UNIDAD DIDÁCTICA 5

REALIZACIÓN DE CONSULTAS AVANZADAS

MÓDULO PROFESIONAL: BASES DE DATOS

Índice

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con consultas en SQL a nivel avanzado. En este sentido, se analizan situaciones que implican la consulta de varias tablas, composiciones con criterios de selección de fila, múltiples columnas de emparejamiento, consultas que involucran a tres o más tablas y otras equicomposiciones. Además, se presentan composiciones basadas en ciertas desigualdades, composición externa, interna y completas, y diferentes tipos de subconsultas. Se analiza una serie de consideraciones que conlleva SQL para consultas de varias tablas y la estructura de composiciones, como las anteriormente mencionadas. Por último, se introducen los conceptos de transacción y concurrencia.

INTRODUCCIÓN

En el ámbito de las bases de datos, las acciones que más impacto producen son las consultas y las subconsultas. En este preciso instante, se están ejecutando millones de consultas en todo el mundo. Por ejemplo, las entidades bancarias generan cientos de miles de consultas a diario de sus clientes y trabajadores. Es importante conocer la tipología de estas consultas, ya que, en función de las tablas que involucra, se realizará una consulta u otra.

Un aspecto importante a tener en cuenta es el tiempo de respuesta que conllevan las consultas a una base de datos. En muchas ocasiones, una consulta o subconsulta genera tiempos elevados de respuesta, las cuales ralentizan el funcionamiento de la aplicación. Este hecho actúa directamente sobre la calidad de dicho aplicativo, hasta el punto de poder crear un impacto negativo en la confianza del cliente, originando pérdidas. Es por ello que, en muchas ocasiones, tanto los tester de aplicaciones como los desarrolladores de software deben conocer la existencia de este tipo de anomalías y promover acciones de optimización de consultas y subconsultas, detectando los cuellos de botellas que éstas puedan ocasionar.

Es por ello que este tema ofrece al alumno un abanico de situaciones que pueden ayudar a realizar consultas y subconsultas óptimas en función de las tablas y de la situación que se desea consultar.

CASO INTRODUCTORIO

Te han pedido elaborar una base de datos de una entidad bancaria en la que una de las tablas almacena información sobre los clientes: nombre, dirección, teléfono, sueldo, etc. Esta tabla posee relaciones directas con otras tablas como, por ejemplo, productos

contratados, operaciones realizadas, etc. A la hora de realizar consultas en las que intervienen varias tablas, se hace necesario estudiar en profundidad el lenguaje SQL y observar la capacidad que ofrece a los usuarios y a los programadores para la interacción con la base de datos.

Al finalizar la unidad conocerás en profundidad las sentencias SQL, serás capaz de realizar consultas para obtener ciertos datos de la base de datos y consultas anidadas para obtener datos que se encuentren en distintas tablas de la base de datos.

1. CONSULTA DE VARIAS TABLAS

Debido a tu buen desempeño con las consultas realizadas anteriormente, tus responsables te piden que te encargues de obtener una serie de información para realizar una serie de cuadros de mando. En este caso, las consultas serán más complejas ya que deberás obtener la información de diferentes tablas al mismo tiempo.

Muchas consultas solicitan datos procedentes de 2 o más tablas de la base de datos. Estas peticiones de datos para la base de datos ejemplo extraen los datos de dos, tres o cuatro tablas:

"Lista vendedores y oficinas donde trabajan (tablas REPVENTAS y OFICINAS)".

"Lista los pedidos remitidos la última semana, mostrando el importe del pedido, el nombre del cliente que lo ordenó y el nombre del producto solicitado (tablas PEDIDOS, CLIENTES y REPVENTAS)".

SQL permite recuperar datos que corresponden a estas peticiones mediante consultas multitabla que componen (JOIN) datos procedentes de 2 o más tablas. Para comprender mejor las facilidades que SQL proporciona para consultas multitabla, lo mejor es comenzar con una petición simple que combina datos de dos tablas diferentes: *"Lista todos los pedidos, mostrando el número de pedido y su importe, y el nombre y el balance contable del cliente que los solicitó".* Los cuatro datos específicos solicitados están evidentemente almacenados en dos tablas diferentes:

- La tabla PEDIDOS contiene el número de pedido y el importe de cada pedido, pero no tiene los nombres de cliente ni los límites de crédito.

- La tabla CLIENTES contiene los nombres de cliente y sus balances, pero le falta la información referente a los pedidos.

Existe, sin embargo, un enlace entre estas dos tablas. En cada fila de la tabla PEDIDOS, la columna CLIE contiene el número del cliente que ordenó el pedido, el cual se corresponde con el valor en la columna NUM_CLIE de una de las filas de la tabla CLIENTES.

Evidentemente, la sentencia SELECT que gestiona la petición debe utilizar de algún modo este enlace entre tablas para generar sus resultados. Se debe tener presente dos cosas:

- Cada fila de resultados de la consulta extraerá sus datos de un par específico de filas, una de la Tabla PEDIDOS y otra de la Tabla CLIENTES.

- El par de filas en cuestión se determinará haciendo coincidir los contenidos de las columnas correspondientes de las tablas.

COMPOSICIONES SIMPLES (EQUICOMPOSICIONES).

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina componer las tablas. La tabla resultante (que contiene datos de las dos tablas originales) se denomina una composición entre las dos tablas. Una composición basada en una coincidencia exacta entre dos columnas se denomina más precisamente una equicomposición. Las composiciones son el fundamento del procesamiento de consultas multitabla en SQL. Todos los datos de una base de datos relacional están almacenados en sus columnas con valores explícitos, de modo que todas las relaciones posibles entre tablas pueden formarse comparando los contenidos de las columnas relacionadas.

EJEMPLO PRÁCTICO

Lista todos los pedidos mostrando su número, importe, número de cliente y el límite de crédito del cliente.

Solución:

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, LIMITE_CREDITO
FROM PEDIDOS, CLIENTES WHERE CLIE = NUM_CLIE;
```

Ésta tiene el mismo aspecto que las consultas del capítulo anterior, con dos características novedosas. Primero, la cláusula FROM lista dos tablas en lugar de una sola. En segundo lugar, la condición de búsqueda:

CLIE = NUM_CLIE

Compara columnas de dos tablas diferentes. A estas dos columnas se denominan columnas de emparejamiento para las dos tablas. Como todas las condiciones de búsqueda, ésta restringe las filas que aparecen en los resultados de la consulta. Puesto que ésta es una consulta de dos tablas, la condición de búsqueda restringe las parejas de filas que generan los resultados. Realmente captura el espíritu de la correspondencia manual de las columnas muy bien, diciendo:

"Genera resultados sólo para los pares de filas en los que el número de clientes (CLIE) en la tabla PEDIDOS coincide con el número de cliente (NUM_CLIE) en la tabla CLIENTES".

La sentencia SELECT no dice nada acerca de cómo SQL debería ejecutar la consulta. No hay mención de "comenzar con los pedidos" o "comenzar con los clientes". En lugar de ello, la consulta dice a SQL qué resultados deberían aparecer, y deja a SQL que decida cómo generarlos.

CONSULTA PADRE/HIJO.

Las consultas multitabla más comunes implican a dos tablas que tienen una relación padre/hijo. La consulta referente a pedidos y clientes de la sección precedente es un ejemplo de tal tipo de consulta.

Cada pedido (hijo) tiene un cliente asociado (padre), y cada cliente (padre) puede tener muchos pedidos asociados (hijos). Los pares de filas que generan los resultados de la consulta son combinaciones de fila padre/hijo. Se puede recordar que las claves ajenas y las claves primarias crean relaciones padre/hijo en una base de datos SQL. La tabla que contiene la clave ajena es el hijo en la relación; la tabla con la clave primaria es el padre. Para ejercitar la relación padre/hijo en una consulta debe especificarse una condición de búsqueda que compare la clave ajena y la clave primaria.

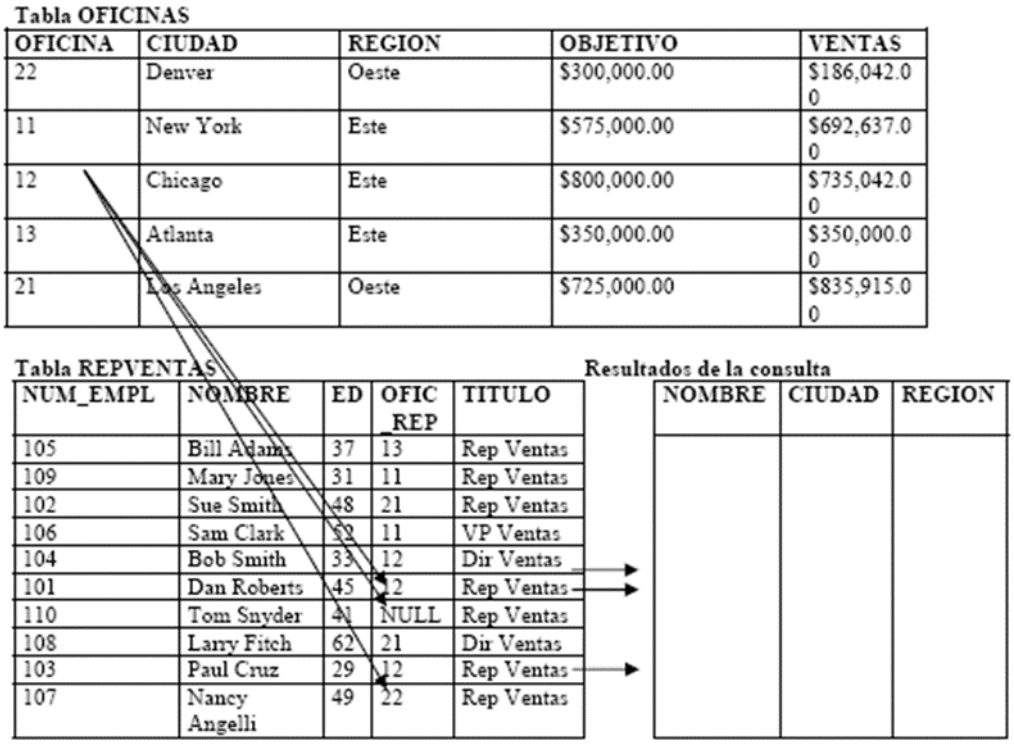
EJEMPLO PRÁCTICO

Lista cada uno de los vendedores y la ciudad y región en donde trabajan.

Solución:

NOMBRE	CIUDAD	REGION
Mary Jones	New York	Este
Sam Clark	New York	Este
Bob Smith	Chicago	Este
Paul Cruz	Chicago	Este
Dan Roberts	Chicago	Este
Bill Adams	Atlanta	Este
Sue Smith	Los Ángeles	Oeste
Larry Fitch	Los Ángeles	Oeste
Nancy Angelli	Denver	Oeste

