TEMA 8: REALIZACIÓN DE VISTAS Y OTROS OBJETOS DE LA BD.

Vistas e índices (Lenguaje DDL)

OBJETIVOS:

- Utilizar asistentes y herramientas gráficas
- Utilizar el lenguaje de definición de datos
- Identificar las herramientas y sentencias para realizar consultas
- Crear vistas y definir índices.

Vistas en SQL

En el caso del SQL, el sistema cuenta con dos mecanismos diferentes implicados en el mantenimiento de la seguridad: el sistema de gestión de vistas, y el subsistema de autorización que nos permitan asignar a los usuarios permisos de acceso a tablas y vistas de la base de datos. Veremos por encima el caso de las vistas.

En SQL, una vista es una tabla virtual en la base de datos cuyos contenidos están definidos por una consulta. Para el usuario de la base de datos, la vista aparece igual que una tabla real, con un conjunto de columnas designadas y filas de datos pero a diferencia de una tabla real, una vista no existe en la base de datos como conjunto almacenado de datos, por lo que no debemos preocuparnos acerca del espacio de disco ocupado por éstas. En su lugar, las filas y columnas de datos visibles a través de la vista son los resultados producidos por la consulta que define la vista.

Las vistas proporcionan una variedad de beneficios, pudiendo destacar los siguientes:

- Seguridad. Cada usuario puede obtener permiso para acceder a la base de datos únicamente a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver. Las vistas permiten que diferentes usuarios vean a la BD desde diferentes perspectivas, así como restringir el acceso a los datos de modo que diferentes usuarios accedan sólo a ciertas filas o columnas de una tabla.
- Simplicidad de consulta. Una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que consultas multitabla se formulen como consultas de una sola tabla con respecto a la vista.
- Simplicidad estructurada. Las vistas pueden dar a un usuario una visión "personalizada" de la estructura que tiene la base de datos presentando está como un conjunto de tablas virtuales que tienen sentido para ese usuario.
- Aislamiento frente al cambio. Una vista representa una imagen consistente inalterada de la base de datos, incluso si las tablas originales se dividen, reestructuran o cambian de nombre.
- Integridad de datos. Si los datos se acceden y se introducen a través de una vista, el gestor de la base de datos puede comprobar automáticarnente los datos para asegurarse que satisfacen restricciones de integridad específicas.

Sin embargo, las vistas presentan también una serie de desventajas al utilizarlas en lugar de una tabla real. Estas desventajas son:

- Rendimiento. Las vistas crean la apariencia de una tabla, pero el gestor de la base de datos debe traducir las consultas con respecto a la vista en consultas con respecto a las tablas fuentes originales.
- Restricciones de actualización. Cuando un usuario trata de actualizar filas de una vista, el gestor de la base de datos debe traducir la petición a una actualización sobre las filas de las tablas fuente. Esto es posible para vistas sencillas, pero vistas complejas no pueden ser actualizadas, son "de solo lectura".

CREACIÓN DE VISTAS.

```
CREATE [OR REPLACE]
    VIEW nombre_vista [(columnas)]
    AS sentencia_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Esta sentencia crea una vista nueva o reemplaza una existente si se incluye la cláusula OR REPLACE. La sentencia_select es una sentencia SELECT que proporciona la definición de la vista. Puede estar dirigida a tablas de la base o a otras vistas.

Se requiere que posea el permiso CREATE VIEW para la vista, y algún privilegio en cada columna seleccionada por la sentencia SELECT. Para columnas incluidas en otra parte de la sentencia SELECT debe poseer el privilegio SELECT. Si está presente la cláusula OR REPLACE, también deberá tenerse el privilegio DELETE para la vista.

Toda vista pertenece a una base de datos. Por defecto, las vistas se crean en la base de datos actual. Pera crear una vista en una base de datos específica, indíquela con base_de_datos.nombre_vista al momento de crearla.

Opcionalmente se puede asignar un nombre a cada columna de la vista. Si se especifica, la lista de nombres de las columnas debe de tener el mismo número de elementos que el número de columnas producidas por la consulta. Si se omiten, cada columna de la vista adopta el nombre de la columna correspondiente en la consulta. Existen dos casos en los que es obligatoria la especificación de la lista de columnas:

- Cuando la consulta incluye columnas calculadas

- Cuando la consulta produce nombres idénticos.

Según el propósito con el que se organizan las vistas, pueden clasificarse en los siguientes tipos:

a) Horizontales o por Restricción

Son las que restringen el acceso de un usuario a únicamente un conjunto de filas de una tabla.

EJEMPLO: Si se desea que un director de ventas vea sólo las filas de la tabla RVENTAS, correspondientes a los vendedores de su región, se debería de definir una vista para cada región.

