

UNIDAD DIDÁCTICA 4

REALIZACIÓN DE CONSULTAS

**MÓDULO PROFESIONAL:
BASES DE DATOS**



CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO	3
INTRODUCCIÓN	3
CASO INTRODUCTORIO	4
1. SINTAXIS Y SENTENCIAS DEL LENGUAJE SQL	5
1.1 Nombres.....	6
1.2 Expresiones	8
1.3 Funciones internas.....	9
1.4 Otras funciones en SQL.....	10
2. CLÁUSULA SELECT	11
2.1 Cláusula FROM	11
2.2 Resultados de consultas.....	12
2.3 Cláusula WHERE	15
3. CONDICIONES DE BÚSQUEDA	17
3.1 Test de comparación.....	17
3.2 Test de rango	19
3.3 Test de pertenencia a conjunto	21
3.4. Test de correspondencia con patrón	22
3.5. Test de valor nulo.....	24
3.6. Condiciones de búsqueda compuestas	25
4. COLUMNAS CALCULADAS.....	26
5. SELECCIÓN DE TODAS LAS COLUMNAS	28
6. FILAS DUPLICADAS (DISTINCT).....	29
7. ORDENACIÓN DE RESULTADOS DE UNA CONSULTA.....	30
8. COMBINACIÓN DE RESULTADOS DE UNA CONSULTA.....	34
8.1 Uniones y ordenación	36
8.2 Uniones múltiples	36
9. CONSULTAS DE RESUMEN, GROUP BY, HAVING	38
9.1 Cálculo del total de una columna (SUM)	38
9.2 Cálculo del promedio de una columna (AVG).....	39
9.3 Determinación de valores extremos (MIN Y MAX).....	39
9.4 Cuenta de valores de datos (COUNT)	41
9.5 Valores NULL y funciones de columna.....	43
9.6 Eliminación de filas duplicadas (DISTINCT)	44
9.7 Consultas agrupadas (GROUP BY).....	45
9.8 Condiciones de búsqueda de grupos (HAVING)	48
9.9 Restricciones en condiciones de búsqueda grupal.....	50
9.10 HAVING sin GROUP BY	51
10. SINTAXIS DE LA ORDEN SELECT	52
11. VISTAS	57

12. OPTIMIZACIÓN DE CONSULTAS.....	58
RESUMEN FINAL	63

RESUMEN INTRODUCTORIO

En esta unidad se estudiarán conceptos relacionados con la sintaxis y las sentencias del lenguaje SQL.

En este sentido, se analizan las instrucciones SELECT y FROM con diferentes consultas, la cláusula WHERE, las condiciones de búsqueda, que incluye Test de comparación, Test de rango, Test de pertenencia a conjunto, Test de correspondencia con patrón, Test de valor nulo y Condiciones de búsqueda compuestas.

Además, se explicará la importancia de las columnas calculadas, la selección de todas las columnas, las filas duplicadas a través de la cláusula DISTINCT y la ordenación y combinación de resultados de una consulta. Asimismo, se introduce el concepto de Vista y sus principales características. Además, se verá el uso de la sentencia EXPLAIN para la optimización de consultas.

Para terminar, se hará un resumen de diferentes consultas que involucran acciones muy utilizadas como SUM, MIN, MAX o COUNT, entre otras.

INTRODUCCIÓN

El lenguaje SQL es universal y multiusos. Tan importante es conocer su radio de acción para una factoría de desarrollo de software como para una organización dedicada a gestión de cobros, venta de billetes de tren o una ONG. Las consultas en SQL deben ser parte de la gestión del conocimiento de toda empresa cuyos empleados o parte de ellos dediquen sus labores a cotejar y gestionar datos e información sobre bases de datos. El potencial del lenguaje SQL abarca un elevado número de funciones, sin embargo, con un pequeño conjunto de ellas ya sería posible realizar acciones muy interesantes sobre todo tipo de bases de datos, independientemente de si son gestionadas por software libre o privativo.

En este sentido, la unidad pretende que el alumno centre su atención en la importancia que tiene el lenguaje SQL para el devenir de su futuro laboral, no solo en el aspecto puramente informático (desarrollador, administrador, tester, etc.) sino en cualquier otro perfil cuyas tareas diarias estén relacionadas con las bases de datos (administrativos, contables, recepcionistas, etc.). En la mayor parte de los casos, el propio software que utiliza el usuario ya implementa las consultas necesarias para realizar ciertas funciones, sin embargo, es posible que alguna de ellas deba realizarse directamente sobre la base de datos, sin la ayuda de este software. Para ello es

fundamental conocer SQL a nivel introductorio como bien se refleja en la presente unidad.

CASO INTRODUCTORIO

La empresa para la que trabajáis Paula y tú está pensando en abrir una tienda online para vender los artículos a través de Internet, y tu tarea es ayudar a la compañía a gestionar su base de datos de productos, clientes, proveedores, pedidos etc. Tu objetivo es aprender a realizar consultas SQL para obtener información relevante sobre las operaciones que se llevan a cabo mediante la tienda online.

Al finalizar la unidad sabrás identificar las principales características y ventajas del lenguaje SQL, conocerás las sentencias SQL, serás capaz de realizar consultas para obtener ciertos datos de la base de datos y realizar consultas anidadas para obtener datos que se encuentren en distintas tablas de la base de datos.

1. SINTAXIS Y SENTENCIAS DEL LENGUAJE SQL

Paula se ha reunido con los responsables de los diferentes departamentos de la empresa. Como es lógico, cada uno de ellos necesita una información concreta de la base de datos, y te han pedido a ti que te encargues de obtener toda esa información.

El lenguaje SQL consta de unas treinta sentencias, resumidas a continuación. Cada sentencia demanda una acción específica por parte del SGBD, tal como la creación de una nueva tabla, la recuperación de datos o la inserción de nuevos datos en la base de datos.

Operación	Sentencia	Descripción
Manipulación de datos	SELECT	Recupera datos de la BD
	INSERT	Añade nuevas filas de datos de la BD
	DELETE	Suprime filas de la BD
	UPDATE	Modifica datos existentes en la BD
Definición de datos	CREATE TABLE	Añade una base de datos a la BD
	DROP TABLE	Suprime una tabla de la BD
	ALTER TABLE	Modifica la estructura de una tabla existente
	CREATE VIEW	Añade una nueva vista a la BD
	DROP VIEW	Suprime una lista de la BD
	CREATE INDEX	Construye un índice para una columna
	DROP INDEX	Suprime el índice para una columna
	CREATE SYNONYM	Define un alias para un nombre de tabla
	DROP SYNONYM	Suprime un alias para un nombre de tabla
	COMMENT	Define comentarios para una tabla
	LABEL	Define el título de una columna
Control de acceso	GRANT	Concede privilegios de acceso a usuarios
	REVOKE	Suprime privilegios de acceso a usuarios
Control de transacciones	COMMIT	Finaliza la transacción actual
	ROLLBACK	Aborta la transacción actual

Tabla: Resumen de sentencias SQL

Todas las sentencias SQL comienzan con un verbo, una palabra clave que describe lo que la sentencia hace. CREATE, INSERT, UPDATE, DELETE y SELECT son verbos típicos. La sentencia continúa con una o más cláusulas. Una cláusula puede especificar los datos sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la

sentencia se supone que hace. Todas las cláusulas comienzan también con una palabra clave, tal como WHERE, FROM, INTO y HAVING.

Algunas cláusulas son opcionales; otras son necesarias. La estructura y contenido específicos varían de una cláusula a otra. Muchas contienen nombres de tablas o columnas; o pueden contener palabras claves adicionales, constantes o expresiones. Se debe evitar palabras clave al nombrar tablas y columnas.

ADA	CURRENT	GO	OF	SOME
ALL	CURSOR	GOTO	ON	SQL
AND	DEC	GRANT	OPEN	SQLCODE
ANY	DECIMAL	GROUP	OPTION	SQLERROR
AS	DECLARE	HAVING	OR	SUM
ASC	DEFAULT	IN	ORDER	TABLE
AUTHORIZATION	DELETE	INDICATOR	PASCAL	TO
AVG	DESC	INSERT	PLI	UNION
BEGIN	DISTINCT	INT	PRESICION	UNIQUE
BETWEEN	DOUBLE	IINTEGER	PRIMARY	UPDATE
BY	END	INTO	PRIVILEGES	USER
C	ESCAPE	IS	PROCEDURE	VALUES
CHAR	EXEC	KEY	PUBLIC	VIEW
CHARACTER	EXISTS	LANGUAGE	REAL	WHENEVER
CHECK	FETCH	LIKE	REFERENCES	WHERE
CLOSE	FLOAT	MAX	ROLLBACK	WITH
COBOL	FOR	MIN	SCHEMA	WORK
COMMIT	FOREIGN	MODULE	SECTION	

Tabla: Palabras clave de las sentencias SQL

1.1. Nombres

Los objetos de una base de datos basada en SQL se identifican asignándoles nombres únicos. Los nombres se utilizan en las sentencias SQL para identificar el objeto de la base de datos sobre la que la sentencia debe actuar. El estándar SQL especifica nombres de tabla (que identifican tablas), nombres de columna (que identifican columnas) y nombres de usuario (que identifican usuarios de la base de datos). Muchas implementaciones SQL contemplan objetos nominados adicionalmente, tales como procedimientos almacenados (Sybase y SQL Server), relaciones clave primaria/clave ajena (DB2) y formularios de entrada de datos (Ingres).

El estándar SQL especifica que los nombres deben contener de 1 a 18 caracteres, comenzar con una letra y que no pueden contener espacios o caracteres de puntuación

especiales. En la práctica los nombres contemplados por los productos SGBD basados en SQL varían significativamente. DB2, por ejemplo, restringe los nombres de usuario a ocho caracteres, pero permite 18 caracteres en los nombres de tablas y columnas. ORACLE permite nombre de tablas y columnas de 30 caracteres, y muchas otras implementaciones SQL también permiten nombres más largos.

Los diferentes productos también se diferencian en los caracteres especiales que permiten en los nombres de tablas. Para conseguir portabilidad es mejor mantener los nombres relativamente breves y evitar el uso de caracteres especiales.

Nombres de tabla (Entidades): Cuando se especifica un nombre de tabla en una sentencia SQL, se presupone que se refiere a una de las tablas propias (es decir, una tabla ya creada). Con el permiso adecuado, también se puede hacer referencia a tablas propiedad de otros usuarios, utilizando un nombre de tabla cualificado. Un nombre de tabla cualificado especifica el nombre del propietario de la tabla junto con el nombre de la tabla, separados por un punto (.).

Por ejemplo, la tabla CUMPLEAÑOS, propiedad del usuario SAM, tiene el nombre de tabla cualificado: SAM.CUMPLEAÑOS

Un nombre de tabla cualificado puede ser utilizado generalmente dentro de una sentencia SQL en cualquier lugar donde pueda aparecer un nombre de tabla.

Nombres de columna (Atributos): Cuando se especifica un nombre de columna en una sentencia SQL, SQL puede determinar normalmente a qué columna se refiere a partir del contexto. Sin embargo, si la sentencia afecta a dos columnas con el mismo nombre correspondientes a dos tablas diferentes, debe utilizarse un nombre de columna cualificado para identificar sin ambigüedad la columna designada. Un nombre de columna cualificado especifica tanto el nombre de la tabla que contiene la columna como el nombre de la columna, separados por un punto (.).

Por ejemplo, la columna de nombre VENTAS de la Tabla REPVENTAS tiene el nombre de columna cualificado: REPVENTAS. VENTAS

Si la columna procede de una tabla propiedad de otro usuario, se utiliza un nombre de tabla cualificado en el nombre de columna cualificado.

Por ejemplo, la columna DIA_MES de la Tabla CUMPLEAÑOS propiedad del usuario SAM se especifica mediante el nombre de columna cualificado completo: SAM.CUMPLEAÑOS.DIA_MES

Los nombres de columna de cualificados se pueden utilizar generalmente en una sentencia SQL en cualquier lugar en el que pueda aparecer un nombre de columna simple (no cualificado); las excepciones se indican en las descripciones de las sentencias SQL individuales.

1.2. Expresiones

Las expresiones se utilizan en el lenguaje SQL para calcular valores que se recuperan de una base de datos y para calcular valores utilizados en la búsqueda en la base de datos.