```
SELECT NOMBRE, CIUDAD, REGION FROM REPVENTAS, OFICINAS
WHERE OFICINA_REP = OFICINA;
```



Resultado tras la ejecución de la consulta.

La Tabla REPVENTAS (hijo) contiene OFICINA_REP, una clave ajena para la Tabla OFICINAS (padre). Esta relación se utiliza para hallar la fila OFICINA correcta para cada vendedor, de modo que pueda incluirse la ciudad y región correctas en los resultados de la consulta.

He aquí otra consulta que hace referencia las mismas dos tablas, pero con los papeles de padre e hijo invertidos.

EJEMPLO PRÁCTICO

Lista las oficinas y los nombres y títulos de sus directores.

Solución:

```
SELECT CIUDAD, NOMBRE, TITULO
FROM OFICINAS, REPVENTAS WHERE DIR = NUM_EMPL;
```

CIUDAD	NOMBRE	TÍTULO
Chicago	Bob Smith	Dir Ventas
Atlanta	Bill Adams	Rep Ventas
New York	Sam Clark	VP Ventas
Denver	Larry Fitch	Dir Ventas
Los Ángeles	Larry Fitch	Dir Ventas

La Tabla OFICINAS (hijo) contiene DIR, una clave ajena para la Tabla REPVENTAS (padre). Esta relación se utiliza para hallar la fila REPVENTAS correcta para cada vendedor, de modo que puedan incluirse el nombre y el título correctos del director en los resultados de la consulta.

1.1 Composiciones con criterios de selección de fila

La condición de búsqueda que especifica las columnas de emparejamiento en una consulta multitabla puede combinarse con otras condiciones de búsqueda para restringir aún más los contenidos de los resultados. Se supone que se desea volver a ejecutar la consulta anterior, mostrando únicamente las oficinas con mayores objetivos de ventas.

EJEMPLO PRÁCTICO

Lista las oficinas con un objetivo superior a $600.000 y su director.

Solución:

```
SELECT CIUDAD, NOMBRE, TITULO FROM OFICINAS, REPVENTAS
WHERE DIR = NUM_EMPL AND OBJETIVO > 600,000.00;
```

CIUDAD	NOMBRE	TITULO
Chicago	Bob Smith	Dir Ventas

Con la condición de búsqueda adicional, las filas que aparecen en los resultados están aún más restringidas. El primer test (DIR = NUM_EMPL) selecciona solamente pares de filas OFICINAS y REPVENTAS que tienen la adecuada relación padre/hijo; el segundo test selecciona adicionalmente sólo aquellos pares de filas en donde la oficina está por encima del objetivo.

1.2 Múltiples columnas de emparejamiento

La tabla PEDIDOS y la tabla PRODUCTOS de la base de datos ejemplo están relacionadas por un par de claves ajena/primaria. Las columnas FAB y PRODUCTO de la tabla PEDIDOS forman juntas una clave ajena para la tabla PRODUCTOS, respectivamente. Para componer las tablas basándose en esta relación padre/hijo, deben especificarse ambos pares de columnas de emparejamiento, tal como se muestra en este ejemplo:

EJEMPLO PRÁCTICO

Lista los pedidos, mostrando los importes y las descripciones del producto.

Solución:

```
SELECT NUM_PEDIDO, IMPORTE, DESCRIPCION
FROM PEDIDOS, PRODUCTOS WHERE FAB = ID_FAB AND PRODUCTO =
ID_PRODUCTO;
```

NUM_PEDIDO	IMPORTE	DESCRIPCIÓN
113027	$4,104.00	Artículo Tipo 2
112992	$760.00	Artículo Tipo 2
113012	$3,745.00	Artículo Tipo 3
112968	$3,978.00	Artículo Tipo 4
112963	$3,276.00	Artículo Tipo 4
112983	$702.00	Artículo Tipo 4
113055	$150.00	Ajustador
113057	$600.00	Ajustador

La condición de búsqueda en la consulta dice a SQL que los pares de filas relacionados en las tablas PEDIDOS y PRODUCTOS son aquéllas en las que ambos pares de columnas coincidentes contienen los mismos valores.

1.3 Consultas de tres o más tablas

SQL puede combinar datos de tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas.

Esta consulta utiliza dos claves de la Tabla PEDIDOS. La columna CLIE es una clave ajena para la Tabla CLIENTES, que enlaza cada pedido con el cliente que lo remitió. La columna REP es una clave ajena para la Tabla REPVENTAS, que liga cada pedido con el vendedor que lo aceptó. Por ejemplo:

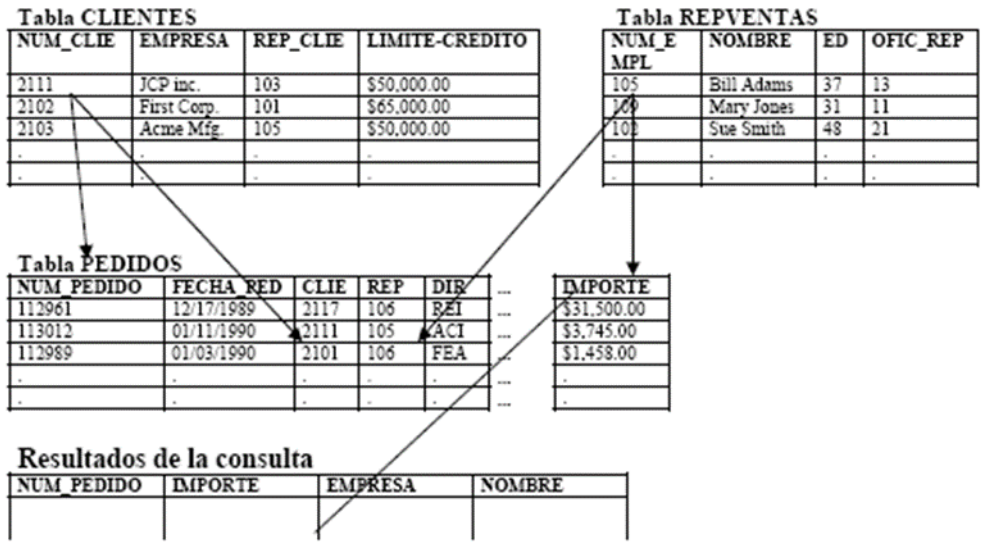
EJEMPLO PRÁCTICO

Lista los pedidos superiores a $25.000, incluyendo el nombre del vendedor que tomó el pedido y el nombre del cliente que lo solicitó.

Solución:

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, NOMBRE
FROM PEDIDOS, CLIENTES, REPVENTAS WHERE CLIE = NUM_CLIE
AND REP = NUM_EMPL AND IMPORTE > 25000.00
```

NUM_PEDIDO	IMPORTE	EMPRESA	NOMBRE
112987	$27,500.00	Acme Mfg.	Bill Adams
113069	$31,350.00	Chen Associates	Nancy Angelli
113045	$45,000.00	Zetacorp	Larry Fitch
112961	$31,500.00	J. P. Sinclair	Sam Clark



Resultado tras la ejecución de la consulta

No es extraño encontrar consultas de tres o incluso cuatro tablas en aplicaciones SQL de producción.

1.4 Otras equicomposiciones

La inmensa mayoría de las consultas multitabla se basan en relaciones padre/hijo, pero SQL no exige que las columnas de emparejamiento están relacionadas como clave primaria y clave ajena. Cualquier par de columnas de dos tablas pueden servir como

columnas de emparejamiento, siempre que tengan tipos de datos comparables. He aquí un ejemplo de una columna que utiliza un par de fechas como columnas de emparejamiento:

EJEMPLO PRÁCTICO

Halla todos los pedidos recibidos en los días en que un nuevo vendedor fue contratado.

Solución:

```
SELECT NUM_PEDIDO, IMPORTE, FECHA_PEDIDO, NOMBRE
FROM PEDIDOS, REPVENTAS WHERE FECHA_PEDIDO = CONTRATO
```

NUM_PEDIDO	IMPORTE	FECHA_PEDIDO	NOMBRE
112968	$3,978.00	12-OCT-89	Mary Jones
112979	$15,000.00	12-OCT-89	Mary Jones
112975	$2,100.00	12-OCT-89	Mary Jones
112968	$3,978.00	12-OCT-89	Larry Fitch
112979	$15,000.00	12-OCT-89	Larry Fitch
112975	$2,100.00	12-OCT-89	Larry Fitch

Los resultados de esta consulta provienen de los pares de filas de las Tablas PEDIDOS y REPVENTAS, en donde el valor en la columna FECH_PEDIDO coincide con el valor en la columna CONTRATO para el vendedor.