```
CREATE VIEW repEste AS
  SELECT *
  FROM  RVENTAS
  WHERE r_oficina IN (SELECT n_oficina
                      FROM OFICINAS
                      WHERE region = 'Este');
```

Las vistas horizontales son adecuadas cuando la tabla sobre la que se definen contiene datos que relacionan a varias organizaciones o usuarios, proporcionando una "tabla privada" para cada usuario compuesta únicamente de las filas necesarias para ese usuario en concreto.

OTRO EJEMPLO:

Definir una vista para el empleado nº 102 que contenga solo los pedidos emitidos, por los clientes asignados al mismo.

```
CREATE VIEW emplCientodos AS
SELECT *
FROM PEDIDOS
WHERE clie IN (SELECT num_clie
               FROM CLIENTES
               WHERE rep_clie = 102);
```

Definir una vista, que muestre únicamente clientes que tienen más de 30000 € en pedidos registrados actualmente.

```
CREATE VIEW clienTresmil AS
SELECT *
FROM CLIENTES
WHERE 30000 < (SELECT SUM (importe)
                    FROM PEDIDOS
                WHERE num_clie = clie);
```

Crear una vista para el Banco Atlántico, que le permita ver la lista de usuarios que pagan en él.

```
CREATE VIEW vistaUsuariosAtlantico AS
SELECT *
FROM USUARIOS
WHERE codigo_banco IN (SELECT ent_suc
                            FROM BANCOS
    WHERE nombre = 'ATLANTICO');
```

b) Verticales o por proyección

Son aquellas que restringen el acceso de un usuario sólo a ciertas columnas de una tabla.

EJEMPLO:

El departamento de procesamiento de pedidos necesita acceso únicamente al número de empleado, nombre y oficina asignada al mismo para procesar los pedidos. Crear una vista que muestre esta información.

```
CREATE VIEW infoPedidos AS
SELECT num_empl, r_oficina
FROM RVENTAS;
```

El uso de estas vistas es aconsejable cuando los datos de la tabla son compartidos también por varios usuarios diferentes. La diferencia está en que ahora solo necesitamos parte de los campos, no todos.

OTRO EJEMPLO:

Definir una vista de la tabla oficinas, para el personal del procesamiento de pedidos que incluya la ciudad, el número de oficina y la región.

```
CREATE VIEW personalPedidos AS

SELECT ciudad, n_oficina,  region

FROM OFICINAS;
```

En la tabla clientes, para el mismo departamento, que incluya el nombre de los clientes y la asignación de los vendedores

```
CREATE VIEW vistaClien AS

SELECT rep_clie

FROM CLIENTES;
```

Vista sobre la tabla de usuarios para el Dpto. de Contabilidad, en el que se incluyan solamente el DNI, el código de banco y paga banco.

```
CREATE VIEW vistaUser AS

SELECT DNI, codigo_banco, paga_banco

FROM USUARIOS;
```

c) Subconjuntos fila / columna

Lo más habitual es crear vistas con parte de las filas y parte de las columnas de una tabla. Por ejemplo:

Definir una vista que contenga el número de cliente, el nombre de la empresa y el límite de crédito de todos los clientes asignados al vendedor Bill Adams.

