Tema 11

Otros elementos de las BdD Relacionales.

11.1. Vistas.

Una vista no es más que una consulta almacenada a fin de utilizarla tantas veces como se desee.

Una vista no contiene datos sino que contendrá la consulta pertinente que se ejecutará cada vez que se use, mostrando los datos que haya en ese momento en la o las tablas subyacentes.

Las vistas se emplean para:

- Ayudarnos en consultas complejas permitiéndonos hacer un paso intermedio.
- Proporcionarnos una tabla (temporal) con datos procedentes de dos o más.
- Mostrar los datos estrictamente necesarios para una situación determinada.

Hay dos tipos de vistas:

- Simples: los datos proceden de una sola tabla y no contienen funciones de agrupación. Permiten realizar operaciones DML sobre ellas.
- Complejas: obtienen datos de varias tablas, pueden utilizar funciones de agrupación. No siempre permiten operaciones DML.

11.1.1. Creación de vistas.

Sintáxis:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

11.1.2. Modificación de vistas.

Se puede realizar con el comando habitual:

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

O usando la opción `OR REPLACE` en la sentencia `CREATE`.

11.1.3. Eliminación de vistas.

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
```

11.1.4. Consultar las vistas existentes.

En la tabla `VIEWS` del esquema `INFORMATION_SCHEMA`:

```sql
SELECT
   *
FROM
   INFORMATION_SCHEMA.VIEWS
WHERE
   table_schema = 'empleados';
```

11.1.5. Ejemplos de vistas.

Vista sobre una tabla.

```sql
CREATE OR REPLACE VIEW nomina_empleados_ventas AS
   SELECT
      id_empleado,
      nombre,
      apellido,
      salario,
      (salario * comision) AS "Comisión Calculada",
      (salario + salario * comision) AS Total
   FROM
      empleados
   WHERE
      id_departamento = 60;
```

Vista sobre varias tablas.

Al establecer un conjunto de nombres entre paréntesis detrás del nombre de la vista estamos estableciendo un alias para cada columna del SELECT.

```
CREATE OR REPLACE VIEW resumen_departamentos
      (nombre , media_salarios , total_salarios) AS
   SELECT
      nombre_departamento, AVG(salario), SUM(salario)
   FROM
      empleados E,
      departamentos D
   WHERE
      E.id_departamento = D.id_departamento
   GROUP BY nombre_departamento;
```

11.1.6. Operaciones sobre vistas.

Se pueden usar como una tabla "normal" para realizar consultas.

```
SELECT * FROM nomina_empleados_ventas;
```

```
SELECT * FROM nomina_empleados_ventas WHERE total > 12000;
```

```
SELECT
   *
FROM
   resumen_departamentos R,
   departamentos D
WHERE
   R.nombre = D.nombre_departamento;
```

11.1.6.1. Operaciones DML sobre vistas.

No es posible ejecutar instrucciones DML sobre vistas que:

- Utilicen funciones de grupo (SUM, AVG,...).
- Usen `GROUP BY` o `DISTINCT`.
- Posean columnas con cálculos (p.ej.: (salario * comision)).
- Si en las tablas referenciadas en la consulta `SELECT` hay campos `NOT NULL` que no aparecen en la consulta.

- Utilizan `UNION` o `UNION ALL`.
- Algunos tipos de `JOIN` o subconsultas. No entraremos en detalles.

Es un aspecto importante pero no le dedicaremos más tiempo. Puedes obtener información más específica aquí.

11.2. WITH.

La clausula `WITH` crea una tabla temporal de forma similar a una vista pero solo existe durante la duración de la sentencia actual.

Se introdujo en SQL (en 1999) para simplificar las subconsultas o consultas multitabla.

Observa que solo hay un punto y coma al final del todo porque es la misma consulta.

```
WITH salario_departamentos
AS
(SELECT
   SUM(salario) as Total
FROM empleados
GROUP BY id_departamento)

SELECT
   AVG(Total) media_sal_departamentos
FROM salario_departamentos;
```

Si por ejemplo no hubiéramos querido almacenar la vista en uno de los ejemplos que hicimos, podríamos haber usado `WITH` (se ha modificado el nombre para evitar conflictos):

```
WITH resumen_depart (nombre , media_salarios ,
   total_salarios) AS
(SELECT
   nombre_departamento, AVG(salario), SUM(salario)
FROM
   empleados E,
   departamentos D
WHERE
   E.id_departamento = D.id_departamento
GROUP BY nombre_departamento)