Ninguna de estas columnas es una clave ajena o una clave primaria, y la relación entre los pares de filas es ciertamente una relación extraña, la única cosa que los pedidos y vendedores correspondientes tienen en común es que resultan

Tabla REPVENTAS

NUM_EMPL	NOMBRE	...	CONTRATO
105	Bill Adams	...	02/12/1988
109	Mary Jones	...	10/12/1989
102	Sue Smith	...	12/10/1986
106	Sam Clark	...	06/14/1988
104	Bob Smith	...	05/19/1987
101	Dan Roberts	...	10/20/1986
110	Tom Snyder	...	01/13/1990
108	Larry Fitch	...	10/12/1989
103	Paul Cruz	...	03/01/1987
107	Nancy Angelli	...	11/14/1989

NUM_PEDIDO	FECHA_PEDIDO	CLIE
113051	02/10/1990	2118
112968	10/12/1989	2102
113036	01/30/1990	2107
113062	02/24/1990	2124
112979	10/12/1989	2114
113027	01/22/1990	2103
112992	11/04/1989	2118
112975	10/12/1989	2111
113055	02/15/1990	2108
.	.	.

Resultado tras la ejecución de la consulta.

Las columnas de emparejamiento como las de este ejemplo generan una relación de muchos a muchos entre las dos tablas. Puede haber muchos pedidos que compartan una única fecha de contrato del vendedor, y más de un vendedor puede haber sido contratado en la fecha de pedidos diferentes (112.968, 112.975 y 112.979) fueron recibidos el 12 de octubre de 1989, y dos vendedores diferentes (Larry Fitch y Mary Jones) fueron contratados el mismo día. Los tres pedidos y los dos vendedores producen seis filas de resultados de la consulta.

Esta relación de muchos a muchos es diferente de la relación de uno a muchos creada por columnas de emparejamiento clave primaria/clave ajena. La situación puede resumirse del siguiente modo:

- Las composiciones que asocian claves primarias a claves ajenas siempre crean relación padre/hijo de uno a muchos.

- En general, las composiciones sobre las columnas de emparejamiento arbitrarias generan relaciones de muchos a muchos.

Estas dos situaciones diferentes no tienen nada que ver con cómo se escriba la sentencia SELECT que expresa la composición.

2. COMPOSICIONES BASADAS EN DESIGUALDAD

Como tienes bastante carga de trabajo implementado las consultas, Paula te ha asignado una persona en prácticas para que te ayude. Para empezar, le pides que se familiarice con el concepto y significado del término JOIN.

El término “composición” (JOIN) se aplica a cualquier consulta que combina datos de dos tablas mediante comparación de los valores en una pareja de columnas de tablas.

EJEMPLO PRÁCTICO

Lista todas las combinaciones de vendedores y oficinas en donde la cuota del vendedor es superior al objetivo de la oficina.

Solución:

```
SELECT NOMBRE, CUOTA, CIUDAD, OBJETIVO FROM REPVENTAS, OFICINAS
WHERE CUOTA > OBJETIVO
```

NOMBRE	CUOTA	CIUDAD	OBJETIVO
Bill Adams	$350,000.00	Denver	$300,000.00
Sue Smith	$350,000.00	Denver	$300,000.00
Larry Fitch	$350,000.00	Denver	$300,000.00

Como en todas las consultas de dos tablas, cada fila de resultados proviene de un par de filas, en este caso de las Tablas REPVENTAS y OFICINAS. La condición de búsqueda: CUOTA > OBJETIVO. Selecciona pares de filas en donde la columna CUOTA de la fila REPVENTAS excede a la columna OBJETIVO de la fila OFICINAS. Observe que los pares de filas REPVENTAS y OFICINAS están relacionadas únicamente de este modo; no se requiere específicamente que la fila REPVENTAS represente a alguien que trabaje en la oficina representada por la fila OFICINAS. El ejemplo es un poco extremado, e ilustra por qué las composiciones basadas en desigualdades no son muy comunes.

3. CONSIDERACIONES SQL PARA CONSULTAS DE VARIAS TABLAS

Para seguir con la formación del nuevo compañero y con el objetivo de que te pueda ayudar con las consultas lo más pronto posible, le pides que tenga en cuenta una serie de consideraciones a la hora de enfrentarse a una consulta que obtiene los datos de varias tablas.

Algunas consultas multitabla no pueden ser expresadas sin las características adicionales del lenguaje SQL descritas en las secciones siguientes. Específicamente:

- Los nombres de columna cualificados son necesarios a veces en consultas multitabla para eliminar referencias de columna ambiguas.

- Las selecciones de todas las columnas (SELECT *) tienen un significado especial para las consultas multitabla.

- Las autocomposiciones pueden ser utilizadas para crear una consulta multitabla que relaciona una tabla consigo misma.

- Los alias de tablas pueden ser utilizadas en la cláusula FROM para simplificar nombres de columna cualificados y permitir referencias de columna no ambiguas en autocomposiciones.

NOMBRES DE COLUMNA CUALIFICADOS.

La base de datos ejemplo incluye varias instancias en donde dos tablas contienen columnas con el mismo nombre. La Tabla OFICINAS y la Tabla REPVENTAS, por ejemplo, tienen ambas una columna de nombre VENTAS. La columna de la Tabla OFICINAS contiene las ventas anuales hasta la fecha para cada oficina; la de la Tabla REPVENTAS contiene las ventas anuales hasta la fecha de cada vendedor. Normalmente, no hay confusión entre ambas columnas, ya que la cláusula FROM determina cuál de ellas es la adecuada en una consulta determinada, como en estos ejemplos:

“Muestra las ciudades en donde las ventas superan al objetivo:”

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

Sin embargo, he aquí una consulta en donde los nombres duplicados provocan un problema:

"Muestra el nombre, las ventas y la oficina de cada vendedor:"

```
SELECT NOMBRE, VENTAS, CIUDAD
FROM REPVENTAS, OFICINAS WHERE OFICINA_REP = OFICINA
```

Error: Nombre de columna ambiguo "VENTAS"

Aunque la descripción textual de la consulta implica que se desea la columna VENTAS de la tabla REPVENTAS, la consulta SQL es ambigua. Para eliminar la ambigüedad, debe utilizarse un nombre de columna cualificado para identificar la columna.

Un nombre de columna cualificado especifica el nombre de una columna y la tabla que contiene a la columna. Los nombres cualificados de las dos columnas VENTAS en base de datos son: OFICINAS, VENTAS y REPVENTAS.VENTAS

Un nombre de columna cualificado puede ser utilizado en una sentencia SELECT en cualquier lugar en donde se permite un nombre de columna. La tabla especificada en el nombre de columna cualificado, debe, naturalmente, corresponder a una de las tablas especificadas en la lista FROM. He aquí una versión corregida de la consulta anterior que utiliza un nombre de columna cualificado:

"Muestra el nombre, las ventas y la oficina de cada vendedor"

```
SELECT NOMBRE, REPVENTAS.VENTAS, CIUDAD
FROM REPVENTAS, OFICINAS  WHERE OFICINA_REP = OFICINA
```

Utilizar nombres de columnas cualificados en una consulta multitabla es siempre una buena medida. La desventaja, naturalmente, es que hacen que el texto de la consulta sea mayor.

SELECCIONES DE TODAS LAS COLUMNAS.

"SELECT *" puede ser utilizado para seleccionar todas las columnas de la tabla designada en la cláusula FROM. En una consulta multitabla, el asterisco selecciona todas las columnas de todas las tablas listadas en la cláusula FROM.

La siguiente consulta produciría 15 columnas de resultados, las 9 columnas de la tabla REPVENTAS seguidas de las 6 columnas de la tabla OFICINAS:

"Informa sobre todos los vendedores y todas las oficinas en las que trabajan"

```
SELECT * FROM REPVENTAS, OFICINAS WHERE OFICINA_REP = OFICINA
```

Obviamente, la forma SELECT * de una consulta resulta ser mucho menos práctica cuando hay dos, tres o más tablas en la cláusula FROM.

En la siguiente consulta, el ítem de selección REPVENTAS.* se expande a una lista que contiene únicamente las columnas halladas en la Tabla REPVENTAS:

"Informa acerca de todos los vendedores y los lugares en los que trabajan".

```
SELECT REPVENTAS.*, CIUDAD, REGION FROM REPVENTAS, OFICINAS
WHERE OFICINA_REP = OFICINA
```

La consulta produciría once columnas de resultados, las nueve columnas de la Tabla REPVENTAS, seguidas de las dos columnas explícitamente solicitadas de la Tabla OFICINAS.

ALIAS DE TABLAS.

Si una consulta se refiere a la tabla de otro usuario, o si el nombre de una tabla es muy largo, puede ser tedioso escribir el nombre de la tabla como cualificador de una columna. Esta consulta, que referencia a la tabla CUMPLEAÑOS propiedad del usuario SAM:

"Lista los nombres, cuotas y cumpleaños de los vendedores."

```
SELECT REPVENTAS, NOMBRE, CUOTA, SAM.CUMPLEAÑOS_FECHA
FROM REPVENTAS, SAM.CUMPLEAÑOS WHERE REPVENTAS.NOMBRE =
SAM.CUMPLEAÑOS.NOMBRE
```

EJEMPLO PRÁCTICO

Resulta más fácil de leer y escribir cuando se utilizan los alias S y B para las dos tablas: "Lista los nombres, cuotas y cumpleaños de los vendedores".

Solución:

```
SELECT S.NOMBRE, S.CUOTA, B.FECHA FROM REPVENTAS S, SAM.CUMPLEAÑOS B
WHERE S.NOMBRE = B.NOMBRE
```

La cláusula tiene dos funciones importantes:

- La cláusula FROM identifica todas las tablas que contribuyen con datos a los resultados de la consulta. Las columnas referenciadas en la sentencia SELECT deben provenir de una de las tablas designadas en la cláusula FROM.

