconsulta se sustituye entonces por los valores de esta columna, por ejemplo:

```
SELECT codpue FROM clientes GROUP BY codpue HAVING COUNT(*) >= ALL (1,4,7,9,10);
```

Por último, se ejecuta la consulta principal. Para cada grupo se

cuenta el número de clientes que tiene. Pasan la restricción del HAVING aquel o aquellos pueblos que en esa cuenta tienen el máximo valor.

SUBCONSULTAS EN LA CLÁUSULA FROM

También es posible incluir subconsultas en la cláusula FROM, aunque en este caso no se utilizan para construir predicados sino para realizar una consulta sobre la tabla que se obtiene como resultado de ejecutar otra consulta. Siempre que se utilice una subconsulta en el FROM se debe dar un nombre a la tabla resultado mediante la cláusula AS.

```
SELECT COUNT(*), MAX(ivat), MAX(dtot) FROM ( SELECT DISTINCT
COALESCE(iva,0)  AS ivat, COALESCE(dto,0) AS dtot
FROM facturas ) AS t;
```

La consulta anterior cuenta las distintas combinaciones de IVA y descuento y muestra el valor máximo de éstos. Nótese que se han renombrado las columnas de la subconsulta para poder referenciarlas en la consulta principal. Esta consulta no se puede resolver si no es de este modo ya que COUNT no acepta una lista de columnas como argumento.

Consultas multitabla

En este apartado se muestra cómo hacer consultas que involucran a

datos de varias tablas. Aunque mediante las subconsultas se ha conseguido realizar consultas de este tipo, aquí se verá que, en ocasiones, es posible escribir consul-tas equivalentes que no hacen uso de subconsultas y que se ejecutan de modo más eficiente. El operador que se introduce es la concatenación (JOIN).

La concatenación: JOIN

La concatenación es una de las operaciones más útiles del lenguaje SQL. Esta operación permite combinar información de varias tablas sin necesidad de utilizar subconsultas para ello.

La concatenación natural (NATURAL JOIN) de dos tablas R y S obtiene como resultado una tabla cuyas filas son todas las filas de R concatenadas con todas las filas de S que en las columnas que se llaman igual tienen los mismos valores. Las columnas por las que se hace la concatenación aparecen una sola vez en el resultado.

La siguiente sentencia hace una concatenación natural de lastablas FACTURAS y CLIENTES. Ambas tablas tienen una columna con el mismo nombre, codcli, siendo FACTURAS.codcli una clave ajena a CLIENTES.codcli (clave primaria).

```
SELECT * FROM facturas NATURAL JOIN clientes;
```

Según la definición de la operación NATURAL JOIN, el resultado tendrá las siguientes columnas: codfac, fecha, codven, iva, dto, codcli, nombre, direccion, codpostal, codpue. En el resultado de la

concatenación cada fila

representa una factura que cuenta con sus datos (la cabecera) y los datosdel cliente al que pertenece. Si alguna factura tiene codcli nulo, no aparece en el resultado de la concatenación puesto que no hay ningún cliente con el que pueda concatenarse.

Cambiando el contenido de la cláusula SELECT, cambia el resultado de la consulta.

Por ejemplo:

Los JOIN se usan para combinar datos de varias tablas.

Ejemplo con INNER JOIN:

```
SELECT empleados.nombre, departamentos.nombre AS departamento
FROM empleados
INNER JOIN departamentos ON empleados.id_departamento =
departamentos.id;
```

Ejemplo con LEFT JOIN:

```
SELECT clientes.nombre, pedidos.total
FROM clientes
LEFT JOIN pedidos ON clientes.id = pedidos.id_cliente;
```

Operadores de conjuntos

Los operadores de conjuntos del álgebra relacional son: el producto cartesiano, la unión, la intersección y la diferencia. El producto cartesiano se realiza en SQL especificando en la cláusula FROM las tablas involucradas en la operación, separadas por comas, tal y

como se ha indicado anteriormente. A continuación, se muestra cómo utilizar el resto de los operadores de conjuntos en las consultas en SQL.

La sintaxis para las uniones, intersecciones y diferencias es la siguiente: sentencia_SELECT UNION | INTERSECT | EXCEPT [ALL]sentencia_SELECT [ORDER BY columna [ASC | DESC] [,columna [ASC | DESC]];

Nótese que la cláusula ORDER BY sólo puede aparecer una vez en la consulta, al final de la misma. La ordenación se realizará sobre el resultado de la unión, intersección o diferencia.