```
CREATE VIEW clieAdams AS

SELECT empresa, lim_credito

FROM CLIENTES

WHERE rep_clie = (SELECT num_empl

                  FROM RVENTAS
```

WHERE nombre = “Bill Adams”),

OTRO EJEMPLO:

Crear una vista con los datos personales de los usuarios de Luanco.

CREATE VIEW clieLuanco AS

SELECT DNI, nombre, apellidos, domicilio, localidad, CP

FROM USUARIOS

WHERE localidad = ‘LUANCO’;

d) Agrupados

Son vistas que incluyen una cláusula GROUP BY en la consulta especificada. Estas vistas incluyen siempre una lista de nombres de columna.

EJEMPLO:

Definir una vista que contenga los datos sumarios de los pedidos para cada vendedor.

CREATE VIEW datSumarios (N_pedidos, Total, Promedio, Máximo, Mínimo, N_vendedor) AS

SELECT COUNT(*), SUM(importe), AVG(importe), MAX(importe), MIN(importe), rep

FROM PEDIDOS

GROUP BY rep_clie;

OTRO EJEMPLO:

1- Definir una vista que muestre el total de ventas y el promedio de edad de cada oficina.

CREATE VIEW totalVentas (Oficina, Total, Promedio_edad) AS

SELECT r_oficina, SUM(ventas), AVG(edad)

FROM RVENTAS

GROUP BY r_oficina;

2- Crear una vista que muestre el número de usuarios que pagan en cada banco, junto con el promedio de pago (el máximo y el mínimo).

CREATE VIEW usuariosPagan (N_usuarios, Promedio, Máximo, Mínimo,Banco) AS

SELECT COUNT(*), AVG(cuota_socio), MAX(cuota_socio), MIN(cuota_socio), codigo_banco

FROM USUARIOS

GROUP BY codigo_banco;

e) Compuestas

Son vistas creadas a partir de los datos de dos o más tablas. Presentan los datos como una única tabla virtual. Al igual que las anteriores se utilizan frecuentemente para facilitar el manejo de consultas complejas de uso frecuente en la base.

EJEMPLO:

Crear una vista de la tabla PEDIDOS con los nombres de los vendedores y clientes, en lugar de sus respectivos números.

```
CREATE VIEW infoPedidos  AS

SELECTnum_pedido,empresa,nombre,importe
FROM PEDIDOS, RVENTAS, CLIENTES

WHERE rep = num_empl

AND clie = num_clie;
```

OTRO EJEMPLO

1-Listado del nombre de los clientes y la descripción e importe total de cada producto de los productos pedidos por el mismo.

```
CREATE VIEW infoClien (nombre_clie, producto,Total )AS

SELECT empresa, descripción, SUM(importe)

FROM CLIENTES, PEDIDOS, PRODUCTOS

WHERE num_clie = clie

AND id_producto = producto

AND id_fab = fab

GROUP BY empresa, id_producto, descripcion;
```

2-DNI de los usuarios y descripción y cuota de la actividad a la que estén apuntados.

```
CREATE VIEW usuApuntaos AS

SELECT DNI, descripcion, cuota
```

```
FROM USUARIOS, ACTIVIDADES A, ACTIVIDADES_USUARIOS AU

WHERE num_socio = codigo_usuario

AND A.codigo_actividad = AU.codigo_actividad;
```

3-Crear una vista q permita obtener un listado del nombre y apellidos de los clientes junto con el nombre del banco en el que pagan.

```
CREATE VIEW infoClientBan (nombr_clie, apell_clie, nombr_banco) AS

SELECT USUARIOS.nombre, apellidos, BANCOS.nombre

FROM USUARIOS, BANCOS

WHERE codigo_banco = ent_suc;
```

MOSTRAR LA LISTA DE VISTAS

Para consultar las vistas creadas se dispone de la vista VIEWS, (de la base de datos INFORMATION_SCHEMA) que permite mostrar una lista de todas las vistas que posee el usuario actual. La columna VIEW_DEFINITION contiene la sentencia que se utilizó para crear la vista (sentencia que es ejecutada cada vez que se invoca a la vista).

Así para visualizar los nombres de las vistas con sus textos, tenemos:

```
SQL> SELECT TABLE_NAME, VIEW_DEFINITION FROM VIEWS;

VIEW_NAME

-----------------------------

TEXT

--------------------------------------------------------------------------------

SPAIN

SELECT "CLT_NUM","CLT_APELL","CLT_NOM","CLT_PAIS","CLT_POB" FROM CLIENTES WHERE
```

Visualizar las vistas que hay en una base de datos y visualizar si son actualizables.