SELECT
   *
FROM
   resumen_depart R,
   departamentos D
WHERE
   R.nombre = D.nombre_departamento;
```

Al igual que las subconsultas, puedes usar `WITH` con operaciones DML. Tienes algunos ejemplos aquí.

Ejercicios 11.1

Los primeros ejercicios son sobre la Base de datos: jardinería. del apéndice D: Otras bases de datos.

1. Crea una vista que se llame listado_pagos_clientes todos los clientes y los pagos que ha realizado cada uno de ellos. La vista deberá tener las siguientes columnas: nombre y apellidos del cliente concatenados, teléfono, ciudad, pais, fecha_pago, total del pago, id de la transacción.
2. Crea una vista que se llame listado_pedidos_clientes que muestre un listado donde aparezcan todos los clientes y los pedidos que ha realizado cada uno de ellos. La vista deberá tener las siguientes columnas: nombre y apellidos del cliente concatendados, teléfono, ciudad, pais, código del pedido, fecha del pedido, fecha esperada, fecha de entrega y la cantidad total del pedido, que será la suma del producto de todas las cantidades por el precio de cada unidad, que aparecen en cada línea de pedido.
3. Usa las vistas que has creado en los pasos anteriores para devolver un listado de los clientes de la ciudad de Madrid que han realizado pagos.

4. Utiliza las vistas que has creado en los pasos anteriores para devolver un listado de los clientes que todavía no han recibido su pedido.
5. Utiliza las vistas que has creado en los pasos anteriores para calcular el número de pedidos que se ha realizado cada uno de los clientes.
6. Usa las vistas que has creado en los pasos anteriores para calcular el valor del pedido máximo y mínimo que ha realizado cada cliente.
7. Elimina las vistas que has creado en los pasos anteriores.

   LOS SIGUIENTES EJERCICIOS SON SOBRE LA BDD EMPLEADOS.

8. Repite las operaciones que hiciste con subconsultas en los ejercicios 10.2.3, 10.3.3 y 10.3.6 usando `WITH` en lugar de una subconsulta.
9. Repite las operaciones que hiciste con subconsultas en los ejercicios 10.2.6, 10.2.7 y 10.2.8 usando `WITH` en lugar de una subconsulta.
10. ¿Puedes realizar los ejercicios 10.2.4 y 10.2.5 usando `WITH` en lugar de subconsultas? ¿por qué?

11.3. Índices.

Los índices se usan para acelerar las operaciones de consulta y ordenación sobre los campos a los que hace referencia. Puedes obtener información más precisa aquí.

Los índices se comportan como punteros sobre las filas de la tabla y nos permiten determinar rápidamente cuáles son las filas que cumplen la condición de la cláusula `WHERE`.

Objetos independientes en la BdD (no se almacena en la misma tabla a la que hacen referencia).

Se crean sobre una o más columnas y establecen una lista ordenada sobre ellas:

- Operaciones más rápidas.
- Mantenimiento más costoso: cualquier modificación implica actualizar el índice –> no conviene tener demasiados índices porque puede afectar negativamente al rendimiento.

Muchos índices se crean de manera implícita al aplicar las restricciones `PRIMARY KEY`, `UNIQUE` y `FOREIGN KEY`.

IMPORTANTE:

El uso de índices correctamente es una poderosísima arma de optimización. Queda fuera del alcance de este curso pero este sitio te proporciona información muy útil para tu futuro.

11.3.1. Creación de índices.

Para crear índices de forma explícita sobre otra columnas.

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL]
      INDEX index_name
   [index_type]
   ON tbl_name (index_col_name, ... )
   [index_option] ...

index_col_name:
   col_name [(length)] [ASC | DESC]

index_option:
   KEY_BLOCK_SIZE [=] value
   | index_type
   | WITH PARSER parser_name
   | COMMENT 'string'

index_type:
   USING {BTREE | HASH}
```

Tipos:

- KEY o INDEX: índice normal que no impone la condición de único (ni ninguna otra), solo sirve para agilizar las consultas.
- UNIQUE: como el anterior pero además exige que el valor en la columna o columnas no se repita en dos o más filas.

  Dependiendo de la definición de la tabla se admiten valores NULL o no por lo que podría repetirse dicho valor.
- PRIMARY: clave primaria.
- FULLTEXT: con un uso mucho más específico. Únicamente se aplica a búsquedas "full text" usando la clausula MATCH() / AGAINST() que nosotros no vemos.