Para poder utilizar cualquiera de estos tres nuevos operadores, las cabeceras de las

sentencias SELECT involucradas deben devolver el mismo número de columnas, y las columnas correspondientes en ambas sentencias deberán ser del mismo tipo de datos.

Operador	Descripción
`UNION`	Une los resultados de dos consultas, eliminando duplicados.
`UNION ALL`	Une los resultados de dos consultas, conservando duplicados.
`INTERSECT`	Devuelve solo los valores en común en ambas consultas.
`EXCEPT`	Devuelve los valores que están en la primera consulta pero no en la segunda.

Operador UNION

Este operador devuelve como resultado todas las filas que devuelve la primera sentencia SELECT, más aquellas filas de la segunda sentencia SELECT que no han sido ya devueltas por la primera. En el resultado no se muestran duplicados. Se puede evitar la eliminación de duplicados especificando la palabra clave ALL. En este caso, si una fila aparece m veces en la primera sentencia y n veces en la segunda, en el resultado aparecerá m + n veces.

Si se realizan varias uniones, éstas se evalúan de izquierda a derecha, a menos que se utilicen paréntesis para establecer un

orden distinto.

La siguiente sentencia muestra los códigos de las poblaciones donde hay clientes o donde hay vendedores:

SELECT codpue FROM

clientes UNION

SELECT codpue FROM vendedores;Operador INTERSECT

Este operador devuelve como resultado las filas que se encuentran tanto en el resultado de la primera sentencia SELECT como en el de la segunda sentencia SELECT. En el resultado no se muestran duplicados.

Se puede evitar la eliminación de duplicados especificando la palabra clave

ALL. En este caso, si una misma fila aparece m veces en la primera Sentencia y n veces en la segunda, en el resultado esta fila aparecerá min(m, n) veces.

Si se realizan varias intersecciones, éstas se evalúan de izquierda a derecha, a menos que se utilicen paréntesis para establecer un orden distinto. La intersección tiene más prioridad, en el orden de evaluación, que la unión, es decir, A UNION B INTERSECT C se evalúa como A UNION (B INTERSECT C).

La siguiente sentencia muestra los códigos de las poblaciones donde hay

clientes y también hay vendedores:

SELECT codpue FROM

clientes INTERSECT

SELECT codpue FROM vendedores;

Ejemplo con UNION:

SELECT nombre FROM clientes
UNION
SELECT nombre FROM proveedores;

Ejemplo con INTERSECT:

SELECT nombre FROM clientes
INTERSECT
SELECT nombre FROM empleados;

Subconsultas correlacionadas

Una subconsulta correlacionada es una consulta anidada que contiene referencias a columnas de las tablas que se encuentran en el FROM de la consulta principal. Son lo que se denomina referencias externas.

Como ya se ha visto, las subconsultas dotan al lenguaje SQL de una gran potencia.Estas pueden utilizarse para hacer restricciones, tanto en la cláusula WHERE como en la cláusula HAVING, y también en la cláusula FROM. Hasta ahora, dichas subconsultas podían tratarse de modo independiente y, para comprender mejor el funcionamiento

de la sentencia, se podía suponer que la subconsulta se ejecuta en primer lugar, y se sustituye ésta en la sentencia SELECT principal por su valor, como se muestra en el siguiente ejemplo:

-- facturas con descuento

máximo SELECT *

FROM facturas

WHERE dto = (SELECT MAX(dto) FROM facturas);

en primer lugar se obtiene el descuento máximo de las facturas, se sustituye la subconsulta por este valor y, por último, se ejecuta la consulta principal.