```
mysql> SELECT TABLE_NAME, IS_UPDATABLE FROM VIEWS WHERE TABLE_SCHEMA="EXAMEN3";

+------------+--------------+

| TABLE_NAME | IS_UPDATABLE |

+------------+--------------+
```

| vista1 | YES |

| vista2 | YES |

| vista3 | YES |

+----------+----------+

ACCESO A VISTAS

El usuario accede a los datos de una vista exactamente igual que si estuviera accediendo a una tabla. De hecho, en general, el usuario no sabrá que está consultando una vista.

EJEMPLOS:

1) Mostrar el nombre, el número de los pedidos, el importe total y el pedido promedio, para cada vendedor, mostrando el nombre del mismo. Ordenar los resultados de mayor a menor importe total.

[CREATE VIEW datSumarios (N_pedidos, Total, Promedio, Máximo, Mínimo, N_vendedor) AS

SELECT COUNT(*), SUM(importe), AVG(importe), MAX(importe), MIN(importe), rep

FROM PEDIDOS

GROUP BY rep_clie;] Esta es la vista sobre la que trabajo

SELECT RV.nombre, dS.N_pedidos, dS.Total, dS.Promedio, dS.N_vendedor

FROM datSumarios dS, RVENTAS RV

WHERE dS.N_vendedor = RV.num_empl

ORDER BY dS.Total DESC;

2) Mostrar los pedidos actuales totales por cada vendedor y empresa.

[CREATE VIEW infoPedidos AS

SELECTnum_pedido,empresa,nombre,importe
FROM PEDIDOS, RVENTAS, CLIENTES

WHERE rep = num_empl

AND clie = num_clie;] Esta es otra vista sobre la que trabajo

SELECT empresa, nombre, SUM(importe) 'Total'

FROM infoPedidos

GROUP BY empresa, nombre;

3) Mostrar los pedidos actuales mayores de 2000€ ordenados por importe, junto con el nombre del cliente y el vendedor.

[Trabajo con la misma vista anterior]

SELECT num_pedido, importe, empresa, vendedor

FROM infoPedidos

WHERE importe > 2000

ORDER BY importe DESC;

4 – ACTUALIZACIÓN A TRAVÉS DE LAS VISTAS:

Para que una vista se pueda actualizar debe existir una relación directa entre las filas y columnas de la vista y las de la tabla fuente. Según el estándar ANSI, se puede actualizar a través de las vistas si la consulta que la define, satisface las siguientes restricciones:

1ª - No especifica DISTINCT, ni UNION

2ª - FROM debe especificar una única tabla

3ª - La lista de selección no puede contener expresiones, columnas calculadas o funciones de columna(como max(), min(), etc), solo referencias a columnas simples.

4ª - WHERE no debe incluir subconsultas.

5ª - No debe incluir ni GROUP BY ni HAVING en la vista.

6ª Ausencia de subconsultas en las clausulas select y from

EJEMPLO:

1) Crear una vista que muestre los vendedores de la región este.

CREATE VIEW repEste AS

SELECT *

FROM RVENTAS

WHERE r_oficina IN (11,12,13); //que serían las situadas al Este

Ahora añadimos un nuevo vendedor a través de la vista.

INSERT INTO repEste (num_empl, nombre, r_oficina, edad, ventas)

VALUES (113, 'Jorge Ruiz', 11, 43, 0.00);

Ahora resulta que trasladan a otro empleado(104) a otra oficina (12).

UPDATE repEste

SET r_oficina = 21 //aquí se equivocan y en lugar de poner 12 ponen 21

WHERE num_empl = 104;

Con lo cual este señor no figuraría en la vista, pero sí en la tabla.

5 – Comprobación de actualizaciones de vistas

Para evitar errores se usará la cláusula WITH CHECK OPTION. Añadiendo esta cláusula a la creación de una vista se asegura que las operaciones de inserción y actualización sobre la vista satisfagan el criterio de búsqueda, definido en la cláusula WHERE de la consulta asociada a la vista. De esta forma se asegura que las filas insertadas o actualizadas sean visibles a través de la vista. En caso de que la fila a insertar o modificar no satisfaga la condición del WHERE, la sentencia de actualización falla sin llevarse a cabo la operación.

EJEMPLO:

(A partir de la vista creada anteriormente de los usuarios de LUANCO)

[CREATE VIEW clieLuanco AS

SELECT DNI, nombre, apellidos, domicilio, localidad, CP

FROM USUARIOS

WHERE localidad = 'LUANCO'

//Ahora se le pondría debajo el

WITH CHECK OPTION;]

Cambiar el domicilio de Jose Ruiz Peña de Luanco, a AVILES.

UPDATE clieLuanco

SET localidad = 'AVILES'

WHERE nombre = 'JOSE'

AND apellidos = 'RUIZ PEÑA';

//Tras actualizarlo de este modo, me da el siguiente mensaje de error: *"ERROR en línea 1402: violación de la cláusula-WHERE en la vista WITH CHECK OPTION".*

6 – Otras cláusulas para creación de vistas.

A) *OR REPLACE:*

Cuando se necesita modificar la definición de una vista, la única solución es crear una nueva vista con el mismo nombre ya que no existe una sentencia similar a la ALTER TABLE. Para evitar que tener que eliminar la vista y volverla a crear, el motor proporciona esta cláusula, la cual permite sustituir la definición de una vista por otra nueva, es decir, la reemplaza.

CREATE OR REPLACE VIEW repEste AS

SELECT *

FROM RVENTAS

WHERE r_oficina IN (11,12,13)

WITH CHECK OPTION;

B) *WITH READ ONLY:*

Esta cláusula no permite que se ejecuten borrados, actualizaciones o inserciones sobre la vista.

