



Módulo 5: Vistas - Apunte

Vistas (Views)

Una **vista** (o *view*, en inglés) es una **consulta almacenada** que se comporta como una **tabla virtual**. Esto significa que, aunque no almacena datos físicamente, permite acceder a información como si se tratara de una tabla. Internamente, una vista es una instrucción **SELECT** predefinida que se ejecuta cada vez que se consulta la vista.

Desde el punto de vista del usuario, una vista actúa como una tabla con filas y columnas. Sin embargo, los datos no están duplicados: se obtienen dinámicamente a partir de las tablas reales cada vez que se accede a la vista.

En MySQL, las vistas se crean con la sentencia **CREATE VIEW** y pueden usarse en consultas del mismo modo que una tabla.

Las vistas se utilizan por diversas razones, tanto funcionales como de diseño. Algunas de las principales ventajas de su uso son:

- **Simplificación de consultas complejas:** Una vista puede encapsular una consulta con múltiples *joins*, filtros y cálculos, facilitando su reutilización sin repetir código.
- **Seguridad y control de acceso:** Es posible otorgar permisos a ciertos usuarios para consultar una vista, sin darles acceso directo a las tablas subyacentes. De este modo, se pueden ocultar columnas sensibles o restringir filas visibles.
- **Separación entre lógica y presentación:** Las vistas permiten abstraer la estructura de los datos, facilitando que los usuarios trabajen con una representación más intuitiva sin conocer el modelo físico.
- **Reutilización y mantenimiento:** Al centralizar la lógica de consulta en una vista, se facilita su mantenimiento. Si la lógica cambia, solo es necesario actualizar la vista, sin modificar todas las consultas que la utilizan.

Diferencias entre una vista y una tabla

Característica	Tabla	Vista
Almacenamiento	Almacena físicamente los datos	No almacena datos; consulta virtual
Actualización directa	Sí, mediante INSERT , UPDATE , DELETE	Solo si es una vista actualizable
Velocidad de acceso	Más rápida al consultar grandes volúmenes de datos	Puede ser más lenta si es compleja



Persistencia de datos	Sí, los datos permanecen hasta ser modificados	Los datos se generan al consultar
Independencia lógica	Baja	Alta, se puede modificar sin afectar tablas

En MySQL, las vistas no pueden tener índices propios ni claves primarias, ya que no almacenan datos.

Se recomienda usar vistas en las siguientes situaciones:

- Cuando se necesita **reutilizar consultas** complejas o frecuentes en varios lugares del sistema.
- Para **restringir el acceso** a ciertas columnas o filas sin modificar la tabla original.
- En aplicaciones donde se busca una **interfaz de datos más clara** o más adaptada a un tipo específico de usuario.
- En procesos de **análisis de datos**, para presentar reportes agregados o filtrados.
- Para **ocultar la complejidad del modelo físico**, ofreciendo una capa intermedia más intuitiva para usuarios finales o desarrolladores.

Las vistas permiten trabajar con los datos a un nivel lógico superior, sin necesidad de conocer detalles del almacenamiento físico o del modelo completo de relaciones.

Sintaxis para crear vistas en MySQL

En MySQL, una vista se define utilizando la instrucción **CREATE VIEW**, seguida del nombre de la vista y una consulta **SELECT** que especifica el contenido de la misma. La estructura general es:

```
CREATE VIEW nombre_vista AS
  SELECT columnas
    FROM tablas
   WHERE condiciones;
```

Esta consulta se ejecuta cada vez que se consulta la vista, y **no se almacenan físicamente los datos**. La vista se comporta como una tabla virtual basada en los datos actuales de las tablas subyacentes.

MySQL requiere que las vistas estén basadas en consultas válidas. No se pueden crear vistas con parámetros dinámicos ni que incluyan subconsultas que usen variables del usuario.

Reglas de nomenclatura

Al crear una vista, se deben seguir las mismas reglas que para nombrar tablas o columnas:

- El nombre debe comenzar con una letra o guion bajo.