EJEMPLO PRÁCTICO

Esta consulta calcula las ventas de cada oficina como porcentaje de su objetivo.

Solución:

```
SELECT CIUDAD, OBJETIVO, VENTAS, (VENTAS/OBJETIVO) * 100  
FROM OFICINAS;
```



EJEMPLO PRÁCTICO

Esta consulta lista las ciudades de las oficinas cuyas ventas son superiores a 50.000€ por encima del objetivo:

Solución:

```
SELECT CIUDAD FROM OFICINAS  
WHERE VENTAS > OBJETIVO + 50000.00;
```

El estándar SQL ANSI/ISO especifica cuatro operaciones aritméticas que pueden ser utilizadas en expresiones: suma (X+Y), resta (X-Y), multiplicación (X*Y) y división (X/Y). También se pueden utilizar paréntesis para formar expresiones más complicadas, como la siguiente: (VENTAS * 1.05) – (OBJETIVO * 0.95)

Estrictamente hablando, los paréntesis no son necesarios en esta consulta, ya que el estándar especifica que la multiplicación y la división tienen una precedencia superior a la suma y la resta. Sin embargo, deberían utilizarse siempre los paréntesis para lograr que las expresiones no sean ambiguas, ya que diferentes dialectos de SQL pueden

utilizar reglas diferentes. Además, los paréntesis incrementan la legibilidad de la sentencia y hacen que las sentencias de SQL programado sean más fáciles de mantener.

El estándar también especifica conversión automática de tipos de datos desde números enteros a decimales, y desde números decimales a números en coma flotante, según se precise.

Por tanto, estos tipos de datos se pueden mezclar en una expresión numérica. Muchas implementaciones SQL incluyen otros operadores y permiten operaciones sobre datos de caracteres y de fechas. SQL, por ejemplo, incluye un operador concatenación de cadenas, escrito con el símbolo “+”.



EJEMPLO PRÁCTICO

Si dos columnas llamadas NOMBRE y APELLIDO contiene los valores “Jim” y “Jackson”, entonces la siguiente expresión produce la cadena “Mr./Mrs. Jim Jackson”.

Solución:

```
('Mr./Mrs. ' + NOMBRE + APELLIDO)
```

Como ya se ha mencionado, también permite la suma, y resta de datos DATE, TIME y TIMESTAMP, para aquellas ocasiones en las que estas operaciones tengan sentido.

1.3 Funciones internas

Es posible que en alguna ocasión interese localizar un registro que tome un valor en uno o varios campos desconocidos, pero que posee, por algún motivo, una propiedad interesante para el usuario.

Un claro ejemplo puede ser el deseo de conocer cuál es el artículo del que menos unidades existen, por ejemplo, para realizar el pedido con más urgencia o del que más unidades hay en el almacén para no realizarlo, o el número medio de unidades para cada artículo presentes en almacén.

SQL dispone de funciones integradas o funciones calculadas para resolver estas cuestiones. Las más habituales realizan los siguientes cálculos:

- **MAX** → Máximo de un conjunto de valores.

- **MIN** → Mínimo de un conjunto de valores.
- **AVG** → Media de un conjunto de valores.
- **SUM** → Suma total de un conjunto de valores.
- **COUNT** → Cuenta el número de valores que hay en un conjunto.



PARA SABER MÁS

En este enlace se recogen las cláusulas y ejemplos con SQL:



1.4 Otras funciones en SQL

Entre otras funciones en SQL, podemos destacar:

Función	Descripción
CURRENT_TIMESTAMP ()	Fecha y hora actuales.
LOWER (cadena)	Convierte una cadena de tipo varchar a minúsculas.
SUBSTRING (fuente, n, lon)	Extrae de la cadena <i>fuente</i> una subcadena, comenzando en el carácter n-ésimo, con una longitud lon
UPPER (cadena)	Convierte una cadena de tipo varchar a mayúsculas.
LEFT(cadena, lon)	Devuelve de la cadena una subcadena, comenzando por la izquierda y con una longitud lon.
LEN(cadena)	Devuelve la longitud de la cadena.
LTRIM(cadena)	Quita los blancos de la izquierda en la cadena.
RTRIM(cadena)	Quita los blancos de la derecha en la cadena.
STR(cadena)	Devuelve la cadena alineada a la derecha.
RIGHT(cadena, num_elem)	Devuelve el número de elementos de la cadena que están a la derecha. RIGHT(CASA,2) = SA
ASCII(cadena)	Devuelve el nº ASCII correspondiente a la cadena.

Tabla: Otras funciones en SQL



ENLACE DE INTERÉS

Lee más sobre las diferentes funciones en SQL en este enlace:



2. CLÁUSULA SELECT

Ya que eres el responsable de obtener la información para los diferentes departamentos de la empresa, Paula te ha asignado a un ayudante al que debes explicar cómo se obtiene y se muestra la información de una base de datos SQL.

La cláusula SELECT especifica los ítems de datos a recuperar por la consulta. Los ítems se especifican generalmente mediante separados por comas. Cada ítem de selección de la lista genera una única columna de resultados de consulta, en orden de izquierda a derecha.

Un ítem de selección puede ser:

- Un nombre de columna, identificando una columna de la tabla designada en la cláusula FROM. Cuando un nombre de columna aparece como ítem de selección, SQL simplemente toma el valor de esa columna de cada fila de la tabla de base de datos y lo coloca en la fila correspondiente de los resultados de la consulta.
- Una constante, especificando que el mismo valor constante va a aparecer en todas las filas de los resultados de la consulta.
- Una expresión SQL, indicando que SQL debe calcular el valor a colocar en los resultados, según el estilo especificado por la expresión.

2.1 Cláusula FROM

La cláusula FROM consta de la palabra clave FROM, seguida de una lista de especificaciones de tablas separadas por comas. Cada especificación de tabla identifica una tabla que contiene los datos a recuperar por la consulta. Estas tablas se denominan tablas fuente de la consulta (y de la sentencia SELECT), ya que constituyen la fuente de todos los datos que aparecen en los resultados. Todas las consultas de este capítulo tienen una única tabla fuente y todas las cláusulas FROM contienen un solo nombre de tabla.



ENLACE DE INTERÉS

Conoce más sobre las cláusulas SELECT y FROM en SQL:



2.2 Resultados de consultas

El resultado de una consulta SQL es siempre una tabla de datos, semejante a las tablas de la base de datos. Si se escribe una sentencia SELECT utilizando SQL interactivo, el SGBD visualizará los resultados de la consulta en forma tabular sobre la pantalla de la computadora. Generalmente los resultados de la consulta formarán una tabla con varias columnas y varias filas.



EJEMPLO PRÁCTICO

Por ejemplo, se va a realizar una consulta con los nombres, oficinas y fecha de contrato de todos los vendedores. Esta consulta produce una tabla de tres columnas (ya que pide tres ítems de datos) y diez filas (ya que hay diez vendedores):

Solución: `SELECT NOMBRE, OFICINA_REP, CONTRATO FROM REPVENTAS`

NOMBRE	OFICINA_REP	CONTRATO
Bill Adams	13	12-FEB-88
Mary Jones	11	12-OCT-89
Sue Smith	21	10-DIC-86
Sam Clark	11	14-JUN-88
Bob Smith	12	19-MAY-87
Dan Roberts	12	20-OCT-86
Tom Snyder	NULL	13-ENE-90
Larry Fitch	21	12-OCT-89
Paul Cruz	12	01-MAR-87
Nancy Angelli	22	14-NOV-88

Las consultas SQL más sencillas solicitan columnas de datos de una única tabla en la base de datos.



EJEMPLO PRÁCTICO

Por ejemplo, la lista de la población, región y ventas de cada oficina:

Solución: `SELECT CIUDAD, REGION, VENTAS FROM OFICINAS`

CIUDAD	REGION	VENTAS
Denver	Oeste	\$186,042.00
New York	Este	\$692,637.00
Chicago	Este	\$735,042.00
Atlanta	Este	\$367,911.00
Los Ángeles	Oeste	\$835,915.00




VÍDEO DE INTERÉS

En este vídeo se puede ver el funcionamiento de SQL para la realización de consultas:



Para consultas sencillas como ésta, `SELECT` sólo incluye las dos cláusulas imprescindibles. La cláusula `SELECT` designa a las columnas solicitadas y la cláusula `FROM` designa a la tabla que las contiene. Conceptualmente, SQL procesa la consulta recorriendo la tabla nominada en la cláusula `FROM`, una fila cada vez. Por cada fila, SQL toma los valores de las columnas solicitadas en la lista de selección y produce una única fila de resultados. Los resultados contienen por tanto una fila de datos por cada fila de la tabla.

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS
22	Denver	Oeste	\$300,000.00	\$186,042.00
11	New York	Este	\$575,000.00	\$692,637.00
12	Chicago	Este	\$800,000.00	\$735,042.00
13	Atlanta	Este	\$350,000.00	\$367,911.00
21	Los Angeles	Oeste	\$725,000.00	\$835,915.00



CIUDAD	REGION	VENTAS
Denver	Oeste	\$186,042.00
New York	Este	\$692,637.00
Chicago	Este	\$735,042.00
Atlanta	Este	\$367,911.00
Los Angeles	Oeste	\$835,915.00

Ejemplo de consulta con cláusulas FROM y SELECT

2.3 Cláusula WHERE

Las consultas SQL que recuperan todas las filas de una tabla son útiles para inspección y elaboración de informes sobre la base de datos, pero para poco más. Generalmente se deseará seleccionar solamente parte de las filas de una tabla, y sólo incluirán esas filas en los resultados. La cláusula WHERE se emplea para especificar las filas que se desean recuperar.



EJEMPLO PRÁCTICO

Muestra las oficinas en donde las ventas exceden del objetivo.

Solución: `SELECT CIUDAD, VENTAS, OBJETIVO FROM OFICINAS
WHERE VENTAS > OBJETIVO`

CIUDAD	VENTAS	OBJETIVO
New York	\$692,637.00	\$575,000.00
Atlanta	\$367,911.00	\$350,000.00
Los Ángeles	\$835,915.00	\$725,000.00



EJEMPLO PRÁCTICO

Muestra los empleados dirigidos por Bob Smith (empleado 104).

Solución: `SELECT NOMBRE, VENTAS FROM REPVENTAS
WHERE DIRECTOR = 104`

NOMBRE	VENTAS
Bill Adams	\$367,911.00
Dan Roberts	\$305,673.00
Paul Cruz	\$286,775.00

La cláusula WHERE consta de la palabra clave WHERE seguida de una condición de búsqueda que especifica las filas a recuperar. En la consulta anterior, por ejemplo, la condición de búsqueda es `DIRECTOR = 104`. Conceptualmente, SQL recorre cada fila de la tabla REPVENTAS, una a una, y aplica la condición de búsqueda a la fila. Cuando aparece un nombre de columna en la condición de búsqueda a la fila (tal como la columna DIRECTOR en este ejemplo), SQL utiliza el valor de la columna en la fila actual. Por cada fila, la condición de búsqueda puede producir uno de los tres resultados.

- Si la condición de búsqueda es TRUE (cierta), la fila se incluye en los resultados de la consulta. Por ejemplo, la fila correspondiente a Bill Adams tiene el valor DIRECTOR correcto y por tanto se incluye.
- Si la condición de búsqueda es FALSE (falsa), la fila se excluye de los resultados de la consulta.
- Si la condición de búsqueda tiene un valor NULL (desconocido), la fila se excluye de los resultados de la consulta.

La condición de búsqueda actúa como un filtro para las filas de la tabla. Las filas que satisfacen la condición de búsqueda atraviesan el filtro y forman parte de los resultados de la consulta. Las filas que no satisfacen la condición de búsqueda son atrapadas por el filtro y quedan excluidas de los resultados de la consulta.

3. CONDICIONES DE BÚSQUEDA

La información que te piden los compañeros de los diferentes departamentos es cada vez más específica y con más condicionantes, por lo que te dispones a investigar la mejor manera de conseguir la información deseada mediante el uso del lenguaje SQL.