```
CREATE OR REPLACE VIEW ofEste AS //el OR REPLACE es opcional
SELECT *
FROM OFICINAS
WHERE region = 'Este'
WITH READ ONLY;
```

7 – Eliminación de vistas

Se hace con DROP VIEW id_vista;

8 – Ventajas e inconvenientes de las vistas

VENTAJAS:

- Las consultas con selecciones complejas se simplifican.
- Permiten personalizar la BD para los distintos usuarios, de forma que presenten los datos con una estructura lógica para los mismos.
- Control de acceso a la BD, haciendo que los usuarios vean y manejen solo determinada información.

INCONVENIENTES:

- Las restricciones referidas a las actualizaciones.
- La caída del rendimiento cuando se construyen vistas con selecciones complejas.

EJERCICIOS:

1) Crear una vista con el nombre de los usuarios y la actividad e las que está dado de alta junto con la fecha en la que se dio de alta.

```
CREATE VIEW fichaSocio AS
SELECT nombre, apellidos, AU.fecha_alta, descripcion
FROM USUARIOS U, ACTIVIDADES_USUARIOS AU, ACTIVIDADES A
WHERE num_socio = codigo_usuario
AND AU.codigo_actividad = A.codigo_actividad;
```

2) Crear una vista de pagos en las que aparezca el nombre del usuario en lugar de su código.

```
CREATE VIEW pagosSocio (Nom_usu, N_mes, Cuota, Observac) AS
SELECT nombre, numero_mes, cuota, observaciones
FROM USUARIOS, PAGOS
WHERE codigo_usuario = num_socio;
```

3) Crear una vista que muestre la información de las actividades de Artes marciales. Crear o insertar a través de esta vista una nueva actividad que es la siguiente: AM00003, TAEKWONDO, 12.5. Mostrar un listado de la vista creada y después hacer una modificación, cambiando el código a MA00003. Volvemos a mostrar el listado. A través de la tabla, volver a dejar las cosas como estaban y modificar la lista para que la vista compruebe que las actualizaciones sean correctas. Una vez hecho esto, volvemos a hacer la modificación para comprobar que no nos deja.

```
CREATE VIEW infoArtesM AS
SELECT *
FROM ACTIVIDADES
WHERE codigo_actividad LIKE 'AM%';

INSERT INTO infoArtesM (codigo_actividad, descripcion, cuota)
VALUES ('AM00003', 'TAEKWONDO', 12.5);

UPDATE infoArtesM
```

```
SET codigo_actividad = 'MA00003'

WHERE codigo_actividad = 'AM00003';


ROLLBACK;
```

Si ahora pongo esto:

```
CREATE OR REPLACE VIEW infoArtesM AS

SELECT *

FROM ACTIVIDADES

WHERE codigo_actividad LIKE 'AM%'

WITH CHECK OPTION;
```

Y vuelvo a poner el insert y el update que hemos hecho antes, me da el mensaje de error siguiente: *"violación de la cláusula-WHERE en la vista WITH CHECK OPTION".*

4) Crear una vista de las actividades, con comprobación de actualización, cuya cuota sea superior a 1000 ptas. Intentar realizar después las siguientes inserciones:

- E00001, BALONCESTO, 600
- E00002, FUTBOL, 1100

```
CREATE VIEW activMayor6 AS

SELECT *

FROM ACTIVIDADES

WHERE cuota > 1000

WITH CHECK OPTION;


INSERT INTO activMayor6 (codigo_actividad, descripcion, cuota)

VALUES ('E00001', 'BALONCESTO', 600);


INSERT INTO activMayor6 (codigo_actividad, descripcion, cuota)