- La cláusula FROM determina la marca que se utiliza para identificar la tabla en referencias de columna cualificadas dentro de la sentencia SELECT. Si se especifica un alias, éste pasa a ser la marca de la tabla; en caso contrario se utiliza como marca el nombre de la tabla, tal como aparece en la cláusula FROM.

La única exigencia para las marcas de tablas en la cláusula FROM es que todas las marcas de una cláusula FROM determinada deben ser distintas unas de otras.

AUTOCOMPOSICIONES (UNA TABLA CONSIGO MISMA).

Algunas consultas multitabla afectan a una relación que una tabla tiene consigo misma. Por ejemplo, se supone que se desea listar los nombres de todos los vendedores y sus directores. Cada vendedor aparece como una fila en la Tabla REPVENTAS, y la columna DIRECTOR contiene el número de empleado del director del vendedor. Parecería que la columna DIRECTOR debería ser una clave ajena para la tabla que contiene datos referentes a los directores. En efecto así es, se trata de una clave ajena para la propia tabla REPVENTAS.

Si se tratara de expresar esta consulta como cualquier otra consulta de dos tablas implicando una coincidencia clave ajena/clave primaria, aparecería tal como ésta:

```
SELECT NOMBRE, NOMBRE
FROM REPVENTAS, REPVENTAS
WHERE DIRECTOR = NUM_EMPL
```

Esta sentencia SELECT es ilegal debido a la referencia duplicada a la tabla REPVENTAS en la cláusula FROM. También podría intentarse eliminar la segunda referencia a la tabla REPVENTAS.

```
SELECT NOMBRE, NOMBRE
FROM REPVENTAS
WHERE DIRECTOR = NUM_EMPL
```

Esta consulta es legal, pero no hará lo que se desea que haga. Es una consulta monotabla, por lo que SQL recorre la tabla REPVENTAS fila a fila, aplicando la condición de búsqueda: DIRECTOR = NUM_EMPL

Las filas que satisfacen esta condición son aquéllas en donde las dos columnas tienen el mismo valor, es decir, las filas en donde un vendedor es su propio director. No existen tales filas, por lo que la consulta no produciría ningún resultado.

Para comprender cómo resuelve SQL este problema, se imagina que hubiera dos copias idénticas de la Tabla REPVENTAS, una llamada EMPS, que contuviera los empleados, y otra llamada DIRS, que contuviera los directores. La columna DIRECTOR de la Tabla EMPS sería entonces una clave ajena para la Tabla DIRS, y la siguiente consulta funcionaría:

"Lista los nombres de los vendedores y sus directores."

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE
FROM EMPS, DIRS WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

Puesto que las columnas de las dos tablas tienen nombres idénticos, todas las referencias de columnas están cualificadas.

SQL utiliza exactamente esta estrategia de "tabla duplicada imaginaria" para componer una tabla consigo misma. En lugar de duplicar realmente el contenido de la tabla. SQL simplemente permite referirse a ella mediante un nombre diferente, llamado un alias de tabla. He aquí la misma consulta, escrita utilizando los alias EMPS y DIRS para la Tabla REPVENTAS.

"Lista los nombres de los vendedores y sus directores."

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE
FROM REPVENTAS EMPS, REPVENTAS DIRS WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

EMPS.NOMBRE	DIRS.NOMBRE
Tom Snyder	Dan Roberts
Bill Adams	Bob Smith
Dan Roberts	Bob Smith
Paul Cruz	Bob Smith
Mary Jones	Sam Clark
Bob Smith	Sam Clark
Larry Fitch	Sam Clark
Sue Smith	Larry Fitch
Nancy Angelli	Larry Fitch

La cláusula FROM asigna un alias "copia" de la tabla REPVENTAS especificando el nombre del alias inmediatamente después del nombre real de la tabla. Como muestra el ejemplo, cuando una cláusula FROM contiene un alias, el alias debe ser utilizado para identificar la tabla en referencias de columna cualificadas.

"Lista los vendedores con una cuota superior a la de sus directores."

```
SELECT REPVENTAS, NOMBRE, REPVENTAS.CUOTA, DIRS.CUOTA
FROM REPVENTAS, REPVENTAS DIRS WHERE REPVENTAS.DIRECTOR =
DIRS.NUM_EMPL
AND REPVENTAS.CUOTA > DIRS.CUOTA
```

4. LA ESTRUCTURA DE UNA COMPOSICIÓN

Tu ayudante ha empezado a realizar algunas de las consultas, pero revisando su trabajo, detectas algunos errores. Es por ello que te dispones a explicarle una serie de conceptos que consideras importantes.

Cuando se componen muchas tablas o cuando las condiciones de búsqueda resultan complejas, sin embargo, es muy difícil tan sólo mirando una sentencia SELECT imaginar lo que significa. Por esta razón, es muy útil definir más cuidadosamente lo que es una composición y qué resultados se producen con una sentencia SELECT determinada.

MULTIPLICACIÓN DE TABLAS.

Una composición es un caso especial de una combinación más general de datos procedentes de dos tablas, conocida como el producto cartesiano de dos tablas. El producto de dos tablas es otra tabla (la tabla producto); que consta de todos los pares posibles de filas de las dos tablas. Las columnas de la tabla producto son todas las columnas de la primera tabla, seguidas de todas las columnas de la segunda tabla.

Si se especifica una consulta de dos Tablas sin una cláusula WHERE, SQL produce el producto de las dos tablas como resultado.

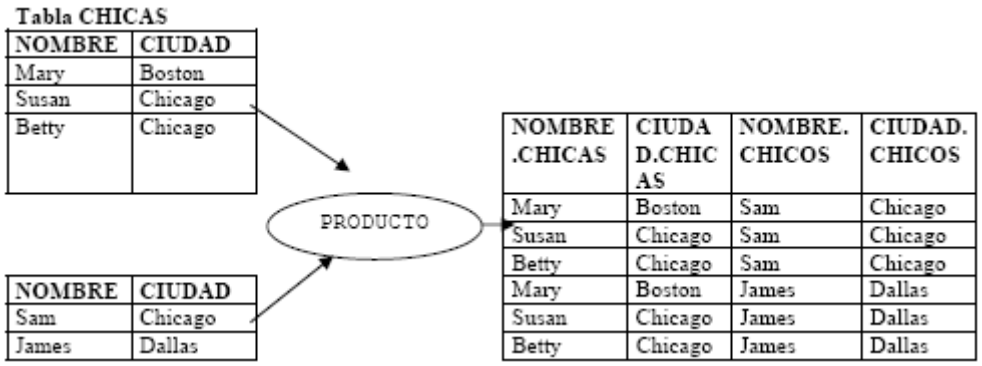
EJEMPLO PRÁCTICO

Muestra todas las combinaciones posibles de vendedores y ciudades.

Solución:

```
SELECT NOMBRE, CIUDAD FROM REPVENTAS, OFICINAS
```

Produciría el producto de las Tablas REPVENTAS y OFICINAS, mostrando todos los pares posibles vendedor/ciudad. Habría 50 filas de resultados (5 oficinas * 10 vendedores = 50 combinaciones).



Resultado tras la ejecución de la consulta.

REGLAS DE PROCESAMIENTO DE CONSULTAS MULTITABLA.

Las reglas definen el significado de cualquier sentencia SELECT multitabla especificando un procesamiento que siempre genera el conjunto correcto de resultados de la consulta. Para generar los resultados de una consulta con una sentencia SELECT:

- Si la sentencia es una UNION de sentencias SELECT, aplicar los pasos 2 hasta 5 a cada una de las sentencias para generar sus resultados individuales.

- Formar el producto de las tablas indicadas en la cláusula FROM. Si la cláusula FROM designa una sola tabla, el producto es esa tabla.

- Si hay una cláusula WHERE, aplicar su condición de búsqueda a cada fila de la tabla producto reteniendo aquellas filas para las cuales la condición de búsqueda es TRUE (descartando aquellas 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 la sentencia es una UNION de sentencia SELECT, mezclar los resultados de consulta para las sentencias individuales en una única tabla de resultados. Eliminar las filas duplicadas a menos que se haya especificado UNION ALL:

- Si hay una cláusula ORDER BY, ordenar la formación de los resultados de la consulta.

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

EJEMPLO PRÁCTICO

Lista el nombre de empresa y pedidos para el número de cliente 2.103.

Solución:

```
SELECT EMPRESA, NUM_PEDIDO, IMPORTE FROM CLIENTES, PEDIDOS
WHERE NUM_CLIE = CLIE AND NUM_CLIE = 2103 ORDER BY NUM_PEDIDO
```

EMPRESA	NUM_PEDIDO	IMPORTE
Acme Mfg.	112963	$3,276.00
Acme Mfg.	112983	$702.00
Acme Mfg.	112987	$27,500.00
Acme Mfg.	113027	$4,104.00

Siguiendo los pasos:

- La cláusula FROM genera todas las combinaciones posibles de filas de la Tabla CLIENTES (21 filas) y de la Tabla PEDIDOS (30 filas), produciendo una Tabla PRODUCTO de 630 filas.

- La cláusula WHERE selecciona únicamente aquellas filas de la tabla PRODUCTO donde los números de cliente coinciden (NUM_CLIE = CLIE) y el número de cliente es el especificado (NUM_CLIE = 2103). Solamente se seleccionan cuatro filas; las otras 626 se eliminan.

- La cláusula SELECT extrae las tres columnas solicitadas (EMPRESA, NUM_PEDIDO e IMPORTE_RED) de cada fila que queda de la Tabla PRODUCTO para generar cuatro filas de resultados detallados.