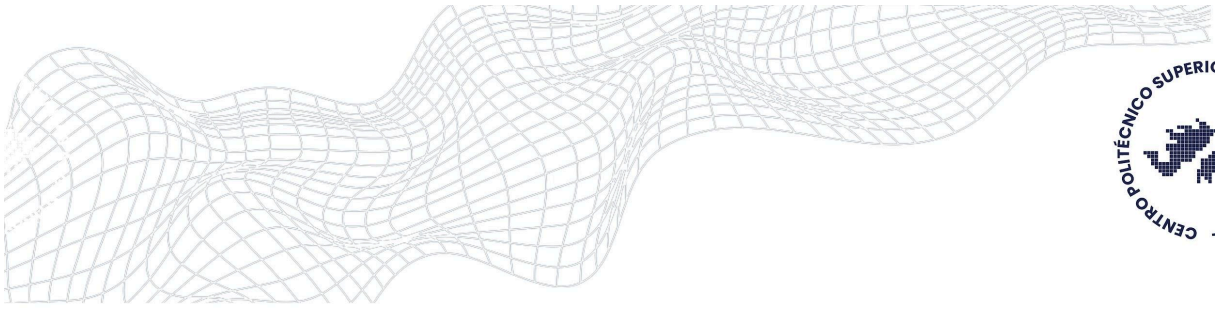
Referencias externas

En ocasiones sucede que la subconsulta se debe recalcular para cada fila de la consulta principal, estando la subconsulta parametrizada mediante valores de columnas de la consulta principal. A este tipo de subconsultas se
les llama subconsultas correlacionadas y a los parámetros de la subconsulta que pertenecen a la consulta principal se les llama referencias externas.

La siguiente sentencia obtiene los datos de las facturas que tienen descuento en todas sus líneas:

```
SELECT * FROM facturas AS f

WHERE 0 < ( SELECT MIN(COALESCE(l.dto,0))
```



```
FROM lineas_fac AS l WHERE l.codfac = f.codfac );
```

La referencia externa es f.codfac, ya que es una columna de la consulta principal. En este caso, se puede imaginar que la consulta se ejecuta del siguiente modo. Se recorre, fila a fila, la tabla de las facturas. Para cada fila se ejecuta la subconsulta, sustituyendo f.codfac por el valor que tiene en la fila actual de la consulta principal. Es decir, para cada factura se obtiene el descuento mínimo en sus líneas.

Si este descuento mínimo es mayor que cero, significa que la factura tiene descuento en todas sus líneas, por lo que se muestra en el resultado. Si no es así, la factura no se muestra. En cualquiera de los dos casos, se continúa procesando la siguiente factura: se obtienen sus líneas y el descuento mínimo en ellas, etc.

Operadores EXISTS, NOT EXISTS

En un apartado anterior se han presentado los operadores que se pueden utilizar con las subconsultas para hacer restricciones en las cláusulas WHEREy HAVING. En aquel momento no se citó, intencionadamente, un operador, ya que éste se utiliza siempre con referencias externas: el operador EXISTS. EXISTS (subconsulta)

La subconsulta se evalúa para determinar si devuelve o no alguna fila. Si devuelve al menos una fila, se evalúa a verdadero. Si no devuelve ninguna fila, se evalúa a falso. La subconsulta puede tener referencias externas, que actuarán como constantes durante la evaluación de la subconsulta.

En la ejecución de la subconsulta, en cuanto se devuelve la primera fila, se devuelve verdadero, sin terminar de obtener el resto de las filas.

Puesto que el resultado de la subconsulta carece de interés (sólo importa si se devuelve o no alguna fila), se suelen escribir las consultas indicando una constante en la cláusula SELECT en lugar de * o cualquier columna:

-- facturas que en alguna línea no

tiene dto SELECT * FROM facturas AS f

WHERE EXISTS (SELECT 1 FROM lineas_fac AS

l WHERE l.codfac = f.codfac AND

COALESCE(dto,0)=0);

NOT EXISTS (subconsulta)

La subconsulta se evalúa para determinar si devuelve o no alguna fila. Si devuelve al menos una fila, se evalúa a falso. Si no devuelve ninguna fila,se

evalúa a verdadero. La subconsulta puede tener referencias externas, que actuarán como constantes durante la evaluación de la subconsulta.

