Vistas e Indices en Bases de Datos MySQL

Vistas en Bases de Datos (MySQL)

Introducción

Las vistas en MySQL son una herramienta poderosa que permite a los usuarios interactuar con los datos de una manera más sencilla y segura. Una vista es una tabla virtual cuyo contenido se define mediante una consulta. A diferencia de una tabla, una vista no almacena datos físicamente, sino que genera los datos dinámicamente cada vez que se consulta.

¿Qué es una Vista?

Una vista es una consulta almacenada que se puede tratar como una tabla. Las vistas se utilizan para simplificar la complejidad de las consultas, proporcionar seguridad y presentar los datos de una manera específica.

Las vistas tienen varios usos, entre ellos:

Simplicidad: Simplificar consultas complejas al encapsularlas en una vista que se puede consultar fácilmente.

Seguridad: Restringir el acceso a datos sensibles proporcionando solo la información necesaria a través de la vista.

Mantenimiento: Facilitar el mantenimiento de consultas complejas al centralizarlas en una vista.

Consistencia: Asegurar que las consultas reutilizadas sean consistentes en toda la aplicación.

Características de las Vistas

1. **Es una Tabla Virtual**: Una vista no almacena datos físicamente, sino que muestra los datos de las tablas subyacentes.
2. **Actualización Dinámica**: Los datos en una vista siempre están actualizados, ya que se generan en tiempo real a partir de las tablas base.
3. **Seguridad**: Las vistas pueden restringir el acceso a datos sensibles al mostrar solo las columnas y filas necesarias.
4. **Simplificación**: Facilitan la escritura de consultas complejas al encapsular la lógica en una sola vista.
5. **Compatibilidad**: Proporciona una interfaz consistente incluso si cambia el esquema de las tablas subyacentes.
6. **Rendimiento**: Aunque MySQL no soporta vistas indexadas, las vistas pueden mejorar la organización y legibilidad de las consultas. También es importante tener en cuenta alternativas como las tablas temporales y la optimización de consultas, en casos de vistas muy complejas.

Tipos de Vistas

1. **Vistas Simples**: Basadas en una sola tabla. Pueden ser "actualizables", es decir usar update, delete e insert a través de ellas, pero solo si están basadas en 1 sola tabla y no usan: join/unión/disctint/subquerys/funciones de agregado.

2. **Vistas Complejas**: Basadas en múltiples tablas y pueden incluir funciones agregadas. No son "actualizables" (complejidad para asegurar consistencia e integridad).

3. **Vistas Materializadas**: MySQL no soporta vistas materializadas/persistentes nativamente, pero se pueden simular mediante tablas temporales y triggers.

Limitaciones de las Vistas

1. *No Almacenan Datos:* Las vistas no almacenan datos físicamente, lo que puede afectar el rendimiento en consultas muy complejas.
2. *Actualización:* No todas las vistas son "actualizables", especialmente las que incluyen funciones agregadas o combinaciones complejas.
3. *Rendimiento:* Consultas a vistas complejas pueden tener un rendimiento inferior en comparación con consultas directas a las tablas subyacentes, ya que la vista necesita ser recalculada cada vez que se accede. Es en estos (y otros casos), el **optimizador de consultas** puede que tenga alguna dificultad al momento de realizar su trabajo debido a que existen factores subyacentes a la vista, que afectan el rendimiento. Por ejemplo: complejidad de la consulta, índices, funciones agregadas, group by y tamaño de las tablas involucradas.
4. *Restricciones:* Algunas vistas no pueden contener ciertas cláusulas, como ORDER BY, a menos que se utilicen con LIMIT para garantizar el orden de los resultados.
5. *Seguridad y Permisos:* Las vistas pueden restringir el acceso a ciertas columnas de las tablas subyacentes, pero no pueden ocultar completamente los datos subyacentes si un usuario tiene permisos suficientes para acceder a las tablas directamente.
6. *Índices:* Las vistas no pueden tener índices propios (ya que no almacenan datos físicamente), lo que puede limitar el rendimiento en consultas que se beneficiarían del uso de índices.
7. *Dependencia de las Tablas Subyacentes:* Los cambios en las tablas subyacentes, como la modificación de la estructura o eliminación de columnas, pueden invalidar las vistas que dependen de esas tablas.
8. *Limitaciones en Consultas de Unión:* Vistas que usan UNION o UNION ALL pueden tener limitaciones en su capacidad para actualizar datos a través suyo.
9. *Limitaciones para depurar consultas c/vistas complejas:* Esto se debe a que la lógica de las vistas esta oculta en su definición y por ende esto dificulta el seguimiento de una consulta que utilice una vista.