- La cláusula ORDER BY ordena las cuatro filas por la columna NUM_PEDIDO para generar el resultado final.

5. COMPOSICIONES EXTERNA, INTERNA Y COMPLETA

La persona a tu cargo está empezando a mejorar, pero en el resultado de una de las consultas, echas en falta bastante registros en la tabla resultante. Es por ello que le pides a tu ayudante que busque información sobre la diferencia entre composición interna, externa y completa.

Las **composiciones** o **JOINs** en SQL se utilizan para **combinar filas de dos o más tablas**, usando para ello un campo común entre ellas y devolviendo, por lo tanto, datos de las diferentes tablas. Una composición o JOIN se produce cuando, al menos dos tablas, se unen en una sentencia SQL. Los principales son los siguientes:

1. **INNER JOIN:** devuelve todas las filas si por lo menos hay **una coincidencia en ambas tablas**. También es conocida como **composición interna.**

2. **LEFT JOIN:** muestra todas las filas de la tabla **izquierda** y las filas que coincidan de la tabla **derecha.** También es conocida como **composición externa por la izquierda.**

3. **RIGHT JOIN:** muestra todas las filas de la tabla **derecha** y las coincidentes de la tabla izquierda. También conocida como **composición externa por la derecha.**

4. **FULL OUTER JOIN:** o **composición completa.** Esta composición **no está implementada en MySQL** por lo que, para emular su comportamiento, será necesario usar el operador **UNION**, para poder unir el resultado de dos consultas. El resultado esperado de un FULL OUTER JOIN es obtener la intersección de dos consultas, más las filas de ambas tablas que no se han podido combinar. En otras palabras, el resultado sería equivalente a realizar la unión de una consulta con un LEFT JOIN y otra consulta con un RIGHT JOIN sobre las mismas tablas.

A continuación, se muestra un ejemplo con las tablas Clientes y Pedidos.

CLIENTES		
ClienteID	NombreCliente	Contacto
1	Antonio López	44444444
2	Lydia Sánchez	55555555
3	Tomás Gómez	66666666
4	Josefa Pérez	77777777

PEDIDOS		
ClienteID	NombreCliente	Contacto
234	4	160
235	2	48
236	3	64
237	4	92

Si realizáramos la siguiente consulta sin hacer uso de ningún tipo de composición:

```
SELECT NombreCliente, PedidoID FROM Clientes, Pedidos
```

Obtendríamos un resultado como este:

NombreCliente	PedidoID
Antonio López	234
Lydia Sánchez	234
Tomás Gómez	234
Josefa Pérez	234
Antonio López	235
Lydia Sánchez	235
Tomás Gómez	235
Josefa Pérez	235
Antonio López	236
Lydia Sánchez	236
Tomás Gómez	236
Josefa Pérez	236
Antonio López	237
Lydia Sánchez	237
Tomás Gómez	237
Josefa Pérez	237

Producto cartesiano.

El resultado de esta consulta es consecuencia de que no se han incluido en la misma las referencias a las columnas asociadas en ambas tablas, es decir, la columna ClienteID. Al no establecer esta asociación, MySQL hace lo que se conoce como un producto cartesiano, por lo que cruza todas las filas de la tabla Clientes con todas las filas de la tabla Pedidos. Si nos fijamos, el resultado muestra, por cada cliente de la tabla Clientes,

todos los ID de pedido de la tabla Pedidos. Evidentemente no es la solución que estamos buscando.

Si a esta misma consulta le aplicamos una composición interna o INNER JOIN:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
INNER JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID
```

Otra forma de aplicar una composición interna sin hacer uso de la cláusula INNER JOIN sería la siguiente:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes, Pedidos
WHERE Clientes.ClienteID = Pedidos.ClienteID
```

Si hay filas en Clientes que no tienen coincidencias en Pedidos, los Clientes no se mostrarán. Las sentencias anteriores mostrarán el siguiente resultado:

CLIENTES	
NombreCliente	PedidoID
Josefa Pérez	234
Lydia Sánchez	235
Tomás Gómez	236
Josefa Pérez	237

Josefa aparece dos veces ya que ha realizado dos pedidos. Sin embargo, Antonio López no aparece ya que no ha realizado ningún pedido.

EJEMPLO PRÁCTICO

Realiza una consulta que devuelva únicamente los nombres de los profesores asignados a un departamento, junto con el nombre del departamento.

Solución:

```
SELECT Profesor.Nombre, Departamento.Nombre
FROM Profesor
INNER JOIN Departamento ON Profesor.DeptoID=Departamento.ID
```

VÍDEO DE INTERÉS

Para saber más sobre las composiciones internas puedes ver este vídeo:

Si queremos que la consulta muestre todos los clientes, aunque no hayan realizado ningún pedido, tendremos que utilizar la composición externa LEFT JOIN. LEFT JOIN mantiene todas las filas de la tabla izquierda (Clientes). Las filas de la tabla derecha (Pedidos) se mostrarán si hay una coincidencia con las de la izquierda. Si existen valores en la tabla izquierda pero no en la tabla derecha, ésta mostrará NULL.

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
LEFT JOIN Pedidos ON Clientes.ClienteID = Pedidos.ClienteID
ORDER BY Pedidos.PedidoID
```

El resultado de esta consulta sería el siguiente:

CLIENTES	
Nombre Cliente	PedidoID
Antonio López	NULL
Josefa Pérez	234
Lydia Sánchez	235
Tomás Gómez	236
Josefa Pérez	237

Ahora podemos ver que se muestran todas las filas de la tabla Clientes, que es la tabla de la izquierda, tantas veces como coincidencias haya con el lado derecho (Pedidos). Antonio López no ha realizado ningún pedido, por lo que se muestra NULL.

EJEMPLO PRÁCTICO

Realiza una consulta que devuelva los nombres de los profesores, aunque no estén asignados a un departamento, junto con el nombre del departamento.

Solución:

```
SELECT Profesor.Nombre, Departamento.Nombre
FROM Profesor
LEFT JOIN Departamento ON Profesor.DeptoID=Departamento.ID
```

RIGHT JOIN es el mismo concepto que LEFT JOIN, pero al revés. Ahora se muestran todas las filas de la tabla derecha (Pedidos). Las filas de la tabla izquierda (Clientes) se mostrarán solo si hay una coincidencia con las de la derecha. Si existen valores en la tabla derecha pero no en la tabla izquierda, éste se mostrará NULL.

Imaginemos ahora que en la tabla Pedidos tuviéramos un pedido que no tuviera cliente asignado.

PEDIDOS		
PedidoID	ClienteID	Factura
234	4	160
235	2	48
236	3	64
237	4	92
238	NULL	345

Si queremos que la consulta nos devuelva este pedido, aunque no tenga un cliente asociado, usaremos la siguiente consulta:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.ClienteID =
Pedidos.ClienteID
```

Con la que obtendremos el resultado siguiente:

CLIENTES	
Nombre Cliente	PedidoID
Josefa Pérez	234
Lydia Sánchez	235
Tomás Gómez	236
Josefa Pérez	237
NULL	238

EJEMPLO PRÁCTICO

Realiza una consulta que devuelva los nombres de todos los departamentos, aunque éstos no tengan ningún profesor asociado, junto con el nombre del profesor.

Solución:

```
SELECT Profesor.Nombre, Departamento.Nombre
FROM Profesor
RIGHT JOIN Departamento ON Profesor.DeptoID=Departamento.ID
```

VÍDEO DE INTERÉS

Visualiza más sobre las composiciones externas en este vídeo:

Por último, veamos un ejemplo de cómo podríamos simular una composición completa en MySQL:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
LEFT JOIN Pedidos ON Clientes.ClienteID = Pedidos.ClienteID
```

```
UNION

SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.ClienteID = Pedidos.ClienteID
```

El resultado de esta consulta sería el siguiente:

CLIENTES	
Nombre Cliente	PedidoID
Antonio López	NULL
Josefa Pérez	234
Lydia Sánchez	235
Tomás Gómez	236
Josefa Pérez	237
NULL	238

Por lo tanto, nos está devolviendo tanto los registros para los que existe coincidencia en las tablas como para los que no existe esta coincidencia.

Recuerda:

- En una consulta multitabla, las tablas que contienen los datos son designadas en la cláusula FROM.
- Cada fila de resultados es una combinación de datos procedentes de una única fila en cada una de las tablas, y es la única fila que extrae sus datos de esa combinación particular.
- Las consultas multitabla más habituales utilizan las relaciones padre/hijo creadas por las claves primarias y claves ajenas.
- En general, las composiciones pueden construirse comparando cualquier par(es) de columnas de las dos tablas compuestas, utilizando un test de desigualdad o cualquier otro test de comparación.
- Una composición puede ser considerada como el producto de dos tablas del cual se han suprimido algunas de las filas.
- Una tabla puede componerse consigo misma; las autocomposiciones requieren el uso de alias.

Las composiciones externas amplían la composición estándar (interna) reteniendo las filas no emparejadas de las tablas compuestas en los resultados de la consulta.

PARA SABER MÁS

Puedes ampliar información sobre las Composiciones o JOINS en esta web:

6. SUBCONSULTAS

La persona que te está ayudando está atascada con una consulta y le resulta imposible obtener el resultado deseado. Viendo la información que está intentando obtener, rápidamente te das cuenta que la solución más fácil es la de realizar una subconsulta.

La característica de subconsulta de SQL permite utilizar los resultados de una consulta como parte de otra. La característica de subconsulta es menos conocida que la característica de composición de SQL, pero juega un papel importante por tres razones:

- Una sentencia SQL con una subconsulta es frecuentemente el modo más natural de expresar una consulta, ya que se asemeja más a la descripción de la consulta en lenguaje natural.

- Las subconsultas hacen más fácil la escritura de sentencias SELECT, ya que permiten “descomponer una consulta en partes” (la consulta y sus subconsultas) y luego “recomponer las partes”.

- Hay algunas consultas que no pueden ser expresadas en el lenguaje SQL si utilizar una subconsulta.

PARA SABER MÁS

Amplia información sobre las subconsultas en SQL en la web:

UTILIZACIÓN DE SUBCONSULTAS.

Una subconsulta es una consulta que aparece dentro de la cláusula WHERE o HAVING de otra sentencia SQL. Las subconsultas proporcionan un modo eficaz y natural de gestionar peticiones de consultas que se expresan en términos de los resultados de otras consultas.

EJEMPLO PRÁCTICO

Lista las oficinas en donde el objetivo de ventas de la oficina excede a la suma de las cuotas de los vendedores individuales. La petición solicita una lista de oficinas de la Tabla OFICINAS, en donde el valor de la columna OBJETIVO satisface cierta condición. Parece razonable que la sentencia SELECT que expresa la consulta debería ser semejante a ésta:

Solución:

```
SELECT CIUDAD FROM OFICINAS WHERE OBJETIVO > ???
```

El valor "???" necesita ser sustituido, y debería ser igual a "la suma de las cuotas de los vendedores asignados a la oficina de cuestión". ¿Cómo se puede especificar ese valor en la consulta? Se sabe que la suma de las cuotas para una oficina específica (por ejemplo, la oficina número 21) puede ser obtenida con esta consulta:

```
SELECT SUM(CUOTA) FROM REPVENTAS WHERE OFICINA_REP = 21
```

Pero ¿cómo se pueden poner los resultados de esta consulta en la consulta primera sustituyendo a los signos de interrogación? Parecería razonable comenzar con la primera consulta y reemplazar los "???" con la segunda consulta. Del modo siguiente:

```
SELECT CIUDAD FROM OFICINAS WHERE OBJETIVO > (SELECT SUM(CUOTA) FROM
REPVENTAS  WHERE OFICINA_REP = OFICINA)
```

De hecho, ésta es una consulta SQL correctamente construida. Por cada oficina, la "consulta interna" (la subconsulta) calcula la suma de las cuotas para los vendedores que trabajan en esa oficina. La "consulta externa" (la consulta principal) compara el objetivo de la oficina con el total calculado y decide si añadir la oficina a los resultados de la consulta principal. Trabajando conjuntamente, la consulta principal y la subconsulta expresan la petición original y recuperan los datos solicitados de la base de datos.

Las subconsultas SQL aparecen siempre como parte de la cláusula WHERE o la cláusula HAVING. En la cláusula WHERE, ayudan a seleccionar las filas individuales que aparecen en los resultados de la consulta. En la cláusula HAVING, ayudan a seleccionar los grupos de filas que aparecen en los resultados de la consulta. Recuerda:

- Una subconsulta debe producir una única columna de datos como resultado. Esto significa que una subconsulta siempre tiene un único elemento de selección en su cláusula SELECT.

- La cláusula ORDER BY no puede ser especificada en una subconsulta. Los resultados de la subconsulta se utilizan internamente por parte de la consulta principal y nunca son visibles al usuario, por lo que tiene poco sentido ordenarlas de algún modo.

- Una subconsulta no puede ser la UNION de varias sentencias SELECT diferentes; sólo se permite una única sentencia SELECT.

- Los nombres de columna que aparecen en una subconsulta pueden referirse a columnas de tablas de la consulta principal.

6.1. Subconsultas de la cláusula WHERE

Las subconsultas suelen ser utilizadas principalmente en la cláusula WHERE de una sentencia SQL. Cuando aparece una subconsulta en la cláusula WHERE, ésta funciona como parte del proceso de selección de filas.

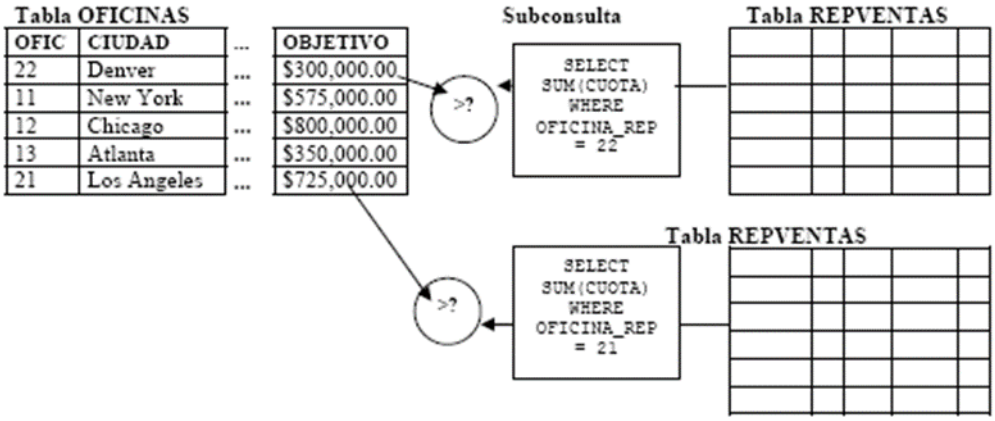
EJEMPLO PRÁCTICO

Lista las oficinas en donde el objetivo de ventas de la oficina excede a la suma de las cuotas de los vendedores individuales.

Solución:

```
SELECT CIUDAD FROM OFICINAS WHERE OBJETIVO > (SELECT SUM(CUOTA) FROM
REPVENTAS WHERE OFICINA_REP = OFICINA)
```

La consulta principal extrae sus datos de la Tabla OFICINAS, y la cláusula WHERE selecciona qué oficinas serán incluidas en los resultados de la consulta. SQL recorre las filas de la Tabla OFICINAS una a una, aplicándoles el test establecido en la cláusula WHERE. La cláusula WHERE compara el valor de la columna OBJETIVO de la fila actual con el valor producido por la subconsulta. Para examinar el valor OBJETIVO, SQL lleva a cabo la subconsulta, determinando la suma de las cuotas para los vendedores de la oficina “actual”. La subconsulta produce un número y la cláusula WHERE compara el número con el valor OBJETIVO, seleccionando o rechazando la oficina actual según la comparación.



Resultado tras la ejecución de la consulta.

REFERENCIAS EXTERNAS.

Dentro del cuerpo de una subconsulta, con frecuencia es necesario referirse al valor de una columna en la fila "actual" de la consulta principal. Considérese una vez más la consulta del apartado anterior.

El papel de la subconsulta en esta sentencia SELECT es calcular la cuota total para los vendedores que trabajan en una oficina particular, específicamente, la oficina que actualmente está siendo examinada por la cláusula WHERE de la consulta principal. La subconsulta lo lleva a cabo explorando la Tabla REPVENTAS. Pero obsérvese que la columna OFICINA en la cláusula WHERE de la subconsulta no se refiere a una columna de la Tabla REPVENTAS; se refiere a una columna de la Tabla OFICINAS, que forma parte de la consulta principal. Conforme SQL recorre cada fila de la Tabla OFICINAS, utiliza el valor OFICINA de la fila actual cuando lleva a cabo la subconsulta. La columna OFICINA en esta subconsulta es un ejemplo de referencia externa. Una referencia externa es un nombre de columna que no se refiere a ninguna de las tablas designadas en la cláusula FROM de la subconsulta en la cual aparece el nombre de la columna. En vez de ello, el nombre de columna se refiere a una columna de una tabla especificada en la Tabla FROM de la consulta principal.

CONDICIONES DE BÚSQUEDA EN SUBCONSULTAS.

Una subconsulta forma parte de una condición de búsqueda en la cláusula WHERE o HAVING. SQL ofrece estas condiciones de búsqueda en subconsultas, además de las descritas en otro tema anterior.

- **Test de comparación subconsulta**. Compara el valor de una expresión con un valor único producido por una subconsulta. Este test se asemeja al test de comparación simple.

- **Test de pertenencia a conjunto subconsulta**. Comprueba si el valor de una expresión coincide con uno del conjunto de valores producido por una subconsulta. Este test se asemeja al test de pertenencia a conjunto simple.

- **Test de existencia**. Examina si una subconsulta produce alguna fila de resultados.

- **Test de comparación cuantificada**. Compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta.

TEST DE COMPARACIÓN SUBCONSULTA (=, <>, <, <=, >, >=).

El test de comparación subconsulta es una forma modificada del test de comparación simple. Compara el valor de una expresión producido por una subconsulta, y devuelve un resultado TRUE si la comparación es cierta. Este test se utiliza para comparar un valor de la fila que está siendo examinada con un valor único producido por una subconsulta.

La subconsulta del ejemplo recupera el objetivo de ventas de la oficina de Atlanta. El valor se utiliza entonces para seleccionar los vendedores cuyas cuotas son superiores o iguales al objetivo.

EJEMPLO PRÁCTICO

Lista los vendedores cuyas cuotas son iguales o superiores al objetivo de la oficina de ventas de Atlanta.

Solución:

```
SELECT NOMBRE FROM REPVENTAS WHERE CUOTA >= (SELECT OBJETIVO FROM
OFICINAS WHERE CIUDAD = 'Atlanta')
```

El test de comparación subconsulta ofrece los mismos seis operadores de comparación (=, <>, <, <=, >, >=) disponibles con el test de comparación simple. La subconsulta especificada en este test debe producir una única fila de resultados. Si la subconsulta produce múltiples filas, la comparación no tiene sentido y SQL informa de una condición de error. Si la subconsulta no produce filas o produce un valor NULL, el test de comparación devuelve NULL.

EJEMPLO PRÁCTICO

Lista todos los productos del fabricante ACI para los cuales las existencias superan a las existencias del producto ACI-41004.

Solución:

```
SELECT DESCRIPCION, EXISTENCIAS FROM PRODUCTOS WHERE ID_FAB = 'ACI'
AND EXISTENCIAS > (SELECT EXISTENCIAS FROM PRODUCTOS
WHERE ID_FAB = 'ACI' AND ID_PRODUCTO = '41004')
```

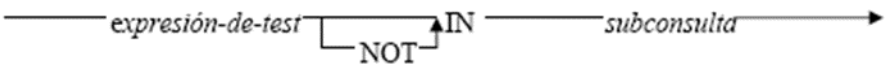
DESCRIPCIÓN	EXISTENCIAS
Artículo Tipo 3	207
Artículo Tipo 1	277
Artículo Tipo 2	167

Obsérvese que el test de comparación subconsulta permite una subconsulta únicamente en el lado derecho del operador de comparación.

Esta composición está permitida: A < (subconsulta). Pero esta otra no está permitida: (subconsulta) > A

TEST DE PERTENENCIA A CONJUNTO (IN).

El test de pertenencia a conjunto subconsulta (IN) es una forma modificada del test de pertenencia a conjunto simple. Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve un resultado TRUE si el valor coincide con uno de los valores de la columna.



EJEMPLO PRÁCTICO

Lista los vendedores que no trabajan en oficinas dirigidas por Larry Fitch (empleado 108).

Solución:

```
SELECT NOMBRE FROM REPVENTAS WHERE OFICINA_REP NOT IN (SELECT
OFICINA
FROM OFICINAS WHERE DIR = 108)
```

NOMBRE
Bill Adams
Mary Jones
Sam Clark
Bob Smith
Dan Roberts
Paul Cruz

La subconsulta produce una columna de valores, y la cláusula WHERE de la consulta principal, comprueba si un valor de una fila de la consulta principal coincide con uno de los valores de la columna.

TEST DE EXISTENCIA (EXISTS).

El test de existencia (EXISTS) comprueba si una subconsulta produce alguna fila de resultados. No hay test de comparación simple que se asemeje al test de existencia; solamente se utiliza con subconsultas. He aquí un ejemplo de una petición que se puede expresar sencillamente utilizando un test de existencia.

EJEMPLO PRÁCTICO

Lista los productos para los cuales se ha recibido un pedido de $25.000 o más". Esta misma consulta también se podría expresar de esta otra forma: "Lista los productos para los cuales existe al menos un pedido en la Tabla PEDIDOS a) que se refiere al producto en cuestión y b) que tiene un importe de al menos $25.000". La sentencia SELECT utilizada para recuperar la lista solicitada de productos se asemeja a la petición así expresada:

Solución:

```
SELECT DISTINCT DESCRIPCION FROM PRODUCTOS WHERE EXISTS (SELECT
NUM_PEDIDO FROM PEDIDOS WHERE PRODUCTO = ID_PRODUCTO  AND FAB =
ID_FAB
AND IMPORTE >= 25,000.00)
```

DESCRIPCIÓN
500-1b Brace
Left Hinge
Right Hinge
Widget Remover

Conceptualmente, SQL procesa esta consulta recorriendo la Tabla PRODUCTOS y efectuando la subconsulta para cada producto. La subconsulta produce para una columna que contiene los números de pedidos de aquellos pedidos del producto "actual" que superan $25,000.00.

Si hay alguno de tales pedidos (es decir, si la columna no está vacía), el test EXISTS es TRUE. Si la subconsulta no produce filas, el test EXISTS es FALSE. El test EXISTS no puede producir un valor NULL.

Se puede invertir la lógica del test EXISTS utilizando la forma NOT EXISTS. En este caso, el test es TRUE si la subconsulta no produce filas, y FALSE en caso contrario.

Se puede observar, que la condición de búsqueda EXISTS no utiliza realmente los resultados de la subconsulta. Simplemente comprueba si la búsqueda produce algún resultado. Por esta razón, SQL suaviza la regla de que "las subconsultas deben devolver una única columna de datos", y permite utilizar la forma SELECT * en la subconsulta de un test EXISTS. La subconsulta podría por tanto haber sido escrita:

```
SELECT DESCRIPCION FROM PRODUCTOS WHERE EXISTS (SELECT *
FROM PEDIDOS
```

```
WHERE PRODUCTO = ID_PRODUCTO AND FAB = ID_FAB AND IMPORTE >=
25000.00)
```

En la práctica, la subconsulta en un test EXISTS se escribe utilizando la notación SELECT *.
Obsérvese que la subconsulta incluye una referencia externa a una columna de la tabla de la consulta principal. En la práctica, la subconsulta de un test EXISTS siempre contienen una referencia externa que "enlaza" la subconsulta a la fila que actualmente está siendo examinada por la consulta principal.

TEST CUANTIFICADOS (ANY y YALL).

La versión subconsulta del test IN comprueba si un valor de dato es igual a algún valor en una columna de los resultados de la subconsulta. SQL proporciona dos test cuantificados, ANY y ALL, que extienden esta noción a otros operadores de comparación, tales como mayor que (>) y menor que (<). Ambos test comparan un valor de dato con la columna de valores producidos por una subconsulta.

El TEST ANY *

El test ANY se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con cada valor de datos de la columna, uno cada vez. Si alguna de las comparaciones individuales produce un resultado TRUE, el test ANY devuelve un resultado TRUE.

EJEMPLO PRÁCTICO

Lista los vendedores que han aceptado un pedido que represente más del 10 por 100 de su cuota:

Solución:

```
SELECT NOMBRE FROM REPVENTAS WHERE (0.1 * CUOTA) < ANY (SELECT
IMPORTE
FROM PEDIDOS WHERE REP = NUM_EMPL)
```

DESCRIPCIÓN
Sam Clark
Larry Fitch
Nancy Angelli

Conceptualmente, la consulta principal examina cada fila de la Tabla REPVENTAS, una a una. La subconsulta encuentra todos los pedidos aceptados por el vendedor "actual" y devuelve una columna que contiene el importe de esos pedidos. La cláusula WHERE de la consulta principal calcula entonces el diez por ciento de la cuota del vendedor actual y la utiliza como valor del test, comparándolo con todos los importes de pedidos producidos por la subconsulta.

Si hay algún importe que exceda al valor de test calculado, el test < ANY devuelve TRUE y el vendedor queda incluido en los resultados de la consulta. Si no, el vendedor no se incluye en los resultados de la consulta.

- Si la subconsulta produce una columna vacía de resultados, el test ANY devuelve FALSE.

- Si el test de comparación es TRUE para al menos uno de los valores de datos en la columna, entonces la condición de búsqueda ANY devuelve TRUE.

- Si el test de comparación es FALSE para todos de los valores de datos de la columna, entonces la condición de búsqueda ANY devuelve FALSE.

- Si el test de comparación no es TRUE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ANY devuelve NULL.

- En esta situación, no se puede concluir si existe un valor producido por la subconsulta para el cual el test de comparación sea cierto; puede haberlo o no, dependiendo de los valores "correctos" de los datos NULL (desconocido).

El operador de comparación ANY puede ser engañoso al utilizarlo en la práctica, especialmente junto con el operador de comparación desigualdad (<>).

EJEMPLO PRÁCTICO

Lista los nombres y edades de todas las personas en el equipo de ventas que no dirigen una oficina:

Solución:

```
SELECT NOMBRE, EDAD FROM REPVENTAS WHERE NUM_EMPL <> ANY (SELECT DIR
FROM OFICINAS)
```

La subconsulta "`SELECT DIR FROM OFICINAS`" produce obviamente los números de empleados que son directores, y por tanto la consulta parece decir: *"Halla los vendedores que no son directores de ninguna oficina".*

Pero esto no es lo que la consulta dice. Lo que dice en realidad es: *"Halla cada uno de los vendedores que, para alguna oficina, no es el director de esa oficina".*

Naturalmente para cualquier vendedor, es posible hallar alguna oficina en donde ese vendedor no es el director. Los resultados de la consulta incluirían a todos los vendedores, y por tanto fallaría en responder a la cuestión que le fue propuesta. La consulta correcta es:

```
SELECT NOMBRE, EDAD
FROM REPVENTAS
WHERE NOT (NUM_EMPL = ANY (SELECT DIR FROM OFICINAS))
```

Siempre se puede transformar una consulta con un test ANY en una consulta con un test EXISTS trasladando la comparación al interior de la condición de búsqueda de la subconsulta. He aquí una forma alternativa de la consulta, utilizando el test EXISTS.