SQL ofrece un rico conjunto de condiciones de búsqueda que permite especificar muchos tipos diferentes de consultas. A continuación, se resumen 5 condiciones básicas de búsqueda (llamadas predicados en el estándar ANSI/ISO):

3.1 Test de comparación

La condición de búsqueda más utilizada en una consulta SQL es el test de comparación. En un test de comparación, SQL calcula y compara los valores de dos expresiones SQL por cada fila de datos. Las expresiones pueden ser tan simples como un nombre de columna o una constante, o pueden ser expresiones aritméticas más complejas. Los operadores son: =, <>, <, <=, >, >=.



EJEMPLO PRÁCTICO

Halla los vendedores contratados antes de 1988.

Solución: `SELECT NOMBRE FROM REPVENTAS
WHERE CONTRATO < '01-ENE-88'`

NOMBRE
Sue Smith
Bob Smith
Dan Roberts
Paul Cruz

La comparación de desigualdad se escribe como $A \neq B$. Varias implementaciones SQL utilizan notaciones alternativas, tales como $A \neq B$. Al comparar los valores de dos expresiones en el test de comparación se pueden producir tres resultados:

- Si la comparación es cierta, el test produce un resultado TRUE.
- Si la comparación es falsa, el test produce un resultado FALSE.

- Si alguna de las dos expresiones produce un valor NULL, la comparación genera un resultado NULL.

Recuperación de una fila: El test de comparación más habitual es el que comprueba si el valor de una columna es igual a cierta constante. Cuando la columna es una clave primaria, el test aísla una sola fila de la tabla, produciendo una sola fila de resultados.



EJEMPLO PRÁCTICO

Recupera el nombre y el límite de crédito del cliente número 2.107.

Solución: `SELECT EMPRESA, LIMITE_CREDITO FROM CLIENTES
WHERE NUM_CLIE = 2107`

EMPRESA	LIMITE_CREDITO
Ace International	\$35,000.00

Este tipo de consulta es el fundamento de los programas de recuperación de base de datos basadas en formularios. El usuario introduce el número de cliente en el formulario, y el programa utiliza el número para construir y ejecutar una consulta. Luego visualiza los datos recuperados en el formulario.



EJEMPLO PRÁCTICO

Lista vendedores que superan sus cuotas.

Solución: `SELECT NOMBRE FROM REPVENTAS WHERE VENTAS <= CUOTA`

NOMBRE
Bob Smith
Nancy Angelli

Consideraciones del valor NULL: El comportamiento de los valores NULL en los test de comparación pueden revelar que algunas nociones obviamente ciertas referentes a consultas SQL no son, de hecho, necesariamente ciertas.

Deberían incluir cada fila de la Tabla REPVENTAS, pero las consultas producen siete y dos filas, respectivamente, haciendo un total de nueve filas, pero hay diez filas en la Tabla REPVENTAS. La fila de Tom Snyder tiene un valor NULL en la columna CUOTA, puesto que aún no se le ha asignado una cuota. Esta fila no aparece en ninguna de las consultas, desaparece en el test de comparación.

Como muestra este ejemplo, es necesario considerar la gestión del valor NULL cuando se especifica una condición de búsqueda. En la lógica trivaluada de SQL, una condición de búsqueda puede producir un resultado TRUE, FALSE o NULL. Sólo las filas en donde la condición de búsqueda genera un resultado TRUE se incluyen los resultados de la consulta.

3.2 Test de rango

SQL proporciona una forma diferente de condición de búsqueda con el test de rango (BETWEEN).



EJEMPLO PRÁCTICO

El siguiente ejemplo muestra un test de rango típico.

Solución: `SELECT NUM_PEDIDO, FECHA_PEDIDO, FAB, PRODUCTO, IMPORTE FROM PEDIDOS WHERE FECHA_PEDIDO BETWEEN '01-OCT-89' AND '31-DIC-89'`

NUM_PEDIDO	FECHA_PEDIDO	FAB	PRODUCTO	IMPORTE
112961	17-DIC-89	REI	A244L	\$31,500.00
112968	12-OCT-89	ACI	41004	\$3,978.00
112963	17-DIC-89	ACI	41004	\$3,276.00
112983	27-DIC-89	ACI	41004	\$702.00
112979	12-OCT-89	ACI	41002	\$15,000.00
112992	04-NOV-89	ACI	41002	\$760.00
112975	12-OCT-89	REI	A244G	\$2,100.00
112987	31-DIC-89	ACI	4100Y	\$27,500.00

El test de rango comprueba si un valor de dato se encuentra entre dos valores especificados. Implica el uso de tres expresiones SQL. La primera expresión define el valor a comprobar; las expresiones segunda y tercera definen los extremos superior e inferior del rango a comprobar. Los tipos de datos de las tres expresiones deben ser comparables.

El test BETWEEN incluye los puntos extremos del rango, por lo que los pedidos remitidos el 1-OCT o el 31-DIC se incluyen en los resultados de la consulta.

La versión negada del test de rango (NOT BETWEEN) comprueba los valores que caen fuera del rango.



EJEMPLO PRÁCTICO

Lista los vendedores cuyas ventas no están entre el 80 y el 120% de su cuota.

Solución: `SELECT NOMBRE, VENTAS, CUOTA FROM REPVENTAS
WHERE VENTAS NOT BETWEEN (0.8 * CUOTA) AND (1.2 * CUOTA)`

NOMBRE	VENTAS	CUOTA
Mary Jones	\$392,725.00	\$300,000.00
Sue Smith	\$474,050.00	\$350,000.00
Bob Smith	\$142,594.00	\$200,000.00
Nancy Angelli	\$186,042.00	\$300,000.00

La expresión de test especificada en el test BETWEEN puede ser cualquier expresión válida, pero en la práctica generalmente es tan sólo un nombre de columna, como en los ejemplos anteriores.

- Si la expresión de test produce un valor NULL, o si ambas expresiones definitorias del rango producen valores NULL, el test BETWEEN devuelve un resultado NULL.
- Si la expresión que define el extremo inferior del rango produce un valor NULL, el test BETWEEN devuelve FALSE si el valor de test es superior al límite superior, y NULL en caso contrario.

- Si la expresión que define el extremo superior del rango produce un valor NULL, el test BETWEEN devuelve FALSE si el valor de test es menor que el límite inferior, y NULL en caso contrario.

El test BETWEEN no añade realmente potencia expresiva a SQL, ya que puede ser expresado mediante dos test de comparación. El test de rango: A BETWEEN B AND C es completamente equivalente a: (A >= B) AND (A <= C).

3.3 Test de pertenencia a conjunto

Examina si un valor de dato coincide con uno de una lista de valores objetivo.



EJEMPLO PRÁCTICO

Lista los vendedores que trabajan en New York, Atlanta o Denver.

Solución: `SELECT NOMBRE, VENTAS, CUOTA
FROM REPVENTAS WHERE OFICINA_REP IN (11, 13, 22)`

NOMBRE	VENTAS	CUOTA
Bill Adams	\$367,911.00	\$350,000.00
Mary Jones	\$392,725.00	\$300,000.00
Sam Clark	\$299,912.00	\$275,000.00
Nancy Angelli	\$186,042.00	\$300,000.00

Se puede comprobar si el valor del dato no corresponde a ninguno de los valores objetivos utilizando la forma NOT IN del test de pertenencia a conjunto. La expresión de test en un test IN puede ser cualquier expresión SQL, pero generalmente es tan sólo un nombre de columna, como en los ejemplos precedentes.

Si la expresión de test produce un valor NULL, el test IN devuelve NULL. Todos los elementos en la lista de valores objetivo deben tener el mismo tipo de datos, y ese tipo debe ser comparable al tipo de dato de la expresión de test.

Al igual que el test BETWEEN, el test IN no añade potencia expresiva a SQL, ya que la condición de búsqueda: X IN (A, B, C) es completamente equivalente a: (X = A) OR (X = B) OR (X = C)

3.4. Test de correspondencia con patrón

Se puede utilizar un test de comparación simple para recuperar las filas en donde el contenido de una consulta de texto se corresponde con un cierto texto particular. Sin embargo, se podría olvidar fácilmente si el nombre de la empresa era “Smith”, “Smithson” o “Smithsonian”.

El test de correspondencia con patrón de SQL puede ser utilizado para recuperar los datos sobre la base de una correspondencia parcial del nombre de cliente. El test de correspondencia con patrón (LIKE), comprueba si el valor de datos de una columna se ajusta a un patrón especificado. El patrón es una cadena que puede incluir uno o más caracteres comodines. Estos caracteres se interpretan de una manera especial.

Caracteres comodines: El carácter comodín signo de porcentaje (%) se corresponde con cualquier secuencia de cero o más caracteres. Se debe reseñar que en algunos SGBD este mismo carácter comodín es el asterisco (*).



EJEMPLO PRÁCTICO

Muestra el límite de crédito de Smithson Corp.

Solución: `SELECT EMPRESA FROM CLIENTES
WHERE EMPRESA LIKE 'Smith% Corp.'`

La palabra clave LIKE dice a SQL que compare la columna NOMBRE con el patrón “Smith% Corp.”. Cualquiera de los nombres siguientes se ajustaría al patrón:

EMPRESA
Smith Corp.
Smithson Corp.
Smithsen Corp.
Smithsonian Corp.

El carácter comodín subrayado (_) se corresponde con cualquier carácter simple, aunque en algunos SGBD, este mismo carácter viene determinado por el signo de interrogación (?) en lugar del (_).



EJEMPLO PRÁCTICO

Si se está seguro de que el nombre de la empresa es o bien “Smithson” o bien “Smithsen”, por ejemplo, se puede utilizar esta consulta.

```
Solución: SELECT EMPRESA, LIMITE_CREDITO  
FROM CLIENTES WHERE EMPRESA LIKE 'Smiths_n Corp.'
```

Los caracteres comodines pueden aparecer en cualquier lugar de la cadena patrón, y puede haber varios caracteres comodines dentro de una misma cadena.

Se pueden localizar cadenas que no se ajusten a un patrón utilizando el formato NOT LIKE del test de correspondencia de patrones. El test LIKE debe aplicarse a una columna con un tipo de datos cadena. Si el valor del dato en la columna es NULL, el test LIKE devuelve un resultado NULL.

Caracteres escape: Uno de los problemas de la correspondencia con patrones en cadenas es cómo hacer corresponder los propios caracteres comodines como caracteres literales. Para comprobar la presencia de un carácter tanto por ciento en una columna de datos de texto, por ejemplo, no se puede simplemente incluir el signo del tanto por cien en el patrón, ya que SQL lo trataría como un comodín.

El estándar SQL especifica una manera de comparar literalmente caracteres comodines, utilizando un carácter escape especial (\). Cuando el carácter escape aparece en el patrón, el carácter que le sigue inmediatamente se trata como un carácter literal en lugar de como un carácter comodín. El carácter escapado puede ser uno de los dos caracteres comodines, o el propio carácter de escape, que ha tomado ahora un significado especial dentro del patrón.



EJEMPLO PRÁCTICO

Devuelve los clientes cuya dirección incluya el símbolo del porcentaje

```
Solución: SELECT * FROM CLIENTES WHERE DIRECCION LIKE '%\%%'
```


El primer signo de porcentaje en el patrón, que sigue a un carácter escape, es tratado como un signo literal; el segundo funciona como un comodín.

3.5. Test de valor nulo

Los valores NULL crean una lógica trivaluada para las condiciones de búsqueda en SQL. Para una fila determinada, el resultado de una condición de búsqueda puede ser TRUE o FALSE, o puede ser NULL debido a que una de las columnas utilizadas en la evaluación de la condición de búsqueda contiene un valor NULL. A veces es útil comprobar explícitamente los valores NULL en una condición de búsqueda y gestionarlos directamente. SQL proporciona un test especial de valor nulo (NULL).



EJEMPLO PRÁCTICO

Halla el vendedor que aún no tiene asignada una oficina.

Solución: `SELECT NOMBRE FROM REPVENTAS
WHERE OFICINA_REP IS NULL`

La forma negada del test de valor nulo (IS NOT NULL) encuentra las filas que no contienen un valor NULL.

A diferencia de las condiciones de búsqueda descritas anteriormente, el test de valor nulo no puede producir un resultado NULL. Será siempre TRUE o FALSE.



EJEMPLO PRÁCTICO

Lista los vendedores a los que se les ha asignado una oficina.