Crear Vistas en MySQL

1- create view comprasXcliente as
select
c.CLI_ID,
c.APELLIDO,
c.NOMBRES,
v.FECHA_COMPRA
from
clientes c,
ventas v
where
c.CLI_ID = v.CLI_ID

2- CREATE
[OR REPLACE]
VIEW nombre_vista [(columna1, columna2, ...)] //por defecto, los mismos nombres que las columnas de las tablas
AS
sentencia_select
[WITH [CASCADED | LOCAL] CHECK OPTION];

```
CREATE OR REPLACE VIEW clientes_activos (id_cliente, nombre_completo, email) AS
SELECT cli_id, CONCAT(apellido, ', ', nombres), correo_electronico
FROM clientes
WHERE estado = 'activo';
```

With CHECK OPTION: se utiliza al crear o modificar una vista para controlar cómo se manejan las inserciones y actualizaciones de datos a través de esa vista. Su objetivo principal es asegurar que cualquier cambio realizado a través de la vista cumpla con las condiciones definidas en la consulta SELECT que define la vista.

LOCAL:
- Cuando se utiliza LOCAL, las comprobaciones de la cláusula WHERE (o cualquier otra condición de filtrado) se aplican solo a la vista específica en la que se define CHECK OPTION.
- Si la vista se basa en otra vista (una vista anidada), las comprobaciones de otras vistas anidadas no se tienen en cuenta.

CASCADED:
- Cuando se utiliza CASCADED, las comprobaciones se aplican no solo a la vista actual, sino también a todas las vistas subyacentes (vistas anidadas) que tengan CHECK OPTION definido.
- Esto asegura que cualquier cambio realizado a través de la vista cumpla con las condiciones de todas las vistas en la cadena de vistas anidadas.

Consultar la vista (como cualquier tabla)

```
select
        *
from
        comprasxcliente
```

Ejemplos

Vista “Simple”

1-
```
CREATE VIEW ResumenDeVentas AS
SELECT
    IDOrdenDeVenta AS SalesOrderID,
    FechaDeOrden AS OrderDate,
    TotalDebido AS TotalDue
FROM
    EncabezadoDeOrdenDeVenta;
```

2-
```
CREATE TABLE empleados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    departamento VARCHAR(50),
    salario DECIMAL(10, 2)
);

CREATE VIEW vista_empleados AS
SELECT nombre, departamento
FROM empleados;

-- Actualizar el departamento de un empleado
UPDATE vista_empleados
SET departamento = 'Recursos Humanos'
WHERE nombre = 'Juan';
```

Vista “Compleja”

1-
```
CREATE VIEW VentasDeProductos AS
SELECT
    p.Nombre,
    SUM(s.CantidadOrdenada) AS TotalVendido
FROM
    Producto AS p
JOIN
    DetalleDeOrdenDeVenta AS s ON p.IDProducto = s.IDProducto
GROUP BY
    p.Nombre;
```

2-
```
CREATE TABLE productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    precio DECIMAL(10, 2)
);

CREATE TABLE ventas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    producto_id INT,
    cantidad INT,
    FOREIGN KEY (producto_id) REFERENCES productos(id)
);

CREATE VIEW vista_productos_ventas AS
SELECT p.nombre, p.precio, SUM(v.cantidad) AS total_ventas
FROM productos p
JOIN ventas v ON p.id = v.producto_id
GROUP BY p.nombre, p.precio;

-- Intentar actualizar a través de la vista generaria un error
UPDATE vista_productos_ventas
SET precio = 25.00
WHERE nombre = 'Producto A';
```

Modificar una vista

```
ALTER VIEW vista_empleados AS
SELECT nombre, departamento, salario
FROM empleados;
```

Eliminar una vista

```
DROP VIEW vista_empleados;
```

Índices en MySQL

Introducción

En una base de datos, los índices son estructuras que mejoran la velocidad de recuperación de datos en las consultas. Cuando trabajamos con grandes volúmenes de datos, la búsqueda de información puede ser muy lenta si no tenemos índices definidos. Un índice actúa como el índice de un libro: nos permite encontrar la información rápidamente sin tener que recorrer todo el contenido.

Concepto Básico
Un índice en MySQL es un objeto de base de datos que puede ser creado sobre una o más columnas de una tabla. El objetivo es permitir un acceso más rápido a los datos en las consultas que usan esas columnas.

¿Por qué usar Índices?
Imaginemos que tenemos una tabla de clientes con miles o millones de registros. Si deseamos encontrar rápidamente a un cliente por su apellido, MySQL tendríamos que revisar fila por fila para encontrar el valor deseado si no existe un índice. Este proceso es ineficiente. Sin embargo, al crear un índice en la columna "Apellido", MySQL puede localizar el dato de manera mucho más rápida, evitando leer todas las filas de la tabla.