```
SELECT NOMBRE, EDAD FROM REPVENTAS
```

```
WHERE NOT EXISTS (SELECT * FROM OFICINAS WHERE NUM_EMPL =
DIR)
```

EL TEST ALL *.

Al igual que el test ANY, el test ALL se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores de datos producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con todos y cada uno de los valores de datos de la columna. Si todas las comparaciones individuales producen un resultado TRUE, el test ALL devuelve un resultado TRUE.

EJEMPLO PRÁCTICO

Lista oficinas y objetivos en donde todos los vendedores tienen ventas que superan el 50 % del objetivo de la oficina:

Solución:

```
SELECT CIUDAD, OBJETIVO FROM OFICINAS WHERE (0.50 * OBJETIVO) < ALL
(SELECT VENTAS FROM REPVENTAS  WHERE OFICINA_REP = OFICINA)
```

CIUDAD	OBJETIVO
Denver	$300,000.00
New York	$575,000.00

Conceptualmente, la consulta principal examina cada fila de la Tabla OFICINAS, una a una. La subconsulta encuentra todos los vendedores que trabajan en la oficina "actual" y devuelve una columna que contiene las ventas correspondientes a cada vendedor. La cláusula WHERE de la consulta principal calcula entonces el 50% del objetivo de la oficina y lo utiliza como valor de test, comparándolo con todos los valores de ventas producidos por la subconsulta.

Si todos los valores de ventas exceden a valor de test calculado, el test < ALL devuelve >TRUE y la oficina se incluye en los resultados de la consulta. Si no, la oficina no se incluye en los resultados de la consulta.

- Si la consulta produce una columna vacía de resultados, el test ALL devuelve TRUE.

- Si el test de comparación es TRUE para todos y cada uno de los valores de datos en la columna, entonces la condición de búsqueda ALL devuelve TRUE.

- Si el test de comparación es FALSE para algún valor de dato en la columna entonces la condición de búsqueda ALL devuelve FALSE.

- Si el test de comparación no es FALSE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ALL devuelve NULL.

Los errores sutiles que pueden ocurrir con el test ANY cuando se combina con el operador de comparación desigualdad (<>) también ocurren con el test ALL. Como con el test ANY, el test ALL siempre puede convertirse a un test EXISTS equivalente mediante el traslado de la comparación al interior de la subconsulta.

6.2. Subconsultas en la cláusula HAVING

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula WHERE, también pueden utilizarse en la cláusula HAVING de una consulta. Cuando una subconsulta aparece en la cláusula HAVING, funciona como parte de la selección de grupo de filas efectuada por la cláusula HAVING.

EJEMPLO PRÁCTICO

Lista los vendedores cuyo tamaño de pedido medio para productos fabricados por ACI es superior al tamaño de pedido medio global:

Solución:

```
SELECT NOMBRE, AVG(IMPORTE) FROM REPVENTAS, PEDIDOS WHERE NUM_EMPL =
REP
 AND FAB = 'ACI' GROUP BY NOMBRE  HAVING AVG(IMPORTE) > (SELECT
AVG(IMPORTE) FROM PEDIDOS)
```

NOMBRE	AVG (IMPORTE)
Sue Smith	$15,000.00
Tom Snyder	$22,500.00

La subconsulta calcula el "tamaño de pedido medio global". Se trata de una sencilla subconsulta que contiene referencias externas, por lo que SQL puede calcular el promedio una vez y utilizarlo luego repetidamente en la cláusula HAVING. La consulta general recorre la Tabla PEDIDOS, hallando todos los pedidos de productos ACI, y los agrupa por vendedor. La cláusula HAVING comprueba entonces cada grupo de filas para ver si el tamaño de pedido medio de ese grupo es superior al promedio de todos los pedidos, calculado con antelación. Si es así, el grupo de filas es retenido; si no, el grupo de filas es descartado. Finalmente, la cláusula SELECT produce una fila resumen por cada grupo, mostrando el nombre del vendedor y el tamaño de pedido medio para cada uno.

7. TRANSACCIONES

Para terminar con la formación de la persona en prácticas, te parece oportuno formarle mínimamente en dos conceptos importantes como son las transacciones y la concurrencia.

Una transacción es una secuencia de operaciones que se ejecutan como una unidad indivisible, lo que significa que, o bien se completan con éxito todas las operaciones de la transacción, o bien no se aplica ninguna. Estas operaciones pueden ser actualizaciones, inserciones, consultas o eliminaciones.

Una transacción empieza ejecutando una única operación como, por ejemplo, la modificación de los registros de una tabla. Si se ejecutaran otras operaciones como parte de la misma transacción, estas deberán ejecutarse como una única unidad atómica o indivisible. Si alguna de esas operaciones falla, la transacción completa se deshace (**Rollback**) y los datos se restablecen a su estado anterior. Si, por el contrario, todas las operaciones tienen éxito, se confirmará la transacción (**Commit**) y los cambios realizados serán permanentes.

El concepto clave de las transacciones es lo que se conoce como propiedades ACID, que se refiere a las características de Atomicidad, Consistencia, Aislamiento y Durabilidad

- **Atomicidad:** es la propiedad que garantiza que todas las operaciones de una transacción se ejecuten como una unidad atómica, asegurándose de que los datos no queden en un estado inconsistente.

- **Consistencia:** los cambios realizados por una transacción deben cumplir con todas las reglas de integridad y con las restricciones de la base de datos.

- **Aislamiento:** cada transacción se debe ejecutar de manera aislada y no interferir con otras transacciones.

- **Durabilidad:** una vez se ha confirmado la transacción, todos los cambios realizados se mantendrán de manera permanente.

La manera de iniciar y finalizar una transacción varía en función del SGBD. En MySQL, se puede iniciar una transacción con la sentencia **START TRANSACTION** y se puede finalizar con el uso de las sentencias **COMMIT** (para confirmar) o **ROLLBACK** (para deshacer).

```
START TRANSACTION;
INSERT INTO Empleados (id, nombre) VALUES (1, 'Juan');
```

```
SELECT @A:=SUM(salario) FROM Empleados WHERE id=1;
UPDATE Pedidos SET Total=@A WHERE id=1;
COMMIT;
```


ENLACE DE INTERÉS

En esta web se recoge información sobre qué es una transacción:

PARA SABER MÁS

Si quieres saber más información sobre las Transacciones consulta este enlace:

7.1 Concurrencia

Las transacciones y la concurrencia son dos conceptos fundamentales en las bases de datos relacionales, ya que son muy importantes para garantizar la integridad de los datos y el rendimiento de las operaciones en entornos multiusuario.

La concurrencia hace referencia a la capacidad o posibilidad de que varias transacciones puedan acceder y modificar la base de datos de forma simultánea. Esto cobra especial relevancia en entornos multiusuario, donde es bastante probable que existan varias transacciones ejecutándose al mismo tiempo. Como se comentó en el apartado anterior, las propiedades ACID deben garantizar que las transacciones no interfieran entre sí y que se mantenga la integridad de los datos.

Por ejemplo, si un usuario hace una consulta al mismo tiempo que otro usuario modifica una de las filas que aparecerán en dicha consulta, ¿qué valor verá el usuario que realiza la consulta? Podría darse el caso de que viera el valor de antes de la modificación, por lo tanto, estaría viendo un resultado que no es el correcto.

En MySQL, la concurrencia se gestiona utilizando técnicas de bloqueo y control de transacciones. Algunas de ellas son:

- **Bloqueo de lectura y escritura:** El bloqueo de lectura evita que otros procesos realicen modificaciones en la tabla, mientras que el bloqueo de escritura evita tanto las modificaciones como la lectura de la tabla por parte de otros procesos.

Adquirir bloqueo de lectura

```
START TRANSACTION;
SELECT stock FROM productos WHERE id = 1 FOR SHARE;
UPDATE productos SET stock = stock - 1 WHERE id = 1;
   Confirmar la transacción
COMMIT;
```

En este ejemplo, la transacción comienza con START TRANSACTION. Luego, se usa la sentencia SELECT con la cláusula FOR SHARE para adquirir un bloqueo de lectura en la fila del producto con id igual a 1. Esto permite que otras transacciones también puedan leer los datos, pero evita que otras transacciones adquieran un bloqueo de escritura en esa fila.

A continuación, realizamos cualquier operación adicional o cálculo necesario. Para terminar, utilizamos la sentencia UPDATE para actualizar el stock. Finalmente, realizamos un COMMIT para confirmar la transacción.

También es posible usar la opción **FOR UPDATE,** por ejemplo:

```
SELECT * FROM usuarios WHERE estado = 'activo' FOR UPDATE;
```

En este ejemplo, la sentencia SELECT selecciona todas las filas de la tabla "usuarios" donde el estado sea "activo" y adquiere un bloqueo de escritura (FOR UPDATE) en esas filas.

- **Niveles de aislamiento:** MySQL ofrece diferentes niveles de aislamiento de transacciones, que controlan cómo se ven los cambios realizados por una transacción en otras transacciones concurrentes. Los niveles de aislamiento incluyen:

- *READ UNCOMMITTED*: en este nivel, una transacción puede leer datos que aún no han sido confirmados (commit) por otras transacciones, lo que podría conllevar la lectura de datos inconsistentes.

- *READ COMMITTED*: es el nivel de aislamiento por defecto en la mayoría de los SGBD. Una transacción solo puede leer datos que han sido confirmados (commit).

- *REPEATABLE READ*: en este nivel, una transacción garantiza que todas las lecturas de datos sean consistentes y repetibles a lo largo de la duración de la transacción, incluso si otros procesos están realizando modificaciones en los datos

- *SERIALIZABLE*: es el nivel de aislamiento más alto. Garantiza que las transacciones se ejecuten como si fueran secuenciales, incluso en entornos con transacciones concurrentes.

PARA SABER MÁS

Puedes ampliar información sobre control de concurrencia en esta dirección web:

RESUMEN FINAL

En esta unidad hemos visto cómo realizar consultas a más de una tabla en base de datos. Para ello, hemos tratado los conceptos de columnas de emparejamiento y los diferentes tipos de composiciones existentes (interna, externa y completa). Otra forma de referirse a estas composiciones es: **INNER JOIN** (interna) **LEFT** y **RIGHT JOIN** (externa) y **FULL OUTER JOIN** (completa).

Es importante recordar que, en MySQL, no existe FULL OUTER JOIN y la manera de emularlo es hacer un UNION de una consulta con LEFT JOIN y la misma consulta con RIGHT JOIN.

También hemos aprendido el concepto de subconsulta y hemos visto que pueden realizarse tanto dentro de la cláusula WHERE como dentro de la cláusula HAVING.
Y, para terminar, también hemos presentado los conceptos de transacción y concurrencia, con algunos ejemplos de cómo gestionar ésta última como, por ejemplo, el bloqueo de lectura y escritura.