- Puede incluir letras, números y guiones bajos.
- No debe contener espacios ni caracteres especiales.
- No debe coincidir con palabras reservadas del lenguaje SQL.
- En bases de datos sensibles a mayúsculas (como en Linux), se recomienda usar nombres en minúsculas para evitar errores.

Alias de columnas y vistas

En una vista, se pueden (y muchas veces se deben) **renombrar las columnas** usando alias con la cláusula **AS**. Esto es útil para hacer que los nombres sean más claros o evitar ambigüedad cuando se combinan varias tablas.

```
CREATE VIEW vista_nombres_completos AS
SELECT nombre AS nombre_alumno, apellido AS apellido_alumno
FROM alumnos;
```

También se puede asignar alias a las **vistas** cuando se usan en consultas mixtas:

```
SELECT va.nombre_alumno
FROM vista_nombres_completos va;
```

Tipos de Vistas

Las vistas en MySQL pueden clasificarse en función de su complejidad, estructura interna y restricciones sobre su actualización. Conocer estos tipos permite definir correctamente qué tipo de vista utilizar según el problema que se quiere resolver o simplificar.

Vistas Simples

Las **vistas simples** son aquellas que:

- Se basan en una **sola tabla**.
- No utilizan funciones de agregación (**SUM**, **AVG**, **COUNT**, etc.).
- No incluyen **GROUP BY**, **DISTINCT**, **UNION**, **LIMIT**, **HAVING**, ni subconsultas.
- Pueden ser, en muchos casos, **actualizables** (es decir, se pueden usar para modificar datos de la tabla base).

Ventajas:

- Son más fáciles de mantener.
- Suelen permitir **INSERT**, **UPDATE** y **DELETE** directamente desde la vista.

Vistas Complejas

Las **vistas complejas** son aquellas que:



- Involucran más de una tabla (JOINS).
- Usan funciones de agregación o agrupación (**GROUP BY, SUM**, etc.).
- Incluyen subconsultas, expresiones, filtros avanzados o transformaciones de datos.

Limitaciones:

- Por lo general **no son actualizables**, ya que MySQL no puede determinar cómo aplicar cambios a las tablas base de forma segura.
- Están pensadas para **consultas de lectura** complejas.

```
CREATE VIEW vista_estadisticas_departamento AS
SELECT
    c.departamento,
    COUNT(a.id) AS total_asignaturas,
    AVG(a.creditos) AS promedio_creditos
FROM asignaturas a
JOIN carreras c ON a.id_carrera = c.id
GROUP BY c.departamento;
```

Vistas con WITH CHECK OPTION

Cuando una vista es actualizable, podemos usar la cláusula **WITH CHECK OPTION** para **restringir las modificaciones** que violen la condición establecida en la vista.

Esto garantiza que cualquier **INSERT** o **UPDATE** hecho a través de la vista **respete el filtro de la misma**.

```
CREATE VIEW vista_alumnos_sistemas AS
SELECT id, nombre, apellido, edad, id_carrera
FROM alumnos
WHERE id_carrera = 2
WITH CHECK OPTION;
```

- Muestra solo alumnos que pertenecen a la carrera 2.
- Restringe las modificaciones (INSERT o UPDATE) que no cumplan esa condición.
- Garantiza la **consistencia de los datos**: todo cambio hecho a través de esta vista **debe respetar** el **WHERE id_carrera = 2**.

WITH CHECK OPTION se utiliza para **evitar que las operaciones de inserción o actualización** hechas a través de una vista violen las condiciones del **WHERE** que define esa vista.

Sirve para mantener la integridad lógica de la vista, especialmente útil en sistemas donde ciertos perfiles de usuario **solo deben ver o modificar subconjuntos** de la información.

Vistas Anidadas

Una **vista anidada** es una vista que **utiliza otras vistas** como parte de su definición. Esto permite **modularizar** consultas complejas en bloques más pequeños y reutilizables.