En la ejecución de la subconsulta, en cuanto se devuelve la primera fila, se devuelve falso, sin terminar de obtener el resto de las filas. Puesto que el resultado de la subconsulta carece de interés (sólo importa si se devuelve o no alguna fila), se suelen

escribir las consultas indicando unaconstante en la cláusula SELECT en lugar de * o cualquier columna:

Ejemplo con EXISTS:

```
SELECT * FROM clientes
WHERE EXISTS (SELECT 1 FROM pedidos WHERE pedidos.id_cliente = clientes.id);
```

Ejemplo con NOT EXISTS:

```
SELECT * FROM clientes
WHERE NOT EXISTS (SELECT 1 FROM pedidos WHERE pedidos.id_cliente = clientes.id);
```

Actividad 1:

1. **Uso de escenarios y casos prácticos**

- 👉 **Caso:** Imagina que tienes una base de datos de empleados y necesitas obtener el nombre del empleado con el segundo salario más alto. ¿Cómo podrías resolverlo utilizando SQL?
- **Desafío:** Los estudiantes deben investigar y proponer una solución usando subconsultas, reflexionando sobre su utilidad.

2. **Comparación de soluciones**

- busquen dos formas de resolver un mismo problema usando diferentes técnicas.
- Ejemplo: "Encuentra la cantidad total de empleados por departamento usando GROUP BY. Luego, intenta resolverlo de otra forma sin GROUP BY. ¿Qué diferencias encuentras?"
- **Objetivo:** Reflexionar sobre la importancia de GROUP BY y cuándo es

indispensable.

3. **Errores comunes y depuración**

   - Se tiene una consulta SQL con errores e identifiquen y corrijan.

Ejemplo:

```
SELECT departamento, COUNT(*)
FROM empleados
HAVING COUNT(*) > 5;
```

- **Pregunta:** ¿Qué error tiene esta consulta? ¿Cómo lo corregirías?
- **Reflexión:** Esto obliga a los estudiantes a entender HAVING y GROUP BY en lugar de solo memorizar definiciones.

4. **Aprendizaje basado en preguntas inversas**

   - "¿Qué hace EXISTS?", se podría preguntar:

Actividad 2:

-Anuncio de la Actividad Práctica: Creación y Manipulación de una Base de Datos en XAMPP

Para esta actividad, se trabajará con **XAMPP** como entorno de desarrollo, utilizando su motor de base de datos **MySQL**. Todas las prácticas se realizarán dentro de este entorno para garantizar una mejor comprensión y aplicación de los conceptos de SQL.

Instrucciones

1. **Configurar el entorno:** Asegúrate de tener instalado y en ejecución **XAMPP**. Inicia el servidor de bases de datos MySQL desde el panel de control de XAMPP.
2. **Crear una base de datos llamada** pruebas.
3. **Crear una tabla llamada** tblUsuarios en la base de datos pruebas con la siguiente estructura:

idx INT PRIMARY KEYAUTO_INCREMENT,
usuario VARCHAR(20), nombre VARCHAR(20), sexo
VARCHAR(1),
nivel TINYINT,
email VARCHAR(50),
telefono VARCHAR(20), marca VARCHAR(20),
compañia VARCHAR(20), saldo FLOAT,
activo BOOLEAN

3.-Cargar la tabla con los siguientes datos:

('1','BRE2271','BRENDA','M','2','brenda@live.com','655-330-5736','SAMSUNG','IUSACELL','100','1'),

('2','OSC4677','OSCAR','H','3','oscar@gmail.com','655-143-4181','LG','TELCEL','0', '1'),

('3','JOS7086','JOSE','H','3','francisco@gmail.com','655-143-3922','NOKIA','MOVIS TAR','150','1'),

('4','LUI6115','LUIS','H','0','enrique@outlook.com','655-137-1279','SAMSUNG','TELCEL','50','1'),

('5','LUI7072','LUIS','H','1','luis@hotmail.com','655-100-8260','NOKIA','IUSACELL','50','0'),

('6','DAN2832','DANIEL','H','0','daniel@outlook.com','655-145-2586','SONY','UNE FON','100','1'),

('7','JAQ5351','JAQUELINE','M','0','jaqueline@outlook.com','655-330-5514','BLAC KBERRY','AXEL','0','1'),

('8','ROM6520','ROMAN','H','2','roman@gmail.com','655-330-3263','LG','IUSACE LL','50','1'),

('9','BLA9739','BLAS','H','0','blas@hotmail.com','655-330-3871','LG','UNEFON','10 0','1'),
('10','JES4752','JESSICA','M','1','jessica@hotmail.com','655-143-6861','SAMSUNG'
,'TELCEL','500','1'),

('11','DIA6570','DIANA','M','1','diana@live.com','655-143-3952','SONY','UNEFON','100','0'),

