



**Universidad  
Europea de Madrid**

**LAUREATE** INTERNATIONAL UNIVERSITIES

**SQL. DEFINICIÓN DE DATOS**

**ÍNDICES Y VISTAS**

**Índice**

Índices y vistas .....	3
Definición de índice .....	4
Creación de índices.....	5
Eliminación de índices .....	6
Definición de vista .....	7
Creación de vistas.....	9
Ejemplos de creación de vistas.....	10
Eliminación de vistas .....	13
Actualización de vistas.....	15
El problema de las filas migratorias .....	16
Resumen .....	17

#### Índices y vistas

Ya sabemos cómo utilizar el LDD de SQL para crear el esquema conceptual de la base de datos, modificarlo y chequear la integridad. El conjunto de datos a los que puede acceder un usuario componen la vista que de la base de datos tiene el mismo. La creación de **vistas** está relacionada con la definición de autorizaciones. Para completar la política de autorizaciones será necesario realizar otras operaciones que ya no es posible llevar a cabo con el LDD. En estos casos, será la aplicación de usuario la encargada de realizarlas.



En ocasiones, es conveniente acelerar el tiempo de ejecución de ciertas consultas que, por ser muy comunes ó por conllevar mucha información, son claves en nuestra base de datos.

Para ello se utilizan **índices**.

En este tema aprenderás a:

- Definir el conjunto de datos a los que puede acceder un usuario.
- Crear índices y vistas.
- Eliminar índices y vistas.
- Diferenciar cuáles son las ventajas e inconvenientes de su uso.

### Definición de índice

Normalmente, la información que se introduce en una base de datos está almacenada de forma permanente en la memoria secundaria. Durante la operación con la BD se producen transferencias de información entre la memoria secundaria y la memoria principal utilizando una memoria temporal llamada **página**. Cada vez que se realiza un acceso a la memoria secundaria para leer una página se ralentiza el tiempo de respuesta del sistema.

El acceso a la información de una tupla de una tabla puede requerir una lectura secuencial de todo el contenido de la base de datos, con sus correspondientes transferencias de información entre memoria secundaria y memoria principal.

Un **índice** es una estructura de datos que permite localizar la situación física de la información a buscar de forma rápida.

El sentido es exactamente el mismo que los índices de los libros. Se podría localizar un término en el libro haciendo una lectura secuencial de todo el libro, pero si éste dispone de un índice, la búsqueda se reduciría a encontrar el término en la página indicada.

El orden establecido en el índice viene determinado por los valores de una o más columnas de una tabla. Estas columnas forman lo que se denomina **campo de indexación**.

Aunque un índice puede llegar a mejorar mucho el tiempo invertido en una consulta, supone una carga extra para el sistema. Por ello, suelen crearse para resolver consultas concretas en tablas que llevan tiempo en uso y cuyo tamaño ha aumentado.



### Creación de índices

A pesar de que el estándar SQL no contempla la creación de índices, la mayoría de los dialectos soportan las siguientes características:

```
CREATE [UNIQUE] INDEX <NombreÍndice> ON <tabla> (<columna1>  
<tipoOrdenación>, ..., <columnan> <tipoOrdenación>)
```

- **<tabla>** indica la tabla sobre la cual vamos a establecer el índice. Sólo pueden crearse índices sobre tablas sencillas (o base), no sobre vistas.
- Las **columnas** que se indiquen formarán el campo de indexación ó clave del índice. Debe especificarse primero la columna más representativa, dejando la menos significativa al final.
- La cláusula **UNIQUE** obligará a que la columna usada para la indexación sea única. Esto es obligatorio para la clave principal.
- **<tipoOrdenación>** permite especificar el orden dentro de cada columna. Puede ser ASC (ascendente) ó DSC (descendente). Por defecto será ascendente.

### Ejemplos

- Dada la **tabla cliente** (DNI, nombre, dirección):

Crea un índice sobre la tabla **clientes**, utilizando **dni** como campo de indexación. Los valores se ordenan de forma ascendente. Como dni es clave primaria de cliente obligamos a que tenga valores distintos con la cláusula **UNIQUE**.

```
CREATE UNIQUE INDEX ClienteID ON cliente(dni);
```

- Dada la **tabla plantilla** (id\_empleado, nombre, departamento, cargo, sueldo):

Crea un índice sobre la tabla **plantilla**, utilizando **departamento** y **nombre** como campos de indexación. Las entradas del índice estarán, por tanto, en orden alfabético por departamento, y dentro de este, ordenados por nombre de empleado.

```
CREATE INDEX IndiceDPTO ON plantilla (departamento, nombre);
```

### Eliminación de índices

Los índices consumen recursos del sistema tanto de almacenamiento como de procesamiento. Hay que pensar bien en qué ocasiones es necesario utilizarlos y cuándo no lo es.

Es posible que exista una consulta muy repetida en el SBD que ya ha dejado de utilizarse ó manejar una tabla que inicialmente tenía mucha información y ya no la tiene.

Ante estas situaciones hay que eliminar el índice que teníamos asociado. La sentencia para eliminar un índice es:

```
DROP INDEX <NombreÍndice>
```

Donde **<nombreÍndice>** especifica el índice que queremos eliminar. De esta forma liberamos los recursos que dicho índice tenía ocupados y, por tanto, mejoramos el rendimiento del sistema.