Solución: `SELECT NOMBRE FROM REPVENTAS
WHERE OFICINA_REP IS NOT NULL`

3.6. Condiciones de búsqueda compuestas

Las condiciones de búsqueda simples, descritas en las secciones precedentes devuelven un valor TRUE, FALSE o NULL cuando se aplican a una fila de datos.

Utilizando las reglas de la lógica, se pueden combinar estas condiciones de búsqueda SQL simples para formar otras más complejas.

La palabra clave OR se utiliza para combinar dos condiciones de búsqueda cuando una o la otra (o ambas) deban ser ciertas.



EJEMPLO PRÁCTICO

Halla los vendedores que están por debajo de la cuota o con ventas inferiores a \$300.000.

Solución: `SELECT NOMBRE, CUOTA, VENTAS
FROM REPVENTAS WHERE VENTAS < CUOTA OR VENTAS < 300000.00`



EJEMPLO PRÁCTICO

También se puede utilizar la palabra clave AND para combinar dos condiciones de búsqueda que deban ser ciertas simultáneamente.

Solución: `SELECT NOMBRE, CUOTA, VENTAS
FROM REPVENTAS WHERE VENTAS < CUOTA AND VENTAS < 300000.00`

Finalmente, se puede utilizar la palabra clave NOT para seleccionar filas en donde la condición de búsqueda es falsa.



EJEMPLO PRÁCTICO

Halla todos los vendedores que están por debajo de la cuota, pero cuyas ventas no son inferiores a \$150.000.

Solución: `SELECT NOMBRE, CUOTA, VENTAS FROM REPVENTAS WHERE VENTAS < CUOTA AND NOT VENTAS < 150000.00`

Utilizando las palabras clave AND, OR y NOT y los paréntesis para agrupar los criterios de búsqueda, se pueden construir criterios de búsqueda muy complejos. Cuando se combinan más de dos condiciones de búsqueda con AND, OR y NOT, el estándar especifica que NOT tiene la precedencia más alta, seguido de AND y por último OR. Para asegurar la portabilidad, es siempre una buena idea utilizar paréntesis y suprimir cualquier posible ambigüedad.

4. COLUMNAS CALCULADAS

El jefe del departamento de contabilidad de la empresa para la que trabajas requiere de una información que no está almacenada como tal en la base de datos. Para conseguir los datos tal y como te los piden, deberás hacer uso de columnas calculadas mediante las operaciones o funciones correspondientes.

Además de las columnas cuyos valores provienen directamente de la base de datos, una consulta SQL puede incluir columnas calculadas cuyos valores se calculan a partir de los valores de los datos almacenados. Para solicitar una columna calculada, se especifica una expresión SQL en la lista de selección. Las expresiones SQL pueden contener sumas, restas, multiplicaciones y divisiones.

También se pueden utilizar paréntesis para construir expresiones más complejas. Naturalmente las columnas referenciadas en una expresión aritmética deben tener un tipo numérico. Si se intenta sumar, restar, multiplicar o dividir columnas que contienen datos de texto, SQL reportará un error. Esta consulta muestra una columna calculada simple:



EJEMPLO PRÁCTICO

Lista la ciudad, la región y el importe por encima o por debajo del objetivo para cada oficina.

Solución: `SELECT CIUDAD, REGION, (VENTAS-OBJETIVO)`
`FROM OFICINAS`

CIUDAD	REGION	(VENTAS – OBJETIVO)
Denver	Oeste	-\$113,958.00
New York	Este	\$117,637.00
Chicago	Este	-\$64,958.00
Atlanta	Este	\$17,911.00
Los Ángeles	Oeste	\$110,915.00

Para procesar la consulta, SQL examina las oficinas, generando una fila de resultados por cada fila de la Tabla OFICINAS. Las dos primeras columnas de resultados provienen directamente de la Tabla OFICINAS. La tercera columna de los resultados se calcula, fila a fila, utilizando los valores de datos de la fila actual de la Tabla OFICINAS. Muchos productos SQL disponen de operaciones aritméticas adicionales, operaciones de cadenas de caracteres y funciones internas que pueden ser utilizadas en expresiones SQL. Estas pueden aparecer en expresiones de la lista de selección.



EJEMPLO PRÁCTICO

Lista el nombre, el mes y el año de contrato para cada vendedor.

Solución: `SELECT NOMBRE, MONTH (CONTRATO), YEAR (CONTRATO)`
`FROM REPVENTAS`

También se pueden utilizar constantes SQL por sí mismas como ítems en una lista de selección. Esto puede ser útil para producir resultados que sean más fáciles de leer e interpretar.



EJEMPLO PRÁCTICO

Lista las ventas para cada ciudad.

Solución: `SELECT CIUDAD, 'tiene ventas de', VENTAS
FROM OFICINAS`

CIUDAD	TIENE VENTAS DE	VENTAS
Denver	tiene ventas de	\$186,042.00
New York	tiene ventas de	\$692,637.00
Chicago	tiene ventas de	\$735,042.00
Atlanta	tiene ventas de	\$367,911.00
Los Ángeles	tiene ventas de	\$835,915.00

Los resultados de la consulta parecen consistir en una frase distinta por cada oficina, pero realmente es una tabla de tres columnas. Las columnas primera y tercera contienen valores procedentes de la Tabla OFICINAS. La columna siempre contiene la misma cadena de texto de quince caracteres.

5. SELECCIÓN DE TODAS LAS COLUMNAS

Algunas de las tablas de la base de datos tienen muchos campos, lo que hace que, aunque la consulta sea fácil de hacer, resulte muy tedioso hacerla ya que tienes que especificar todos y cada uno de los campos de la misma.

A veces es conveniente visualizar el contenido de todas las columnas de una tabla. Esto puede ser particularmente útil cuando uno va a utilizar por primera vez una base de datos y desea obtener una rápida comprensión de su estructura y de los datos que contiene.

Por conveniencia, SQL permite utilizar un asterisco (*) en lugar de la lista de selección como abreviatura de “todas las columnas”.



EJEMPLO PRÁCTICO

Muestra todos los datos de la Tabla OFICINAS.

Solución: `SELECT * FROM OFICINAS`

El resultado de la consulta contiene las seis columnas de la Tabla OFICINAS, en el mismo orden de izquierda a derecha que tienen en la tabla.

La selección de todas las columnas es muy adecuada cuando se utiliza SQL interactivo de forma casual. Debería evitarse en SQL programado, ya que cambios en la estructura de la base de datos pueden hacer fallar al programa.

6. FILAS DUPLICADAS (DISTINCT)

Al realizar algunas de las consultas que te han pedido, te das cuenta de que, por algún motivo, están apareciendo registros repetidos, por lo que la información obtenida es redundante y debes encontrar la forma de que la consulta SQL omita esas filas repetidas.

Si una consulta incluye la clave primaria de una tabla en su lista de selección, entonces cada fila de resultados será única (ya que la clave primaria tiene un valor diferente en cada fila). Si no se incluye la clave primaria en los resultados, pueden producirse filas duplicadas. Por ejemplo, se supone que se hace la siguiente petición:



EJEMPLO PRÁCTICO

Lista los números de empleados de todos los directores de oficinas de ventas.

```
SELECT DIR FROM OFICINAS
```

DIR
108
106
104
105
108

Los resultados tienen cinco filas (uno por cada oficina), pero dos de ellas son duplicados exactos la una de la otra. Porque Larry Fitch dirige las oficinas tanto de Los Ángeles como de Denver y su número de empleado (108) aparece en ambas filas de la Tabla OFICINAS. Estos resultados no son probablemente lo que se pretende cuando se hace la consulta. Si hubiera cuatro directores diferentes, cabría esperar que sólo aparecieran en los resultados cuatro números de empleado. Se pueden eliminar las filas duplicadas de los resultados de la consulta insertando la palabra clave **DISTINCT** en la sentencia **SELECT** justo antes de la lista de selección: **SELECT DISTINCT DIR FROM OFICINAS**.

Conceptualmente, SQL efectúa esta consulta generando primero un conjunto completo de resultados (cinco filas) y eliminando luego las filas que son duplicados exactos de alguna otra para formar los resultados finales. La palabra clave **DISTINCT** puede ser especificada con independencia de los contenidos de la lista **SELECT**. También se puede especificar la palabra clave **ALL** para indicar explícitamente que las filas duplicadas sean incluidas, pero es innecesario ya que este es el comportamiento por omisión.

7. ORDENACIÓN DE RESULTADOS DE UNA CONSULTA

El departamento de marketing quiere hacer una campaña especial para intentar promocionar los artículos con menos ventas. Para ello, te planteas que, probablemente,

deberás hacer una consulta que ordene los productos de menor a mayor en cuanto a las ventas se refiere.

Al igual que las filas de una tabla en la base de datos, las filas de los resultados de una consulta no están dispuestas en ningún orden particular. Se puede pedir a SQL que ordene los resultados de una consulta incluyendo la cláusula ORDER BY en la sentencia SELECT. La cláusula ORDER BY consta de las palabras claves ORDER BY, seguidas de una lista de especificaciones de ordenación separadas por comas.



EJEMPLO PRÁCTICO

Muestra las ventas de cada oficina, ordenadas en orden alfabético por región y dentro de cada región por ciudad.

Solución: `SELECT CIUDAD, REGION, VENTAS FROM OFICINAS
ORDER BY REGION, CIUDAD`

El resultado de la consulta contiene las seis columnas de la Tabla OFICINAS, en el mismo orden de izquierda a derecha que tienen en la tabla.

La primera especificación de ordenación (REGION) es la clave de la ordenación mayor, las que le sigan (CIUDAD, en este caso) son progresivamente claves de ordenación menores, utilizadas para “desempatar” cuando dos filas de resultados tienen los mismos valores para las claves mayores.

Utilizando la cláusula ORDER BY se puede solicitar la ordenación en secuencia ascendente o descendente, y se puede ordenar con respecto a cualquier elemento en la lista de selección de la consulta.

Por omisión, SQL ordena los datos en secuencia ascendente. Para solicitar ordenación en secuencia descendente, se incluye la palabra clave DESC en la especificación de ordenación.



EJEMPLO PRÁCTICO

Lista las oficinas, clasificadas en orden descendente de ventas, de modo que las oficinas con mayores aparezcan en primer lugar.

Solución: `SELECT CIUDAD, REGION, VENTAS
FROM OFICINAS ORDER BY VENTAS DESC`

CIUDAD	REGION	VENTAS
Los Ángeles	Oeste	\$835,915.00
Chicago	Este	\$735,042.00
New York	Este	\$692,637.00
Atlanta	Este	\$367,911.00
Denver	Oeste	\$186,042.00

También se puede utilizar la palabra clave ASC para especificar el orden ascendente, pero puesto que ésta es la secuencia de ordenación por omisión, la palabra clave se suele omitir.

Si la columna de resultados de la consulta utilizada para ordenación es una columna calculada, no tiene nombre de columna que se pueda emplear en una especificación de ordenación. En este caso, debe especificarse un número de columna en lugar de un nombre.



EJEMPLO PRÁCTICO

Lista las oficinas, clasificadas en orden descendente de rendimiento de ventas de modo que las oficinas con mejor rendimiento aparezcan primero.

Solución: `SELECT CIUDAD, REGION, (VENTAS-OBJETIVO)
FROM OFICINAS ORDER BY 3 DESC`

Estos resultados están ordenados por la tercera columna, que es la diferencia calculada entre VENTAS y OBJETIVO para cada oficina.

Las consultas de tabla única son generalmente sencillas y normalmente es fácil entender su significado tan sólo leyendo la sentencia SELECT. Para generar los resultados de una consulta correspondiente a una sentencia SELECT:

- Comenzar con la tabla designada en la cláusula FROM.
- Si hay cláusula WHERE, aplicar su condición de búsqueda a cada fila de la tabla, reteniendo aquellas filas para las cuales la condición de búsqueda es TRUE, y descartando aquéllas para las cuáles es FALSE o NULL.
- Para cada fila restante, calcular el valor de cada elemento en la lista de selección para producir una única fila de resultados. Por cada referencia de columna, utilizar el valor de la columna en la fila actual.
- Si se especifica SELECT DISTINCT, eliminar las filas duplicadas de los resultados que se hubieran producido.
- Si hay una cláusula ORDER BY, ordenar los resultados de la consulta según se especifiquen.