VALUES ('E00002', 'FUTBOL', 1100);
```

¿Qué es un índice?

Un índice (o KEY, o INDEX) es un grupo de datos que MySQL asocia con una o varias columnas de la tabla. En este grupo de datos aparece la relación entre el contenido y el número de fila donde está ubicado.

Los índices -como los índices de los libros- sirven para agilizar las consultas a las tablas, evitando que mysql tenga que revisar todos los datos disponibles para devolver el resultado.

Podemos crear el índice a la vez que creamos la tabla, usando la palabra INDEX seguida del nombre del índice a crear y columnas a indexar (que pueden ser varias):
INDEX nombre_indice (columna_indexada, columna_indexada2...)

La sintaxis es ligeramente distinta según la clase de índice:

```
PRIMARY KEY (nombre_columna_1 [,nombre_columna2...])
UNIQUE INDEX nombre_indice (columna_indexada1 [,columna_indexada2 ...])
INDEX nombre_index (columna_indexada1 [,columna_indexada2...])
```

Podemos también añadirlos a una tabla después de creada:

ALTER TABLE nombre_tabla ADD INDEX nombre_indice (columna_indexada);

Si queremos eliminar un índice: ALTER TABLE tabla_nombre DROP INDEX nombre_indice

¿para que sirven ?

Los index permiten mayor rapidez en la ejecución de las consultas a la base de datos tipo SELECT ... WHERE

La regla básica es pues crear tus índices sobre aquellas columnas que vayas a usar con una cláusula WHERE, y no crearlos con aquellas columnas que vayan a ser objeto de un SELECT: SELECT texto from tabla_libros WHERE autor = Vazquez; En este ejemplo, la de autor es una columna buena candidata a un índice; la de texto, no.

Otra regla básica es que son mejores candidatas a indexar aquellas columnas que presentan muchos valores distintos, mientras que no son buenas candidatas las que tienen muchos valores idénticos, como por ejemplo sexo (masculino y femenino) porque cada consulta implicará siempre recorrer prácticamente la mitad del índice.

La regla de la izquierda

Si necesitamos un select del tipo SELECT ... WHERE columna_1 = X AND columna_2 = Y y ya tenemos un INDEX con la columna_1, podemos crear un segundo índice con la columna 2, o mejor todavía, crear un único indice combinado con las columnas 1 y 2. Estos son los índices multicolumna, o compuestos.

No obstante si tienes índices *multicolumna* y los utilizas en las clausulas WHERE, debes incluir siempre de izquierda a derecha las columnas indexadas; o el índice NO se usará:

Supongamos un INDEX usuario (id, name, adress), y una cláusula SELECT ... WHERE NAME = x. Este Select no aprovechará el índice. Tampoco lo haría un SELECT ... WHERE ID =X AND ADRESS = Y. Cualquier consulta que incluya una columna parte del index sin incluir además las columnas a su izquierda, no usará el índice.

Por tanto en nuestro ejemplo solo sacarían provecho del índice las consultas SELECT ... WHERE ID = x, o WHERE ID = X AND NAME = y o WHERE ID = x AND NAME = y AND ADRESS = Z

Cuando un índice contiene mas de una columna, cada columna es leida por el orden que ocupa de izquierda a derecha, y a efectos prácticos, cada columna (por ese orden) es como si constituyera su propio índice. Esto significa que en el ejemplo anterior, no haría falta crear otro INDEX ID (id) ya que podríamos usar nuestro INDEX USUARIO simplemente con la cláusula SELECT ... WHERE ID = X;

Puedes ver si tu llamada `sql` usa o no los índices correctos anteponiendo a `select` la orden `explain`:
`EXPLAIN SELECT * FROM mitable WHERE ....`
Y para ser sinceros, usando explain para comprobar el uso de índices en distintos selects con índices multicolumna, he obtenido resultados poco consistentes con la '*regla de la izquierda*' ya que en muchos casos parece que se usaban índices que teóricamente no debían estar disponibles ... posiblemente un caso de mala configuración en mi tabla-test

Tipos de índice

En algunas bases de datos existen diferencias entre KEY e INDEX. No así en MySQL donde son sinónimos.

Un índice que sí es especial es el llamado PRIMARY KEY. Se trata de un índice diseñado para consultas especialmente rápidas. Todos sus campos deben ser UNICOS y no admite NULL

Un índice UNIQUE es aquel que no permite almacenar dos valores iguales.

Los índices FULL TEXT permiten realizar búsquedas de palabras. Puedes crear índices FULLTEXT sobre columnas tipo CHAR, VARCHAR o TEXT.

Una vez creado puedes hacer búsquedas del tipo:
```
SELECT * FROM nombre_tabla WHERE MATCH(nombre_indice_fulltext)
AGAINST('palabra_a_buscar');
```

Algunas limitaciones de los índices fulltext: solo busca por palabras completas. `índice` no encontrará `índices`. No se indexan las palabras de menos de cuatro letras. No se indexan columnas que contengan menos de tres filas, ni palabras que aparezcan en la mitad o más de las filas. Las palabras separadas por guiones se cuentan como dos palabras.

Los índices ordinarios no tienen restricciones en cuanto a la existencia de valores idénticos o nulos. Una posibilidad interesante, si pensamos crear un índice sobre columnas CHAR y VARCHAR es la de limitar el campo a indexar. Por ejemplo, cada entrada en la columna puede ser de hasta 40 caracteres y nosotros indexar únicamente los primeros 10 de cada una. Para crear estos índices basta con indicar entre paréntesis el numero de caracteres a indexar después del nombre de la columna:
```
ALTER TABLE libros ADD INDEX idx_autor(nombre(10), apellidos(10));
```

Desventajas de los índices

Los índices se actualizan cada vez que se modifica la columna o columnas que utiliza. Por ello no es aconsejable usar como índices columnas en las que serán frecuentes operaciones de escritura (INSERT, UPDATE, DELETE).

Tampoco tendría sentido crear índices sobre columnas cuando cualquier select sobre ellos va a devolver una gran cantidad de resultados; por ejemplo una columna booleana que admita los valores Y/N. En fin, tampoco es necesario usar índices en tablas demasiado pequeñas, ya que en estos casos no hay ganancia de rapidez frente a una consulta normal.

Finalmente, los índices ocupan espacio. A veces, incluso más que la tabla de datos.

Ejemplos:

Para definir estos índices se usan indistintamente las opciones *KEY* o *INDEX*.