Ventajas principales:

1. **Mejora en la velocidad de consulta:** Un índice bien diseñado puede hacer que las consultas sobre una gran cantidad de datos sean mucho más rápidas.
2. **Ordenamiento más eficiente:** Los índices facilitan el ordenamiento rápido de resultados.
3. **Optimización de búsquedas:** Las búsquedas que usan las columnas indexadas son más eficientes.

Desventajas o Consideraciones:

1. **Uso de espacio:** Los índices ocupan espacio adicional en la base de datos.
2. **Impacto en inserciones, actualizaciones y eliminaciones:** Cada vez que los datos de una tabla cambian, el índice también debe actualizarse, lo que puede ralentizar las operaciones de modificación de datos.

Tipos de Índices en MySQL

Índice Clustered (Agrupado)

Un índice clustered o índice agrupado determina el orden físico de los datos en la tabla. Solo puede haber un índice agrupado por tabla, ya que solo puede haber un orden físico en los datos.

- La tabla misma se organiza de acuerdo con el índice agrupado.
- Cuando realizas una consulta que utiliza este índice, MySQL puede encontrar los datos mucho más rápido, ya que están ordenados físicamente.

Ejemplo de Creación de un Índice Agrupado:

```
CREATE CLUSTERED INDEX IX_Clientes_Apellido ON Clientes (Apellido);
```

Índice Non-Clustered (No Agrupado)

Un índice non-clustered o índice no agrupado no afecta el orden físico de los datos en la tabla, sino que crea una estructura separada que apunta a las filas correspondientes en la tabla base.

- Podemos tener varios índices no agrupados en una tabla.
- Este índice mantiene una referencia a la ubicación física de los datos.

Ejemplo de Creación de un Índice No Agrupado:

```
CREATE INDEX IX_Clientes_Email ON Clientes (Email);
```

Índices Compuestos

Un índice compuesto es un índice que se crea sobre más de una columna de una tabla. Es útil cuando las consultas filtran u ordenan datos en más de una columna.

Ejemplo de Creación de un Índice Compuesto:

```
CREATE INDEX IX_Clientes_Apellido_Nombre ON Clientes (Apellido, Nombre);
```

Cómo Utilizan los Índices las Consultas

Consulta sin Índices: Imaginemos una tabla de clientes sin ningún índice. Si realizamos una consulta para buscar clientes con un apellido específico, MySQL debe revisar todas las filas de la tabla para encontrar los resultados:

```
SELECT * FROM Clientes WHERE Apellido = 'González';
```

Consulta con Índices: Al agregar un índice en la columna Apellido, MySQL puede usar el índice para buscar rápidamente la fila o filas que contienen "González":

```
SELECT * FROM Clientes WHERE Apellido = 'González';
```

Mantenimiento de Índices

Debido a que los índices afectan el rendimiento de las consultas, pero también impactan las operaciones de modificación de datos, es importante realizar mantenimiento regular de los índices. MySQL ofrece herramientas como OPTIMIZE TABLE para optimizar los índices.

Reorganizar y Reconstruir Índices:

```
OPTIMIZE TABLE Clientes;
```

Buenas Prácticas

- No crear índices en todas las columnas. Esto aumenta el uso de espacio y puede afectar el rendimiento de las operaciones de inserción, actualización y eliminación.
- Evaluar la frecuencia de uso de columnas en las consultas antes de crear un índice.
- Realizar un mantenimiento regular de los índices, especialmente en tablas que reciben muchas inserciones y actualizaciones.
- Utilizar índices compuestos en columnas que frecuentemente se consultan juntas.

Modificar un Índice

Para modificar un índice, generalmente tendríamos que eliminar el índice existente y luego crear uno nuevo con las modificaciones deseadas. MySQL no permite modificar directamente un índice existente. Se haría asi:

1- DROP INDEX nombre_del_indice ON nombre_de_la_tabla;

Ejemplo: DROP INDEX IX_Clientes_Apellido ON Clientes;

2- CREATE INDEX nuevo_nombre_del_indice ON nombre_de_la_tabla (nueva_columna1, nueva_columna2);

Ejemplo: CREATE INDEX IX_Clientes_Nombre_Apellido ON Clientes (Nombre, Apellido);

Eliminar un Índice

Para eliminar un índice en MySQL, podemos usar el comando DROP INDEX. Aquí la sintaxis y un ejemplo:

DROP INDEX nombre_del_indice ON nombre_de_la_tabla;

Ejemplo: DROP INDEX IX_Clientes_Email ON Clientes;