Las filas generadas por este procedimiento forman los resultados de la consulta.

8. COMBINACIÓN DE RESULTADOS DE UNA CONSULTA

Según avanzas con las consultas, algunas te están dando problemas ya que no consigues obtener toda la información deseada. Paula se da cuenta de esto y te dice que la solución más fácil es hacer dos consultas y combinar el resultado de las mismas.

Ocasionalmente, es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL permite esta capacidad gracias a la característica UNION de la sentencia SELECT.



EJEMPLO PRÁCTICO

Listar todos los productos donde el precio del producto exceda de 2000€ o donde más de 30000€ del producto hayan sido incluidos en un solo pedido. La operación UNION produce una única tabla de resultados que combina las filas de la primera consulta con las filas de los resultados de la segunda consulta. La sentencia SELECT que especifica la operación UNION tiene el siguiente aspecto:

Solución:

```
SELECT ID_FAB, ID_PRODUCTO FROM PRODUCTOS WHERE PRECIO > 2000.00  
UNION SELECT DISTINCT FAB, PRODUCTO FROM PEDIDOS  
WHERE IMPORTE > 30000.00
```

Hay varias restricciones sobre las tablas que pueden combinarse con una operación UNION:

- Ambas tablas deben contener el mismo número de columnas.
- El tipo de datos de cada columna en la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda tabla.
- Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY. Sin embargo, los resultados combinados pueden ser ordenados, según se describe en la sección siguiente.

Los nombres de columna de las dos consultas combinadas mediante una UNION no tienen que ser idénticos. En el ejemplo anterior, la primera tabla de resultados tenía columnas de nombre ID_FAB e ID_PRODUCTO, mientras que la segunda tabla de resultados tenía columnas de nombres FAB y PRODUCTOS. Puesto que las columnas de las dos tablas pueden tener nombres diferentes, las columnas de los resultados producidos por la operación UNION están sin designar.

Solamente permite nombres de columna o una especificación de todas las columnas (SELECT *) en la lista de selección, y prohíbe las expresiones en la lista de selección. Por omisión, la operación UNION elimina las filas duplicadas como parte de su procesamiento. La eliminación de filas duplicadas en los resultados de la consulta es un proceso que consume mucho tiempo, especialmente si los resultados contienen un gran número de filas. Si se sabe, en base a las consultas individuales implicadas, que la operación UNION no puede producir filas duplicadas, se debería utilizar específicamente la operación UNION ALL, ya que la consulta se ejecutará mucho más rápidamente.

8.1 Uniones y ordenación

La cláusula ORDER BY no puede aparecer en ninguna de las dos sentencias SELECT combinadas por una operación UNION. No tendría mucho sentido ordenar los dos conjuntos de resultados de ninguna manera, ya que éstos se dirigen directamente a la operación UNION y nunca son visibles al usuario. Sin embargo, el conjunto combinado de los resultados de la consulta producidos por la operación UNION puede ser ordenado especificando una cláusula ORDER BY después de la segunda sentencia SELECT. Ya que las columnas producidas por la operación UNION no tienen nombres, la cláusula ORDER BY debe especificar las columnas por número.



EJEMPLO PRÁCTICO

He aquí la misma consulta de productos con los resultados ordenados por fabricante y número de producto:

Solución: `SELECT ID_FAB, ID_PRODUCTO FROM PRODUCTOS WHERE PRECIO > 2000.00 UNION SELECT DISTINCT FAB, PRODUCTO FROM PEDIDOS WHERE IMPORTE > 30000.00 ORDER BY 1, 2`

FAB	PRODUCTO
ACI	4100Y
ACI	4100Z
IMM	775C
REI	A244L
REI	A244R

8.2 Uniones múltiples

La operación UNION puede ser utilizada repetidamente para combinar tres o más conjuntos de resultados. La unión de TABLA B y TABLA C en la figura produce una única tabla combinada. Esta tabla se combina luego con la TABLA A en otra operación UNION.

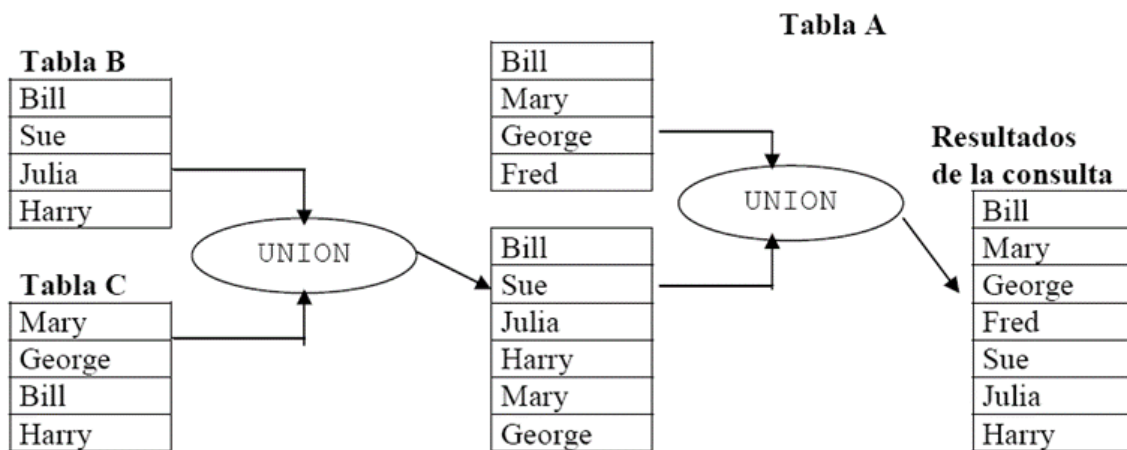


EJEMPLO PRÁCTICO

La consulta de la figura se escribe de este modo:

Solución: `SELECT * FROM A UNION (SELECT * FROM B UNION SELECT * FROM C)`

Los paréntesis que aparecen en la consulta indican que UNION debería ser realizada en primer lugar. De hecho, si todas las uniones de la sentencia eliminan filas duplicadas, o si todas ellas retienen filas duplicadas, el orden en que se efectúan no tienen importancia. Estas tres expresiones son equivalentes: A UNION (B UNION C), (A UNION B) UNION C, (A UNION C) UNION B. Sin embargo, si las uniones implican una mezcla de UNION y UNION ALL, el orden de la evaluación sí importa. Si esta expresión: A UNION ALL B UNION C se interpreta como: A UNION ALL (B UNION C). Entonces se producen diez filas de resultados (seis de la UNION interna, más cuatro filas de la TABLA A). Sin embargo, si se interpreta como obtendrá 7 filas de resultado (quita los repetidos): (A UNION ALL B) UNION C.

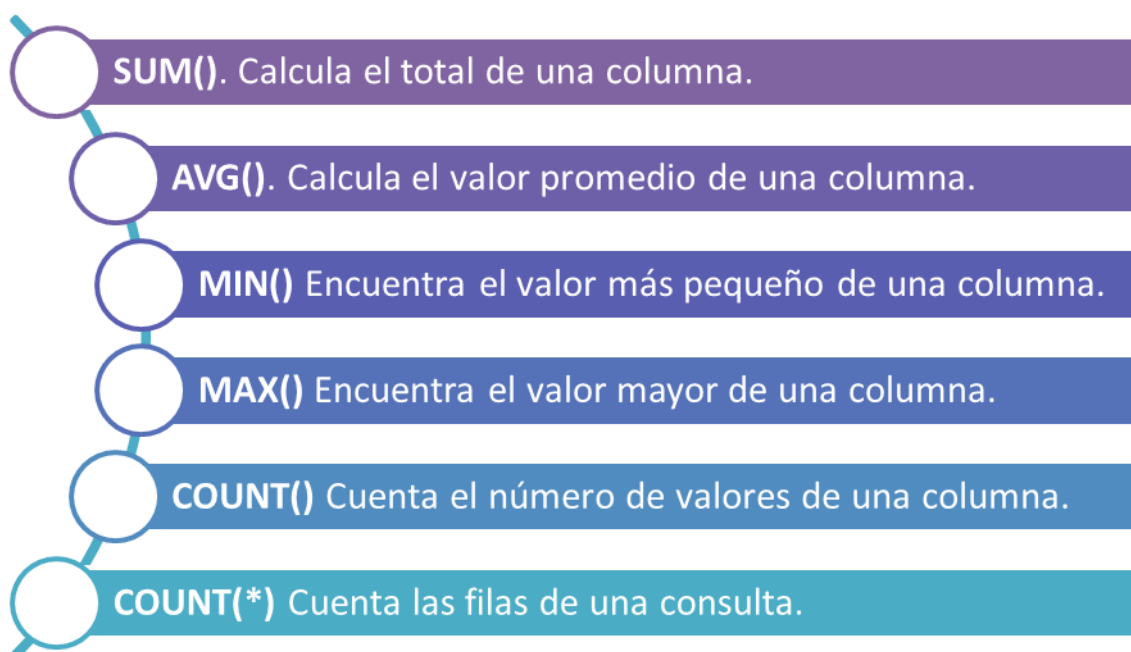


Ejemplo de UNION entre 3 tablas

9. CONSULTAS DE RESUMEN, GROUP BY, HAVING

Te encuentras en una situación similar a cuando tuviste que emplear las columnas calculadas. Algunos de los datos que necesitan no están en la base de datos y para obtenerlos debes usar algunas de las funciones que ofrece SQL, como SUM o COUNT. Para ello, es imprescindible que conozcas el funcionamiento de las cláusulas GROUP BY y HAVING.

SQL permite resumir datos de la base de datos mediante un conjunto de funciones de columna. Una función de columna SQL acepta una columna entera de datos como argumento y produce un único dato que resume la columna. Las funciones de columna ofrecen diferentes tipos de datos resumen:



Funciones de columna

El argumento de una función columna puede ser un solo nombre de columna o puede ser una expresión SQL.

9.1 Cálculo del total de una columna (SUM)

La función columna SUM() calcula la suma de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico (entero, decimal, coma flotante o monetario). El resultado de la función SUM() tiene el mismo tipo de datos básico que los datos de la columna, pero el resultado puede tener una precisión superior.

9.2 Cálculo del promedio de una columna (AVG)

La función de columna AVG() calcula el promedio de una columna de valores de datos. Al igual que una función SUM(), los datos de la columna deben tener un tipo numérico. Ya que la función AVG() suma los valores de la columna y luego lo divide por el número de valores, su resultado puede tener un tipo de dato diferente al de los valores de columna.

Si se aplica la función AVG() a una columna de enteros, el resultado será un número decimal o un número de coma flotante, dependiendo del producto SGBD concreto que se esté utilizando.



EJEMPLO PRÁCTICO

Calcula el precio medio de los productos del fabricante ACI:

Solución: `SELECT AVG(PRECIO) FROM PRODUCTOS
WHERE ID_FAB = 'ACI'`

AVG(PRECIO)
\$804.29

9.3 Determinación de valores extremos (MIN Y MAX)

Las funciones de columna MIN() Y MAX() determinan los valores menor y mayor de una columna, respectivamente. Los datos de la columna pueden contener información numérica, de cadena o de fecha/hora. El resultado de la función MIN() y MAX() tiene el mismo tipo de dato que los datos de la columna.



EJEMPLO PRÁCTICO

¿Cuáles son las cuotas asignadas mínima y máxima?:

Solución: `SELECT MIN (CUOTA) , MAX (CUOTA) FROM REPVENTAS`

MIN(CUOTA)	MAX(CUOTA)
\$200,000.00	\$350,000.00



EJEMPLO PRÁCTICO

¿Cuál es la fecha de pedido más antigua en la base de datos?:

Solución: `SELECT MIN (FECHA_PEDIDO) FROM PEDIDOS`

MIN(FECHA_PEDIDO)
04-ENE-89

Cuando las funciones de columnas MIN() y MAX() se aplican a datos numéricos, SQL compara los números en orden algebraico (los números negativos grandes son menores que los números negativos pequeños, los cuáles son menores que cero, el cual a su vez es menor que todos los números positivos). Las fechas se comparan secuencialmente (las fechas más antiguas son más pequeñas que las fechas más recientes).