Por ejemplo si queremos acelerar la búsqueda sobre el nombre del departamento.

```
CREATE INDEX idx_nom_depart
   ON departamentos(nombre_departamento);
```

Podemos hacerlo de tal manera que incorpore la opción `UNIQUE` para la columna, evitando valores repetidos.

```
CREATE UNIQUE INDEX idx_nom_depart
   ON departamentos(nombre_departamento);
```

O incluir más de una columna.

```
CREATE INDEX idx_ape_nom_emple
   ON empleados(apellido, nombre);
```

Este último índice se aplicará en las consultas por apellido y nombre o en las consultas por apellido, pero no en las que únicamente consulten por nombre.

No se pueden crear índices sobre vistas.

La decisión de crear un índice o no depende de cada caso, pero vamos a ver unas indicaciones generales:

Factores que favorecen la creación de un índice:

- Campos sobre los que se realicen muchas consultas (aparezcan frecuentemente en `WHERE`, `GROUP BY` u `ORDER BY`) y contengan muchos valores.
- Campos con muchos valores nulos.

Factores que indican que es mejor no crear un índice:

- La tabla que contiene el campo no tenga muchas filas.
- No aparezcan a menudo en las consultas.
- El campo o la tabla a la que pertenece se actualizan muy frecuentemente.
- Se utilizan muy habitualmente en expresiones o cálculos.

Es posible crear índices en las sentencias `CREATE TABLE` y `ALTER TABLE`.

11.3.2. Eliminación de índices.

```
DROP INDEX index_name ON tbl_name
   [algorithm_option | lock_option] ...

algorithm_option:
   ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
   LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Por ejemplo:

```
DROP INDEX idx_nom_depart ON departamentos;
```

```
DROP INDEX idx_ape_nom_emple ON empleados;
```

En algunos SGBD (p.ej. Oracle) no es necesario especificar el nombre de la tabla por lo que se haría:

```
DROP INDEX idx_ape_nom_emple;
```

11.3.3. Consultar los índices.

Podemos consultar los índices de una tabla:

```
SHOW INDEXES FROM empleados;
```

Observa cómo hemos añadido un índice pero existen los que se crean de manera implícita. (Figura 11.1):

11.3.4. EXPLAIN.

Permite obtener información sobre cómo se realiza un consulta:

Table	Non_unique	Key_name	Seq_in_index	Column_name
empleados	0	PRIMARY	1	id_empleado
empleados	1	empleados_fk1	1	id_director
empleados	1	empleados_fk2	1	id_departamento
empleados	1	empleados_fk3	1	id_trabajo
empleados	1	idx_ape_nom_emple	1	apellido
empleados	1	idx_ape_nom_emple	2	nombre

Figura 11.1: Índices de la tabla empleados.

```
EXPLAIN SELECT
   nombre, apellido, salario, id_departamento
FROM
   empleados
WHERE
   apellido LIKE 'G%';
```

Si ejecutas la consulta anterior antes y después de crear el índice, verás como en un caso indica que no se está usando y en el otro sí.

Recuerda que creamos el índice sobre apellido y nombre era únicamente el segundo criterio, lo que hacía que si filtrábamos por él, no se aplicara el índice.

Puedes comprobarlo usando:

```
EXPLAIN SELECT
   nombre, apellido, salario, id_departamento
FROM
   empleados
WHERE
   nombre LIKE 'G%';
```

Las columnas que nos devuelve son:

- **id**: número secuencial que identificará cada una de las tablas que se procesan en la consulta realizada.
- **select_type**: tipo de SELECT que se ha ejecutado. En función del tipo de SELECT nos permitirá identificar qué tipo de consulta se trata. Existen varios tipos distintos: SIMPLE, PRIMARY, UNION, DERIVED...
- **table**: muestra el nombre de la tabla a la que se refiere la información de la fila.

También, puede ser alguno de los siguientes valores:

- tabla resultante de la unión de las tabas cuyos campos id son M y N.
- tabla resultante de una tabla derivada cuyo id es N. Una tabla derivada puede ser, por ejemplo, una subconsulta en la cláusula `FROM`.
- tabla resultante de una subconsulta materializada cuyo id es N.

- **partitions**: aparece pero no se muestra el resultado en nuestra versión de MySQL.
- **type**: muestra el tipo de unión que se ha empleado. Sin entrar en detalles, puedes ver los tipos aquí.
- **possible_keys**: muestra qué índices puede escoger MySQL para buscar las filas de la tabla. Si esta columna es `NULL`, significa que no hay índices que puedan ser usados para mejorar la consulta. Puede ser interesante examinar las columnas empleadas en el `WHERE` para analizar si nos podría interesar crear un índice por alguna otra columna.
- **key**: muestra el índice que MySQL ha decido usar para ejecutar la consulta. Es posible que el nombre del índice no esté presente en la lista de possible_keys.
- **ref**: qué columnas o constantes son comparadas con el índice especificado en la columna key.
- **rows**: muestra el número de filas que MySQL cree que deben ser examinadas para ejecutar la consulta. Este número es aproximado.
- **filtered**: muestra el porcentaje estimado de filas que serán filtradas por la condición de la consulta. El valor máximo es 100, que indica que no se ha filtrado ninguna fila.
- **Extra**: información adicional sobre cómo MySQL ejecuta la consulta.

11.4. Secuencias.

Una secuencia sirve para generar automáticamente números distintos.

Se utilizan para generar valores para claves o identificadores cuando no hemos asignado un valor concreto a cada fila (como pasaría con NIF o número de socio).

Las secuencias se almacenan independientemente de la tabla, por lo que la misma secuencia se puede utilizar para más de una tabla.

MySQL 8 no permite usar secuencias pero podemos obtener un resultado similar utilizando `AUTO_INCREMENT` tal y como vimos en DDL.

Ejercicios 11.2

LOS SIGUIENTES EJERCICIOS SON SOBRE LA BDD JARDINERÍA.

1. Consulta cuáles son los índices que hay en la tabla `producto` utilizando las diferentes opciones vistas en clase **a lo largo del curso**.
2. Crea un índice de tipo `INDEX` compuesto por las columnas `apellido_contacto` y `nombre_contacto` de la tabla `cliente`.

   Una vez creado el índice del ejercicio anterior realiza las siguientes consultas haciendo uso de `EXPLAIN`:

   - Busca el cliente Javier Villar. ¿Cuántas filas se han examinado hasta encontrar el resultado?
   - Busca el ciente anterior utilizando solamente el apellido Villar. ¿Cuántas filas se han examinado hasta encontrar el resultado?
   - Busca el ciente anterior utilizando solamente el nombre Javier. ¿Cuántas filas se han examinado hasta encontrar el resultado? ¿Qué ha ocurrido en este caso?
3. Haciendo uso de `EXPLAIN` para obtener información sobre cómo se están realizando las consultas y determina cuál de las dos consultas realizará menos comparaciones para encontrar el producto que estamos buscando. ¿Cuántas comparaciones se realizan en cada caso? ¿Por qué?:

   ```sql
   SELECT *
   FROM producto
   WHERE codigo_producto = 'OR-114';
   ```

   ```sql
   SELECT *
   FROM producto
   WHERE nombre = 'Evonimus Pulchellus';
   ```
4. Queremos saber optimizar las siguientes consultas. ¿Cuál de las dos sería más eficiente?. Se recomienda hacer uso de `EXPLAIN` para obtener información sobre cómo se están realizando las consultas.


```sql
SELECT AVG(total)
FROM pago
WHERE YEAR(fecha_pago) = 2008;
```

```sql
SELECT AVG(total)
FROM pago
WHERE fecha_pago ⩾ '2008-01-01' AND
   fecha_pago ⩽ '2008-12-31';
```

5. Optimiza la siguiente consulta creando índices cuando sea necesario. Se recomienda hacer uso de EXPLAIN para obtener información sobre cómo se están realizando las consultas.

```sql
SELECT *
FROM cliente INNER JOIN pedido
ON cliente.codigo_cliente = pedido.codigo_cliente
WHERE cliente.nombre_cliente LIKE 'A%';
```

11.5. Bibliografía.

Algunos ejercicios de este tema los he obtenido en el material de José Juan Sánchez Hernández al que agradezco que comparta su trabajo.

- Structured Query Language. Wikibook. Wikipedia.
- MySQL 8.0 Reference Manual.
- José Juan Sánchez Hernández (2021/2022). Bases de datos / Gestión de Bases de datos
- Gestión de Bases de Datos. IES Luis Vélez de Guevara - Departamento de Informática.
- Iván López Montalbán, Manuel de Castro Vázquez, John Ospino Rivas (2022). Bases de Datos (2ª Edición) . Garceta.
- What Is the WITH Clause in SQL?. Zahin Rahman (LearnSQL).
- Uso de la sentencia EXPLAIN en MySQL. Rafael Vindel Amor (adictosaltrabajo).