```
mysql> CREATE TABLE mitabla2 (
    -> id INT,
    -> nombre CHAR(19),
    -> INDEX (nombre));
Query OK, 0 rows affected (0.09 sec)
```

O su equivalente:

```
mysql> CREATE TABLE mitabla3 (
    -> id INT,
    -> nombre CHAR(19),
    -> KEY (nombre));
Query OK, 0 rows affected (0.09 sec)
```

También podemos crear un índice sobre parte de una columna:

```
mysql> CREATE TABLE mitabla4 (
    -> id INT,
    -> nombre CHAR(19),
    -> INDEX (nombre(4)));
Query OK, 0 rows affected (0.09 sec)
```

Este ejemplo usará sólo los cuatro primeros caracteres de la columna 'nombre' para crear el índice.

Claves únicas

Para definir índices con claves únicas se usa la opción *UNIQUE*.

La diferencia entre un índice único y uno normal es que en los únicos no se permite la inserción de filas con claves repetidas. La excepción es el valor *NULL*, que sí se puede repetir.

```
mysql> CREATE TABLE mitabla5 (
    -> id INT,
    -> nombre CHAR(19),
    -> UNIQUE (nombre));
Query OK, 0 rows affected (0.09 sec)
```

Una clave primaria equivale a un índice de clave única, en la que el valor de la clave no puede tomar valores *NULL*. Tanto los índices normales como los de claves únicas sí pueden tomar valores *NULL*.

Por lo tanto, las definiciones siguientes son equivalentes:

```
mysql> CREATE TABLE mitabla6 (
    -> id INT,
    -> nombre CHAR(19) NOT NULL,
    -> UNIQUE (nombre));
Query OK, 0 rows affected (0.09 sec)
```

Y:

```
mysql> CREATE TABLE mitabla7 (
    -> id INT,
    -> nombre CHAR(19),
    -> PRIMARY KEY (nombre));
Query OK, 0 rows affected (0.09 sec)
```

Los índices sirven para optimizar las consultas y las búsquedas de datos. Mediante su uso es mucho más rápido localizar filas con determinados valores de columnas, o seguir un determinado orden. La alternativa es hacer búsquedas secuenciales, que en tablas grandes requieren mucho tiempo.