('12','RIC8283','RICARDO','H','2','ricardo@hotmail.com','655-145-6049','MOTORO LA','IUSACELL','150','1'),

('13','VAL6882','VALENTINA','M','0','valentina@live.com','655-137-4253','BLACK BERRY','AT&T','50','0'),

('14','BRE8106','BRENDA','M','3','brenda2@gmail.com','655-100-1351','MOTORO LA','NEXTEL','150','1'),

('15','LUC4982','LUCIA','M','3','lucia@gmail.com','655-145-4992','BLACKBERRY'
,'IUSACELL','0','1'),

('16','JUA2337','JUAN','H','0','juan@outlook.com','655-100-6517','SAMSUNG','AX EL','0','0'),

('17','ELP2984','ELPIDIO','H','1','elpidio@outlook.com','655-145-9938','MOTOROL A','MOVISTAR','500','1'),

('18','JES9640','JESSICA','M','3','jessica2@live.com','655-330-5143','SONY','IUSA CELL','200','1'),

('19','LET4015','LETICIA','M','2','leticia@yahoo.com','655-143-4019','BLACKBER RY','UNEFON','100','1'),

('20','LUI1076','LUIS','H','3','luis2@live.com','655-100-5085','SONY','UNEFON','15

0','1'),

('21','HUG5441','HUGO','H','2','hugo@live.com','655-137-3935','MOTOROLA','AT &T','500','1');

Consultas

1. Listar los nombres de los usuarios
2. Calcular el saldo máximo de los usuarios de sexo “Mujer”
3. Listar nombre y teléfono de los usuarios con teléfono NOKIA, BLACKBERRY o SONY
4. Contar los usuarios sin saldo o inactivos
5. Listar el login de los usuarios con nivel 1, 2 o 3
6. Listar los números de teléfono con saldo menor o igual a 300
7. Calcular la suma de los saldos de los usuarios de la compañia telefónica NEXTEL
8. Contar el número de usuarios por compañía telefónica
9. Contar el número de usuarios por nivel
10. Listar el login de los usuarios con nivel 2
11. Mostrar el email de los usuarios que usan gmail
12. Listar nombre y teléfono de los usuarios con teléfono LG, SAMSUNG o MOTOROLA

3.- Actividad Integradora de Cierre:

1. Listar nombre y teléfono de los usuarios con teléfono que no sea de la marca LG o SAMSUNG
2. Listar el login y teléfono de los usuarios con compañia

telefónica IUSACELL

3. Listar el login y teléfono de los usuarios con compañia telefónica que no sea TELCEL
4. Calcular el saldo promedio de los usuarios que tienen teléfono marca NOKIA
5. Listar el login y teléfono de los usuarios con compañia telefónica IUSACELL o AXEL
6. Mostrar el email de los usuarios que no usan yahoo
7. Listar el login y teléfono de los usuarios con compañia telefónica que no sea TELCEL o IUSACELL
8. Listar el login y teléfono de los usuarios con compañia telefónica UNEFON
9. Listar las diferentes marcas de celular en orden alfabético descendentemente
10. Listar las diferentes compañias en orden alfabético aleatorio
11. Listar el login de los usuarios con nivel 0 o 2
12. Calcular el saldo promedio de los usuarios que tienen teléfono marca LG.

4.-Cierre:

Tomamos un momento para repensar lo que hemos aprendidEn esta clase se aprendió sobre bases de datos como debe ser, sin duda, Las habilidades y funciones de un administrador de bases de datos y SQL que aborda todo lo necesario para aprender a trabajar con bases de datos mediante consultas, usando el lenguaje SQL.

🎯 En esta clase aprendimos a realizar consultas avanzadas en SQL, incluyendo:

✅ Funciones de agregación (SUM, AVG, COUNT).

✅ GROUP BY y HAVING para agrupar datos.

✅ Subconsultas en WHERE, HAVING y FROM.
✅ Uso de JOIN para consultas multitabla.
✅ Operadores de conjuntos y referencias externas.
📌 **Practicar con estas consultas fortalecerá tu dominio de SQL** 🚀

5.-Bibliografía Obligatoria:

- Marquez, M. (2011) : Base de datos. Editorial Universitat Jaume.
- Catheren M. R. (2009): Base de Datos. Edición McGraw Hill