Ventajas:

- Mejora la legibilidad del código SQL.
- Favorece la reutilización de lógica de negocio.
- Facilita el mantenimiento del sistema.

```
CREATE VIEW vista_alumnos_mayores_18 AS
SELECT id, nombre, apellido, edad, id_carrera
FROM alumnos
WHERE edad > 18;
```

```
CREATE VIEW vista_mayores_carrera_3 AS
SELECT nombre, apellido
FROM vista_alumnos_mayores_18
WHERE id_carrera = 3;
```

La segunda vista (**vista_mayores_carrera_3**) **usa como fuente a otra vista** (**vista_alumnos_mayores_18**), cumpliendo así con el concepto de **vista anidada** de forma clara y didáctica.

Ventajas y Limitaciones de las Vistas

Las **vistas** son objetos virtuales de una base de datos que permiten encapsular consultas complejas bajo un nombre accesible. Su uso aporta múltiples beneficios en diseño, mantenimiento y seguridad, pero también conlleva ciertas limitaciones que deben tenerse en cuenta.

Ventajas de las Vistas

1. **Ocultamiento de complejidad**
 - Permiten simplificar el acceso a estructuras complejas. Por ejemplo, una vista puede encapsular múltiples **JOIN**, filtros y funciones agregadas, facilitando su uso por parte de otros usuarios o aplicaciones.
2. **Mayor seguridad**
 - Las vistas pueden **restringir el acceso a columnas sensibles** sin necesidad de crear nuevas tablas.
 - Por ejemplo, se puede crear una vista que omita campos como salarios o direcciones y otorgar permisos solo sobre esa vista.
3. **Reutilización de consultas**



- En lugar de repetir constantemente una misma consulta compleja, se puede **crear una vista que la encapsule** y usarla tantas veces como sea necesario.
- Esto reduce errores y facilita el mantenimiento.

4. Simplificación de reportes y análisis

- Las vistas permiten presentar los datos ya filtrados, agrupados o transformados según los requerimientos del área que los consume, como informes gerenciales, dashboards o reportes académicos.

Limitaciones de las Vistas

1. No todas las vistas son actualizables

- En MySQL, **solo algunas vistas permiten operaciones INSERT, UPDATE o DELETE**, y deben cumplir con ciertas condiciones:
 - No tener funciones agregadas (SUM, AVG, etc.).
 - No incluir DISTINCT, GROUP BY, HAVING, UNION, LIMIT, JOIN complejos, ni subconsultas en el SELECT.
 - No derivar de más de una tabla sin clave primaria clara.
- En esos casos, la vista se vuelve de solo lectura.

2. Pérdida de rendimiento en vistas complejas

- Las vistas no almacenan datos físicamente, por lo tanto, **cada vez que se consultan se ejecuta la consulta base**.
- Si la vista es muy compleja o mal optimizada, puede afectar negativamente el rendimiento.

3. No poseen índices propios

- A diferencia de las tablas, **las vistas no tienen índices** porque no almacenan datos.
- El rendimiento depende de los índices definidos en las tablas subyacentes.

4. No almacenan datos físicamente

- Las vistas son **estructuras virtuales**: su contenido se genera dinámicamente al momento de la consulta.
- Esto implica que no se pueden utilizar como almacenamiento intermedio, ni como "caché" permanente de resultados.

Actualización de Datos a través de Vistas

Una vista es **actualizable** cuando permite que las operaciones de modificación de datos como **INSERT, UPDATE y DELETE** se realicen a través de ella, afectando directamente a la tabla (o tablas) subyacente(s).

Sin embargo, **no todas las vistas lo son**. Una vista solo será actualizable si se cumplen ciertas condiciones que aseguren que la modificación puede realizarse sin ambigüedad.

En MySQL, una vista puede considerarse actualizable **siempre que:**

- Se basa en **una sola tabla**.
- No contiene funciones de agregación (SUM, AVG, COUNT, etc.).



- No tiene cláusulas **DISTINCT, GROUP BY, HAVING, UNION.**
- No utiliza subconsultas en la lista de selección.
- No utiliza **LIMIT u ORDER BY** (en versiones anteriores a MySQL 8).
- No renombra columnas con alias complejos que impidan mapear los nombres a la tabla original.
- Todas las columnas requeridas para insertar o actualizar están **incluidas en la vista**.

Si una vista no cumple con estas condiciones, **solo podrá utilizarse para consultar datos**, no para modificarlos.