Cuando se utilizan MIN() y MAX() con datos de cadenas, la comparación de las cadenas depende del conjunto de caracteres que esté siendo utilizado. En una computadora personal o en una minicomputadora los dígitos se encuentran delante de las letras en la secuencia de ordenación, y todos los caracteres mayúsculos se encuentran delante de todos los caracteres minúsculos.

9.4 Cuenta de valores de datos (COUNT)

La función de columna COUNT() cuenta el número de valores de datos que hay en una columna. Los datos de la columna pueden ser de cualquier tipo. La función COUNT() siempre devuelve un entero, independientemente del tipo de datos de la columna.



EJEMPLO PRÁCTICO

¿Cuántos clientes hay?

Solución: `SELECT COUNT (NUM_CLIE) FROM CLIENTES`

COUNT(NUM_CLIE)
21



EJEMPLO PRÁCTICO

¿Cuántos vendedores superan su cuota?

Solución: `SELECT COUNT (NOMBRE) FROM REPVENTAS
WHERE VENTAS > CUOTA`

COUNT(NOMBRE)
7



EJEMPLO PRÁCTICO

¿Cuántos pedidos de más de 25000€ hay en los registros?

Solución: `SELECT COUNT IMPORTE) FROM PEDIDOS
WHERE IMPORTE > 25000.00`

COUNT(IMPORTE)
4

Obsérvese que la función COUNT() ignora los valores de los datos de la columna; simplemente cuenta cuántos datos hay. No importa realmente qué columna se especifica como argumento de la función COUNT(). SQL permite una función de columna especial COUNT(*) que cuenta filas en lugar de valores de datos.



EJEMPLO PRÁCTICO

He aquí la misma consulta, reescrita para utilizar la función COUNT (*)

Solución: `SELECT COUNT (NUM_CLIE) FROM CLIENTES`

COUNT(*)
4

Si se piensa en la función COUNT(*) como en una función “cuenta filas”, la consulta resulta más fácil de leer. En la práctica, se utiliza casi siempre la función COUNT(*) en lugar de la función COUNT() para contar filas.

9.5 Valores NULL y funciones de columna

Las funciones de columna SUM(), AVG(), MIN(), MAX() y COUNT() aceptan cada una de ellas una columna de valores de datos como argumento y producen un único valor como resultado. ¿Qué sucede si uno o más de los valores de la columna es un valor NULL? El estándar SQL ANSI/ISO especifica que los valores NULL de la columna sean ignorados por las funciones de la columna.



EJEMPLO PRÁCTICO

Esta consulta muestra cómo la función de columna COUNT() ignora los valores NULL de una columna:

Solución: `SELECT COUNT (*), COUNT (VENTAS), COUNT (CUOTA)
FROM REPVENTAS`

COUNT(*)	COUNT(VENTAS)	COUNT(CUOTA)
10	10	9

La tabla REPVENTAS contiene diez filas, por lo que COUNT(*) devuelve una cuenta de diez. La columna VENTAS contiene diez valores no NULL, por lo que la función COUNT(VENTAS) también devuelve una cuenta de diez. La columna CUOTA es NULL para el vendedor más reciente. La función COUNT(CUOTA) ignora este valor NULL y devuelve una cuenta de nueve. Debido a estas anomalías, la función COUNT(*) es utilizada casi siempre en lugar de la función COUNT(), a menos que específicamente, se desee excluir del total los valores NULL de una columna particular.

MIN() y MAX(): Ignorar los valores NULL no tiene impacto en las funciones MIN() y MAX().

SUM(): Sin embargo, para la función SUM() el impacto es grande.

El resultado devuelto por las siguientes selecciones no sería el mismo: SUM(VENTAS) – SUM(CUOTA), SUM(VENTAS-CUOTA)

El motivo es que hay un vendedor que tiene con un valor NULL en la columna CUOTA. La expresión SUM(VENTAS) totaliza las ventas para los diez vendedores, mientras que la expresión SUM(CUOTA) lo totaliza solamente los nueve valores de cuota no NULL. La

expresión `SUM(VENTAS) – SUM(CUOTA)` calcula la diferencia de estos dos importes. Sin embargo, la función de columna `SUM(VENTAS-CUOTA)` tiene un valor de argumento no NULL para sólo nueve de los diez vendedores. En la fila con un valor de cuota NULL, la resta produce un NULL, que es ignorado por la función `SUM()`. Por tanto, las ventas del vendedor sin cuota, que están incluidas en el cálculo previo, se excluyen de este cálculo.

9.6 Eliminación de filas duplicadas (DISTINCT)

Recuérdese que se puede especificar la palabra clave `DISTINCT` al comienzo de la lista de selección para eliminar las filas duplicadas del resultado de la consulta. También se puede pedir a SQL que elimine valores duplicados de una columna antes de aplicarle una función de columna. Para eliminar valores duplicados, la palabra clave `DISTINCT` se incluye delante del argumento de la función de columna, inmediatamente después del paréntesis abierto.



EJEMPLO PRÁCTICO

¿Cuántos títulos diferentes tienen los vendedores?:

Solución: `SELECT COUNT (DISTINCT TITULO)`
`FROM REPVENTAS`

COUNT (DISTINCT TITULO)
3

Cuando se utiliza la palabra clave `DISTINCT`, el argumento de la función columna debe ser un simple nombre de columna; no puede ser una expresión. El estándar no permite el uso de la palabra clave `DISTINCT` con las funciones de columna `MIN()` y `MAX()`. `DISTINCT` no puede ser especificado con la función `COUNT(*)`.

La palabra clave `DISTINCT` sólo se puede especificar una vez en una consulta. Si aparece en el argumento de una función de columna, no puede aparecer en ninguna otra. Si se especifica delante de la lista de selección, no puede aparecer en ninguna función de columna.

9.7 Consultas agrupadas (GROUP BY)

Las consultas resumen descritas hasta ahora son, como los totales al final de un informe. Condensan todos los datos detallados del informe en una única fila resumen de datos.



EJEMPLO PRÁCTICO

¿Cuál es el tamaño medio de pedido?:

Solución: `SELECT COUNT (DISTINCT TITULO)
FROM REPVENTAS`

AVG(IMPORTE)
\$8,256.37



EJEMPLO PRÁCTICO

¿Cuál es el pedido medio de cada vendedor?:

Solución: `SELECT REP, AVG (IMPORTE) FROM PEDIDOS`

REP	AVG(IMPORTE)
101	\$8,876.00
102	\$5,694.00
103	\$1,350.00
105	\$7,685.40
106	\$16,479.00
107	\$11,477.33
108	\$3,552.50
110	\$11,566.00

La primera consulta es una consulta resumen simple como la de los ejemplos anteriores. La segunda consulta produce varias filas resumen, una fila por cada grupo, resumiendo los pedidos aceptados por un solo vendedor. Conceptualmente, SQL lleva a cabo la consulta del modo siguiente:

- SQL divide los pedidos en grupos de pedidos, un grupo por cada vendedor. Dentro de cada grupo, todos los pedidos tienen el mismo valor en la columna REP.
- Por cada grupo, SQL calcula el valor medio de la columna IMPORTE para todas las filas del grupo, y genera una única fila resumen de resultados. La fila contiene el valor de la columna REP del grupo y el pedido medio calculado.

Una consulta que incluya la cláusula GROUP BY se denomina consulta agrupada, ya que agrupa los datos de las tablas fuentes y produce una única fila resumen por cada grupo de filas. Las columnas indicadas en la cláusula GROUP BY se denominan columnas de agrupación de la consulta ya que son las que determinan cómo se dividen las filas en grupos.

MÚLTIPLES COLUMNAS DE AGRUPACIÓN: SQL puede agrupar resultados de consulta basándose en contenidos de dos o más columnas.



EJEMPLO PRÁCTICO

Se supone que se desean agrupar los pedidos por vendedor y por cliente. Esta consulta agrupa los datos basándose en ambos criterios. Calcula los pedidos totales por cada cliente y por cada vendedor:

Solución: `SELECT REP, CLIE, SUM(IMPORTE) FROM PEDIDOS
GROUP BY REP, CLIE`

REP	CLIE	SUM(IMPORTE)
101	2102	\$3,978.00
101	2108	\$ 150.00
101	2113	\$22,500.00
102	2106	\$4,026.00
102	2114	\$15,000.00
102	2120	\$3,750.00
103	2111	\$2,750.00
105	2103	\$35,582.00
105	2111	\$3,745.00

Las órdenes de agrupación deben agruparse por los campos que no son funciones de columna en la lista de campos.

RESTRICCIONES EN CONSULTAS AGRUPADAS: Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula FROM de la consulta. No se pueden agrupar las filas basándose en el valor de una expresión calculada.

También hay restricciones sobre los elementos que pueden aparecer en la lista de selección de una consulta agrupada. Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas. Básicamente, esto significa que un elemento de selección en una consulta agrupada puede ser:

- Una constante.
- Una función de columna.
- Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo.
- Una expresión que afecte a combinaciones de los anteriores.

En la práctica, una consulta agrupada incluirá siempre una columna de agrupación y una función de columna en su lista de selección.

VALORES NULL EN COLUMNAS DE AGRUPACIÓN: Un valor NULL presenta un problema especial cuando aparece en una columna de agrupación. Si el valor de la columna es desconocido, ¿en qué grupo debería colocarse la fila? En la cláusula WHERE, cuando se comparan dos valores NULL diferentes, el resultado es NULL (no TRUE), es decir, los dos valores NULL no se consideran iguales. Aplicando el mismo convenio a la cláusula GROUP BY se forzaría a SQL a colocar cada fila con una columna de agrupación NULL en un grupo aparte.

En la práctica, esta regla se demuestra que es demasiado rígida. En vez de ello, el estándar SQL ANSI/ISO considera que dos valores NULL son iguales a efectos de la cláusula GROUP BY. Si dos filas tienen NULL en las mismas columnas de agrupación no NULL, se agrupan dentro del mismo grupo de filas.

9.8 Condiciones de búsqueda de grupos (HAVING)

Al igual que la cláusula WHERE puede ser utilizada para seleccionar y rechazar filas individuales que participan en una consulta, la cláusula HAVING puede ser utilizada para seleccionar y rechazar grupos de filas. El formato de la cláusula HAVING es análogo al de la cláusula WHERE, consistiendo en la palabra clave HAVING seguida de una condición de búsqueda.



EJEMPLO PRÁCTICO

¿Cuál es el tamaño de pedido promedio para cada vendedor cuyos pedidos totalizan más de \$30000?:

Solución: `SELECT REP, AVG(IMPORTE) FROM PEDIDOS
GROUP BY REP HAVING SUM(IMPORTE) > 30000.00`

REP	AVG(IMPORTE)
105	\$7,865.40
106	\$16,479.00
107	\$11,477.33
108	\$8,376.14

La cláusula GROUP BY dispone primero de los pedidos en grupos por vendedor. La cláusula HAVING elimina entonces los grupos en donde el total de los pedidos no excede de \$30.000. Finalmente, la cláusula SELECT calcula el tamaño de pedido medio para cada uno de los grupos restantes y genera los resultados de la consulta. Las condiciones de búsqueda que se pueden especificar en la cláusula HAVING son las mismas de la cláusula WHERE.



EJEMPLO PRÁCTICO

Por cada oficina con dos o más personas, calcular la cuota total y las ventas totales para todos los vendedores que trabajan en la oficina:

Solución: `SELECT CIUDAD, SUM(CUOTA), SUM(REPVENTAS.VENTAS)
FROM OFICINAS, REPVENTAS WHERE OFICINA = OFICINA_REP GROUP BY CIUDAD
HAVING COUNT(*) >= 2`

CIUDAD	SUM(CUOTA)	SUM(REPVENTAS.VENTAS)
Chicago	\$775,000.00	\$735,042.00
Los Ángeles	\$700,000.00	\$835,915.00
New York	\$575,000.00	\$692,637.00

SQL gestiona esta consulta del modo siguiente:

- Compone las Tablas OFICINAS y REPVENTAS para hallar la ciudad en donde trabaja cada vendedor.
- Agrupa las filas resultantes por oficina.
- Elimina los grupos con dos o menos filas, éstas representan oficinas que no satisfacen el criterio de la cláusula HAVING.
- Calcula la cuota total y las ventas totales para cada grupo.