Cuando se crea una vista que incluye un **WHERE**, puede usarse la cláusula **WITH CHECK OPTION** para evitar que se inserten o modifiquen datos que **violén la condición de la vista**.

Modificación y Eliminación de Vistas

REPLACE: modificar una vista

En MySQL, una vez que una vista ha sido creada, puede **modificarse o eliminarse** según sea necesario. A continuación, se presentan las distintas formas en que se pueden gestionar vistas existentes.

```
CREATE OR REPLACE VIEW nombre_vista AS
SELECT ...
```

Esta sentencia reemplaza una vista existente **por una nueva definición**. Es útil cuando queremos cambiar:

- Las columnas seleccionadas.
- Filtros o condiciones.
- Joins con otras tablas o vistas.

Ventajas:

- No es necesario eliminar la vista previamente.
- Conserva el nombre de la vista.
- Se actualiza automáticamente con la nueva definición.

```
CREATE OR REPLACE VIEW vista_alumnos_basica AS
SELECT id, nombre, apellido, edad
FROM alumnos;
```

DROP VIEW: eliminar una vista

Para eliminar una vista de la base de datos se utiliza:

```
DROP VIEW nombre_vista;
```

También se pueden eliminar varias vistas a la vez:

```
DROP VIEW vista1, vista2, vista3;
```

Una vez eliminada, **ya no puede ser utilizada** ni por consultas ni por otras vistas que la referencian.

Si una vista es eliminada con **DROP VIEW**, cualquier consulta, procedimiento almacenado, trigger u **otra vista** que dependiera de ella dejará de funcionar y arrojará error en tiempo de ejecución.

Recomendaciones

- Usar **CREATE OR REPLACE VIEW** para modificar vistas en lugar de **DROP + CREATE**, ya que evita problemas con permisos y dependencias.
- Documentar las vistas dependientes en sistemas grandes.

Consultas Mixtas con Tablas y Vistas

En bases de datos, llamamos **consultas mixtas** a aquellas que combinan **vistas** con **tablas reales** dentro de una misma instrucción **SELECT, JOIN, WHERE**, etc.

Estas consultas permiten **aprovechar la simplificación y abstracción** que ofrecen las vistas, junto con la flexibilidad de acceder directamente a las tablas reales cuando sea necesario.

Las vistas no almacenan datos: son una forma de **encapsular una consulta compleja** para que otros usuarios puedan acceder a ella fácilmente. Al combinarlas con tablas reales, se logra:

- Reutilizar lógica compleja sin repetirla.
- Unir información resumida con datos detallados.
- Construir reportes dinámicos y legibles.

Supongamos que ya tenemos una vista **vista_alumnos_basica** que muestra **id, nombre, apellido** y **id_carrera** de la tabla **alumnos**.

Ahora combinamos esa vista con la tabla real **carreras** para mostrar el nombre completo del alumno y el nombre de su carrera:

```
SELECT vab.nombre, vab.apellido, c.nombre_carrera
FROM vista_alumnos_basica vab
JOIN carreras c ON vab.id_carrera = c.id;
```

Las consultas mixtas son muy útiles en entornos donde se requiere generar:

- Reportes dinámicos y resumidos.
- Informes filtrados por criterios específicos.
- Dashboards interactivos (por ejemplo, en herramientas como Power BI, Tableau o aplicaciones web).

Ventajas clave:

- **Modularidad:** si se cambia la definición de la vista, se actualizan automáticamente los reportes sin modificar todas las consultas.
- **Seguridad:** se puede restringir el acceso directo a las tablas sensibles, exponiendo solo lo necesario a través de vistas.
- **Claridad:** se separa la lógica de negocio (en la vista) del diseño del reporte (en la consulta mixta).

Vistas almacenadas en MySQL

Una **vista almacenada** (también conocida como **vista materializada**) es un tipo especial de vista que **almacena físicamente los datos** que devuelve una consulta, en lugar de recalcularlos cada vez que se accede. Esto mejora el rendimiento cuando se trata de consultas complejas o datos que no cambian con frecuencia.