RECUERDA

SQL es un lenguaje universal válido tanto para sistemas de bases de datos privativos como de software libre.



EJEMPLO PRÁCTICO

Muestra el precio, las existencias y la cantidad total de los pedidos de cada producto para los cuales la cantidad total pedida es superior al 75 por 100 de las existencias:

Solución:

```
SELECT DESCRIPCION, PRECIO, EXISTENCIAS, SUM(CANT)
FROM PRODUCTOS, PEDIDOS WHERE FAB = ID_FAB AND PRODUCTO =
ID_PRODUCTO
GROUP BY DESCRIPCION, PRECIO, EXISTENCIAS HAVING SUM(CANT) > (0.75 *
EXISTENCIAS) ORDER BY EXISTENCIAS DESC
```

DESCRIPCION	PRECIO	EXISTENCIAS	SUM(CANT)
Reductor	\$355.00	38	32
Ajustador	\$25.00	37	30
Bancada motor	\$243.00	15	16
Bisagra Dcha.	\$4,500.00	12	15
Riostra 1-Tm	\$1,425.00	5	22

Para procesar esta consulta, SQL efectúa conceptualmente los siguientes pasos:

- Compone las Tablas PRODUCTOS y PEDIDOS para obtener la descripción, precio y existencias de cada producto pedido.
- Agrupa las filas resultantes por fabricante e id de producto.
- Elimina los grupos en donde la cantidad pedida es menor al 75% de las existencias.
- Calcula la cantidad total pedida para cada grupo.
- Genera una fila resumen de resultados por cada grupo.
- Ordena los resultados para que los productos con el mayor valor de existencias aparezcan en primer lugar.

9.9 Restricciones en condiciones de búsqueda grupal

La cláusula HAVING se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales. Esto significa que un elemento que aparezca dentro de la condición de búsqueda en una cláusula HAVING puede ser:

- Una constante

- Una función de columna, que produzca un único valor resumen de las filas del grupo.
- Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo.
- Una expresión que afecte a combinaciones de los anteriores

En la práctica, la condición de búsqueda de la cláusula HAVING incluirá siempre al menos una función de columna. Si no lo hiciera, la condición de búsqueda podría expresarse con la cláusula WHERE y aplicarse a filas individuales. El modo más fácil de averiguar si una condición de búsqueda pertenece a la cláusula WHERE o a la cláusula HAVING es recordar cómo se aplican ambas cláusulas:

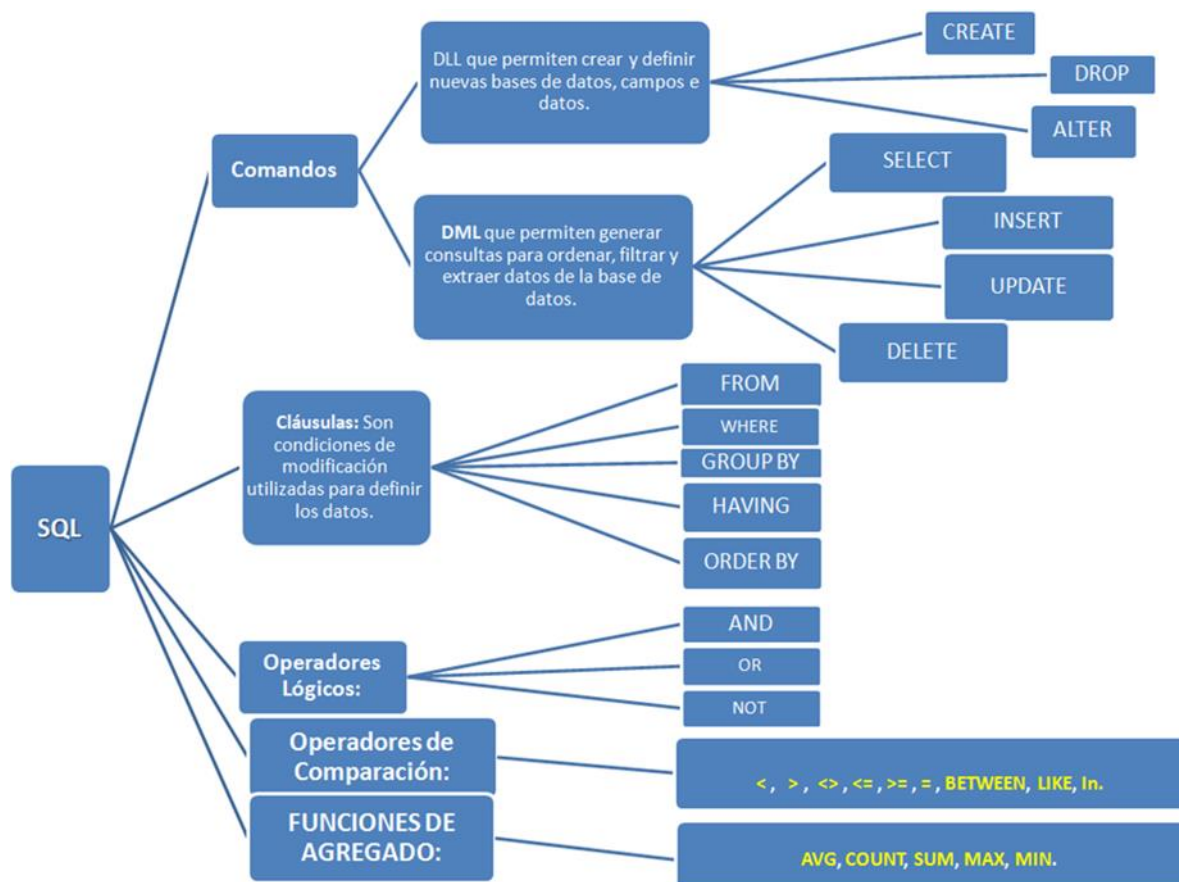
- La cláusula WHERE se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales.
- La cláusula HAVING se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.

Al igual que la condición de búsqueda WHERE, la condición de búsqueda de la cláusula HAVING puede producir uno de los tres resultados siguientes:

- Si la condición de búsqueda es TRUE, se retiene el grupo de filas y contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es FALSE, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es NULL, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.

9.10 HAVING sin GROUP BY

La cláusula HAVING se utiliza casi siempre juntamente con la cláusula GROUP BY, pero la sintaxis de la sentencia SELECT no lo precisa. Si una cláusula HAVING aparece sin una cláusula GROUP BY, SQL considera el conjunto entero de resultados detallados como un único grupo. En otras palabras, las funciones de columna de la cláusula HAVING se aplican a un solo y único grupo para determinar si el grupo está incluido o excluido de los resultados, y ese grupo está formado por todas las filas. El uso de una cláusula HAVING sin una cláusula correspondiente GROUP BY casi nunca se ve en la práctica.



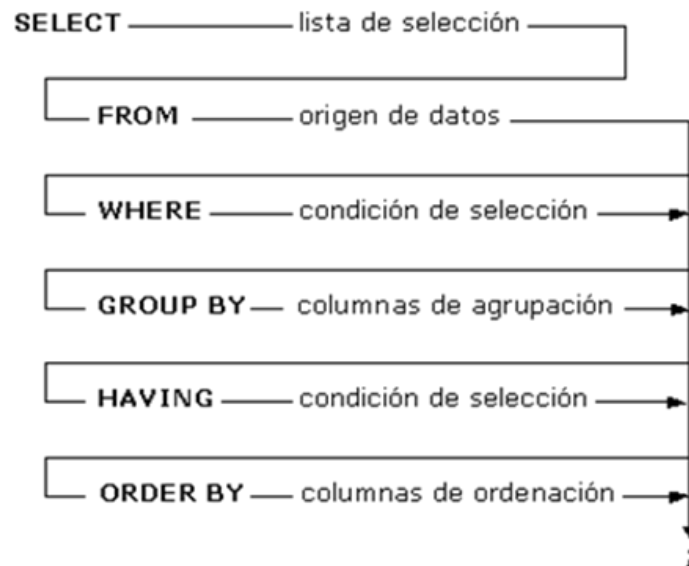
Cuadro resumen del lenguaje SQL

10. SINTAXIS DE LA ORDEN SELECT

A estas alturas ya conoces perfectamente la sintaxis de la orden SELECT. Sin embargo, tu ayudante tiene menos experiencia que tú y aún comete algunos errores a la hora de ordenar las diferentes cláusulas dentro de la misma.

Las cláusulas SELECT y FROM son necesarias para la creación de sentencias. Las cuatro cláusulas restantes (WHERE, ORDER, GROUP BY, HAVING) son opcionales.

Se incluyen en la sentencia SELECT cuando se desean usar sus funciones.



Resumen cláusula SELECT

- La cláusula `SELECT` lista los datos a recuperar por la sentencia `SELECT`. Los ítems pueden ser columnas de la base de datos o columnas a calcular por SQL cuando efectúe la consulta.
- La cláusula `FROM` lista las tablas que contienen los datos a recuperar por la consulta.
- La cláusula `WHERE` dice a SQL que incluya sólo ciertas filas de datos en los resultados de la consulta.
- La cláusula `GROUP BY` especifica una consulta resumen. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta resumen agrupa todas las filas similares y luego produce una fila resumen de resultados para cada grupo.
- La cláusula `HAVING` dice a SQL que incluya sólo ciertos grupos producidos por la cláusula `GROUP BY` en los resultados de la consulta. Al igual que la cláusula `WHERE`, utiliza una condición de búsqueda para especificar los grupos deseados.
- La cláusula `ORDER BY` ordena los resultados de la consulta en base a los datos de una o más columnas.



PARA SABER MÁS

Para ampliar información sobre la sintaxis de la cláusula SELECT, visita la web:



A modo de resumen:

- La sentencia SELECT se utiliza para expresar una consulta SQL. Toda sentencia SELECT produce una tabla de resultados que contienen una o más columnas y cero o más filas.
- La cláusula FROM especifica la(s) tabla(s) que contiene(n) los datos a recuperar por una consulta.
- La cláusula SELECT especifica la(s) columna(s) de datos a incluir en los resultados de la consulta, que pueden ser columnas de datos de la base de datos o columnas calculadas.
- La cláusula WHERE selecciona las filas a incluir en los resultados aplicando una condición de búsqueda a las filas de la base de datos.
- Una condición de búsqueda puede seleccionar filas mediante comparación de valores, mediante comparación de un valor con un rango o un grupo de valores, por correspondencia con un patrón de cadena o por comprobación de valores NULL.
- Las condiciones de búsqueda simples pueden combinarse mediante AND, OR y NOT para formar condiciones de búsqueda más complejas.
- La cláusula ORDER BY especifica que los resultados de la consulta deben ser ordenados en sentido ascendente o descendente, basándose en los valores de una o más columnas.

- La operación UNION puede ser utilizada dentro de una sentencia SELECT para combinar dos o más conjuntos de resultados y formar un único conjunto.
- Las consultas resumen utilizan funciones de columna SQL para condensar una columna de valores en un único valor que resuma la columna.
- Las funciones de columna pueden calcular el promedio, suma, el valor mínimo y máximo de una columna, contar el número de valores de datos de una columna o contar el número de filas de los resultados de la consulta.
- Una consulta resumen sin una cláusula GROUP BY genera una única fila de resultados, resumiendo todas las filas de una tabla o de un conjunto compuestos de tablas.

Es importante recordar:

- Una subconsulta es una consulta dentro de una consulta. Las subconsultas aparecen dentro de una de las condiciones de búsqueda subconsulta en la cláusula WHERE o HAVING.
- Cuando aparece una subconsulta en la cláusula WHERE, los resultados de la subconsulta se utilizan para seleccionar las filas individuales que contribuyen a los datos de los resultados de la consulta principal.
- Cuando una subconsulta aparece en la cláusula HAVING, los resultados de la subconsulta se utilizan para seleccionar los grupos de filas que contribuyen con datos a los resultados de la consulta.
- La forma subconsulta del test de comparación utiliza uno de los operadores de comparación simple para comparar un valor de test con el valor único devuelto por una subconsulta.
- La forma subconsulta del test de pertenencia a conjunto (IN) compara el valor de test con el conjunto de valores devuelto por una subconsulta.
- El test de existencia (EXISTS) comprueba si una subconsulta devuelve algún valor.
- Los test cuantificados (ANY y ALL) utilizan uno de los operadores de comparación simple para comparar un valor de test con todos los valores devueltos por una subconsulta, comprobando si la comparación resulta cierta para alguno o todos los valores.

- Una subconsulta puede incluir una referencia externa a una tabla de cualquiera de las consultas que la contienen, enlazando la subconsulta con la fila “actual” de esa consulta.

Las cláusulas de las sentencias SELECT proporcionan un conjunto potente y flexible de características para recuperar datos de la base de datos. Cada cláusula juega un papel específico en la recuperación de datos:

- La cláusula FROM especifica las tablas fuente que contribuyen con datos a los resultados de la consulta. Todos los nombres de columna en el cuerpo de la sentencia SELECT deben identificar sin ambigüedad a una columna de una tabla fuente en una consulta externa.
- La cláusula WHERE, si está presente, selecciona combinaciones individuales de filas procedentes de las tablas fuente que participan en los resultados de la consulta. Las subconsultas en la cláusula WHERE se evalúan para cada fila individual.
- La cláusula GROUP BY, si está presente, agrupa las filas individuales seleccionadas por la cláusula WHERE en grupos de filas.
- La cláusula HAVING, si está presente, selecciona grupos de filas que participan en los resultados de la consulta. Las subconsultas de la cláusula HAVING se evalúan para cada grupo de filas.
- La cláusula SELECT determina qué valores de datos aparecen realmente como columnas en los resultados finales.
- La palabra clave DISTINCT, si está presente, elimina filas duplicadas de los resultados de la consulta.
- El operador UNION, si está presente, mezcla los resultados producidos por las sentencias SELECT individuales en un único conjunto de resultados de la consulta.
- La cláusula ORDER BY, si está presente, ordena los resultados finales basándose en una o más columnas.

11. VISTAS

Algunas de las consultas que te han pedido hacer son bastante complejas por lo que, además de la dificultad de implementarlas, te encuentras con que las mismas van a ser poco eficientes en cuanto al tiempo que van a tardar en devolver el resultado esperado. Para evitar esto, Paula te recomienda el uso de Vistas.

En las bases de datos relacionales, una vista es una representación virtual de los datos almacenados en una o más tablas. Por lo tanto, una vista no contiene datos reales, sino que se basa en la definición de una consulta SQL que recupera los datos deseados de una o varias tablas. Actúa por lo tanto como una tabla virtual que se puede utilizar para consultar datos.

Dependiendo del SGBD que estemos usando, también es posible realizar inserciones, modificaciones o borrados en la tabla base a través de la vista. Para ello deben cumplirse ciertas condiciones:

- Que la vista esté basada en una única tabla.
- Que la vista no contenga elementos complejos como subconsultas, UNION o funciones agregadas.

Por ello, en la mayoría de los casos, estas operaciones se realizan directamente sobre las tablas subyacentes.

Algunos de los usos y características fundamentales de las vistas son:

- **Simplificación de consultas complejas:** las vistas permiten simplificar consultas el encapsular operaciones y relaciones complejas en una única vista.
- **Seguridad y control de acceso:** también se utilizan para aplicar restricciones de seguridad y control de acceso a los datos, pudiendo restringir los datos visibles a ciertos usuarios.
- **Mejora de rendimiento:** en ocasiones, las vistas pueden mejorar el rendimiento de las consultas. Por ejemplo, si una vista almacena los resultados de una consulta compleja y esta se usa con frecuencia, se evita así tener que ejecutar la consulta en cada instancia obteniéndose por lo tanto un mejor rendimiento.



EJEMPLO PRÁCTICO

```
CREATE VIEW VistaVentas AS
```

```
SELECT NombreProducto, Cantidad, (precio*cantidad) AS
```

```
IngresoTotal FROM Productos;
```

Una vez creada la vista, puedes utilizarla para consultar los datos de la siguiente manera:

```
SELECT * FROM VistaVentas;
```

12. OPTIMIZACIÓN DE CONSULTAS

Por fin has terminado de implementar todas las consultas, pero en algunos casos, estás recibiendo quejas por el tiempo que tardan dichas consultas en mostrar los resultados. Paula te recomienda que busques información sobre cómo optimizar consultas en MySQL.

La optimización de consultas es un proceso muy importante para mejorar el rendimiento de las mismas y garantizar una mayor eficiencia en la ejecución de operaciones en una base de datos. Algunos puntos clave a considerar para optimizar una consulta son los siguientes:

- **Índices:** son fundamentales para mejorar el rendimiento de las consultas. Es conveniente tener los índices adecuados en las columnas más frecuentemente utilizadas en las condiciones de selección.
- **Uso adecuado de cláusulas:** utilizar las cláusulas WHERE, GROUP BY u ORDER BY de manera eficiente para reducir el número de registros que se deben procesar.
- **Sentencias OR:** MySQL no puede usar índices si se usa la sentencia OR y alguna de las restricciones es una columna no indexada.
- **Evitar consultas innecesarias:** evitar el uso excesivo de comodines (%) en las cláusulas LIKE e intentar evitar subconsultas.

- **Análisis de consultas y perfiles:** existen herramientas de análisis de consultas y perfiles, como **EXPLAIN** y **SHOW PROFILES**, muy útiles para comprender cómo se ejecutan las consultas y detectar cuellos de botella o ineficiencias.
- **Actualización de versiones:** es conveniente mantener actualizado el SGBD con las actualizaciones más recientes.

Asimismo, SQL ofrece una herramienta de gran utilidad que se utiliza para analizar una consulta SQL. Dicha herramienta es la sentencia **Explain**, utilizada por la mayoría de los SGBD para obtener información sobre cómo se ejecuta una la consulta y la manera de acceder a los datos. Proporciona detalles sobre el plan de ejecución, el orden de las operaciones, índices utilizados y estadísticas. La sentencia EXPLAIN se ejecuta incluyendo la palabra EXPLAIN delante de la consulta: **EXPLAIN SELECT Nombre, Apellidos FROM EMPLEADO.**

Esta sentencia devolverá una tabla con una serie de filas que nos ofrecen información sobre las tablas involucradas en la consulta. Del resultado de este comando, cabe destacar las siguientes tres columnas:

- **Orden de las filas:** orden el que la base de datos consulta las tablas.
- **Columna Key:** indica, si lo hay, el índice que se está usando para acceder a una determinada tabla.
- **Columna select_type:** describe el tipo de operación de selección que se está realizando. Algunos de los valores que puede tomar son:
 - Simple: indica que es una simple operación que no involucra subconsultas ni uniones complejas.
 - Primary: Se refiere a la tabla base de una consulta SELECT, UPDATE o DELETE. Es la tabla principal de la consulta.
 - Subquery: representa una subconsulta que aparece en la lista de selección o en la cláusula WHERE.
- **Columna Type:** indica el tipo de acceso a la tabla. A continuación, se muestran algunos de los posibles valores, ordenados de mejor a peor:
 - System: la tabla tiene una sola fila.
 - Const: la tabla devuelve como máximo una fila. La tabla se lee una vez y el valor se considera como una constante.
 - Eq_ref: el índice primario o único se está utilizando completamente para la unión.
 - Ref: se está utilizando un índice de igualdad.
 - Fulltext: se basa en un índice de texto.
 - Index: se escanea el árbol del índice en lugar de la tabla completa.
 - All: se escanea toda la tabla.

Si algún acceso es Index o All, se deberían revisar los índices o la consulta.

- **Columna possible_keys:** indica los índices disponibles que el optimizador de consultas considera como posibles opciones para utilizar en la ejecución de la consulta.
- **Columna rows:** el número de filas que MySQL espera examinar para devolver el resultado de la consulta.

La tabla devuelta por EXPLAIN muestra las tablas en el orden en el que han sido procesadas (la primera fila se corresponde con la primera tabla tratada). Este orden es importante ya que está indicando el plan de ejecución de la consulta.



EJEMPLO PRÁCTICO

Dada la siguiente consulta:

EXPLAIN select * from employees where gender = 'F' and first_name LIKE 'B%

Se obtiene el siguiente resultado:

id	select_type	table	type	possible_keys	key	rows
1	SIMPLE	employees	ref	idx_gender	idx_gender	149601

Resultado clausula EXPLAIN con índice

La columna **select_type** tiene el valor SIMPLE, lo que indica que es una consulta sencilla, sin subconsultas o uniones. La columna **Table** muestra el nombre de la tabla sobre la cual se realiza la consulta. La columna **Type**, que hace referencia al tipo de acceso, tiene el valor ref, lo que indica que está usando un índice de igualdad.

Dicho índice podemos verlo en la columna **Key**, que es la columna que nos muestra los índices que se han usado. Como se puede apreciar, se ha usado un índice llamado **idx_gender**. Por último, la columna **rows** indica que, para devolver el resultado de la consulta, MySQL debe examinar 149.601 filas.

Si la tabla no tuviera ningún índice, el resultado sería el siguiente:

id	select_type	table	type	possible_keys	key	rows
1	SIMPLE	employees	ALL	NULL	NULL	299202

Resultado clausula EXPLAIN sin índice

Como se puede ver, la columna **type** ahora tiene el valor ALL, ya que no ha usado ningún índice y ha tenido que recorrer toda la tabla. Esto se puede apreciar también viendo el valor de la columna **rows**, que indica que ahora se necesitan examinar 299.202 filas para devolver el resultado de la consulta.



EJEMPLO PRÁCTICO

Dada la siguiente consulta:

```
EXPLAIN SELECT C.ID_Categoria, C.Nombre, P.ID_Producto, P.Nombre, P.PrecioUnitario,  
PR.ID_Proveedor, PR.Nombre  
FROM productos P
```

```
INNER JOIN categorias C ON P.ID_Categoria = C.ID_Categoria  
INNER JOIN proveedores PR ON PR.ID_Proveedor = P.ID_Proveedor
```

Se obtiene el siguiente resultado:

id	select_type	table	type	possible_keys	key
1	SIMPLE	C	ALL	PRIMARY	NULL
1	SIMPLE	P	ref	ID_Categoria,ID_Proveedor	ID_Categoria
1	SIMPLE	PR	eq_ref	PRIMARY	PRIMARY

Resultado clausula EXPLAIN

Como se puede apreciar, la columna **Table** muestra el alias de las tablas de la consulta. Así se puede ver el orden en el que han sido procesadas.

La columna **Type** hace referencia al tipo de acceso. Vemos que para la primera tabla el valor es **ALL**, por lo que habría que revisar por qué se está haciendo ese tipo de acceso.

La respuesta a esto se puede ver en la columna **Key**, que es la columna que nos muestra los índices que se han usado. Como se puede apreciar, para esta tabla no hay índices, por eso el tipo de acceso es **ALL**.



PARA SABER MÁS

Para ampliar información sobre la optimización de consultas y la sentencia EXPLAIN, visita este enlace:





ENLACE DE INTERÉS

Conoce más sobre la cláusula EXPLAIN:



VÍDEO DE INTERÉS

En este vídeo podrás ampliar información sobre la sentencia EXPLAIN:



RESUMEN FINAL

En la presente unidad se ha explicado cómo hacer consultas a bases de datos. Para ello, se ha estudiado la estructura de la sentencia SELECT, que especifica los ítems de datos a recuperar por la consulta, así como las diferentes opciones existentes para establecer criterios de búsqueda. La cláusula SELECT, por ejemplo, designa a las columnas solicitadas y la cláusula FROM designa a la tabla que las contiene.

Además, se han vistos las diferentes formas en las que los resultados de la consulta pueden ser ordenados y agrupados, así como la opción de eliminar resultados duplicados. Al mismo tiempo, se ha tratado la creación de vistas. Algunos de sus usos y características son: Simplificación de consultas complejas, Seguridad y control de acceso y Mejora del rendimiento.

Por último, se ha introducido la sentencia EXPLAIN para tratar el apartado de optimización de consultas. Esta optimización nos ayuda a mejorar el rendimiento de las mismas y garantizar una mayor eficiencia en la ejecución de las operaciones en una base de datos. La sentencia EXPLAIN se ejecuta incluyendo la palabra EXPLAIN delante de la consulta.