Diferencias entre Vista virtual y Vista almacenada

Característica	Vista virtual	Vista almacenada
Almacenamiento	No almacena datos físicos	Sí, almacena los datos en disco
Actualización automática	Siempre actualiza al consultar	Puede requerir actualización manual o programada
Rendimiento en consultas	Depende de la complejidad de la consulta	Más rápida si no cambia frecuentemente
Consumo de espacio	No ocupa espacio extra	Requiere almacenamiento adicional

Permisos y Seguridad sobre Vistas

Las **vistas** no solo sirven para simplificar consultas complejas, sino también para **restringir el acceso a los datos** de forma controlada. Esto convierte a las vistas en una poderosa herramienta de **seguridad** dentro de una base de datos.

Como mencionamos anteriormente, una vista puede mostrar solo **una parte específica** de la información de una tabla, omitiendo columnas sensibles o aplicando filtros. Esto permite que ciertos usuarios accedan a los datos que necesitan **sin ver información confidencial**.

En MySQL, se pueden asignar permisos directamente sobre una vista, **sin necesidad de dar acceso a la tabla base**. Esto permite un control más detallado y seguro.



```
GRANT SELECT ON vista_alumnos_publica TO 'usuario'@'localhost';
```

Este comando permite que el usuario pueda consultar la vista, pero no necesariamente la tabla **alumnos**.

Casos de uso comunes

Privacidad de datos

Evitar mostrar información sensible como edades, direcciones, sueldos, etc.

Auditorías

Crear vistas que solo muestren logs o datos relevantes para control, sin permitir modificar la información.

Control por rol

- Los **administradores** acceden a todas las columnas.
- Los **docentes** ven solo nombre y apellido.
- Un **departamento determinado** accede a créditos o información académica.

Esto se logra **creando diferentes vistas** para cada grupo de usuarios, y asignando permisos según el perfil.

Recomendaciones

- Crear **vistas dedicadas** para cada perfil de acceso.
- No otorgar permisos sobre tablas a usuarios que no lo requieran.
- Si una vista requiere actualización, considerar la opción de usar **WITH CHECK OPTION** para validar los datos permitidos.

Buenas Prácticas en el Uso de Vistas

Las vistas son herramientas muy útiles para simplificar consultas y mejorar la seguridad en las bases de datos, pero para sacarles el máximo provecho es importante seguir ciertas buenas prácticas en su diseño y uso.

Usar nombres descriptivos y consistentes

- El nombre de una vista debe reflejar claramente su contenido y propósito.
- Evitar abreviaturas confusas o nombres genéricos como **vista1** o **datos**.
- Por ejemplo: **vista_alumnos_activos**, **vista_reportes_ventas_anuales**.

Esto facilita la comprensión y mantenimiento del esquema de la base de datos.

Documentar el propósito de cada vista

- Es recomendable agregar comentarios que expliquen qué información ofrece la vista y para qué se utiliza.
- Esto ayuda a futuros desarrolladores y administradores a entender su función sin necesidad de analizar el código SQL.

Evitar anidamientos innecesarios

- Las vistas que dependen de otras vistas (vistas anidadas) pueden generar consultas muy complejas y dificultar la optimización.
- Siempre que sea posible, crear vistas que se basen directamente en tablas o que tengan niveles mínimos de anidamiento.
- Esto mejora el rendimiento y facilita el control.

Optimizar las consultas usadas dentro de vistas

- Las vistas ejecutan la consulta definida cada vez que se usan, por lo que es fundamental que su definición sea eficiente.
- Evitar usar subconsultas innecesarias o funciones costosas dentro de la vista.
- Indexar adecuadamente las tablas base para acelerar los joins y filtros que usa la vista.

Separar vistas operativas y analíticas

- **Vistas operativas:** se usan dentro del sistema para procesos diarios, con datos actuales y pocas agregaciones.
- **Vistas analíticas:** diseñadas para reportes y análisis, que pueden incluir agrupaciones, cálculos y datos históricos.
- Mantener esta separación ayuda a evitar que consultas analíticas pesadas afecten el rendimiento del sistema operativo.