### Ejemplos

Con las siguientes sentencias, eliminamos los índices creados en los ejemplos de la pantalla anterior.

```
DROP INDEX ClienteID;  
  
DROP INDEX IndiceDPTO;
```



### Definición de vista

Por regla general, no todos los usuarios necesitan ver el contenido de la base de datos completa, más aún, si estos datos son privados.



#### Departamento comercial

En una empresa comercial, el administrativo que da de alta los pedidos, es muy probable que tenga denegado el acceso a la tabla de empleados y salarios de los mismos.



#### Departamento de RRHH

Esta información la manejan los empleados del departamento de RRHH.



#### Departamento de ventas

Estos, por su parte, no tendrán demasiado interés en la información que se maneja en el departamento de ventas.



Para definir el subconjunto de datos a los que cada usuario tiene acceso se utilizan las **vistas**.

Una vista es una **relación virtual** que contiene el subconjunto de datos definidos para un determinado tipo de usuario.

Para especificar cuáles serán los datos en concreto que formarán parte de la vista, se define una sentencia **SELECT** sobre una ó más tablas base ó sobre otras vistas. A esta consulta se le denomina **consulta de definición**.

Normalmente lo que se almacena no es el conjunto de datos en sí, sino la consulta de definición de la vista. De esta forma se genera en el momento en que se hace uso de ella, conteniendo siempre los datos actualizados.



### Creación de vistas

La sentencia para definir vistas es:

```
CREATE VIEW <NombreVista> (<NombreColumna1>,..., <nombreColumnan>
) AS <subconsulta>
```

Donde:

- **NombreVista** es el nombre que le damos a esta tabla virtual.
- **nombreColumna1,...,nombreColumnan** son los nombres que les damos a las columnas en la tabla virtual. Estos nombres no tienen por qué coincidir con los nombres de las columnas de la tabla resultado de la consulta de definición. Si no se establecen se mantienen estos últimos. Sí hay que ponerlos en el caso de que pueda existir ambigüedad en la subconsulta que genera la vista.
- **Subconsulta** es la consulta de definición. Es una sentencia **SELECT** que define el conjunto de tuplas que formarán el contenido de la vista.

### Ejemplo

Dada la relación **empleados** (**dni**, **nombre**, **dirección**, **ciudad**, **teléfono**, **cargo**, **salario**), el administrativo que gestiona los pedidos podrá consultar los nombres de los empleados de la empresa junto con el cargo y el número de teléfono por si necesita contactar con ellos.

La vista sería:

```
CREATE VIEW datosEmpleados(nombre_emp, puesto, telefono) AS
(SELECT nombre, cargo, teléfono FROM empleados);
```



## Ejemplos de creación de vistas

Vamos a utilizar la base de datos de la aerolínea como base para nuestros ejemplos. Una vista puede ser el resultado de:

**Aplicar una selección de filas a una tabla base (vistas horizontales)**

**Ejemplo.** Cada **aeropuerto** necesita saber el **listado de vuelos** en los que va a intervenir. Para ello se crea una vista que contiene la información de los vuelos para cada aeropuerto. Así, el aeropuerto de Barcelona utilizaría esta vista:

**Vista resultado****Ejemplo****Vista resultado**

La vista resultado que se obtiene es (la tabla se ha dividido en dos por tener muchos atributos):

```
CREATE VIEW vuelosBarcelona AS (SELECT * FROM vuelos WHERE
(salida_aeropuerto = 'aeropuertobarce' OR
llegada_aeropuerto='aeropuertobarce' ) ) ;
```

cod_ vuelo	clase	avion	salida_ fecha	salida_ hora	salida_ciudad	salida_aeropuerto	salida_ terminal
309	economica	dc70	4 julio	20:15	barcelona	aeropuertobarce	t2
332	economica	dc76	13 julio	00:36	barcelona	aeropuertobarce	t2

llegada_ fecha	llegada_ hora	llegada_ ciudad	llegada_aeropu erto	llegada_ terminal	empresa	duración
5 julio	00:36	londres	aeropuertolondr	t2	easyjet	2
14 julio	00:20	nigeria	aeropuertoniger	t2	easyjet	8

**Aplicar selección de columnas a una tabla base (vistas verticales)**

**Ejemplo.** Vista reducida de la tabla de vuelos donde se especifique sólo el **código de vuelo**, la **empresa**, la **fecha de salida**, el **aeropuerto de salida** y la **terminal**.

**Vista resultado**

Ejemplo

**Vista resultado**

La vista resultado que se obtiene es:

```
CREATE VIEW Salidas (vuelo, empresa, fecha, aeropuerto, terminal)AS
```

```
(SELECT cod_vuelo, empresa, salida_fecha, salida_aeropuerto, salida_terminal FROM vuelos);
```

vuelo	empresa	fecha	aeropuerto	terminal
123	easyjet	6 julio	aeropuertomadri	t2
189	spanair	6 julio	aeropuertonueva	t4
271	spanair	8 julio	aeropuertomalag	t4
284	iberia	5 julio	aeropuertomalag	t1
309	easyjet	4 julio	aeropuertobarce	t2
332	easyjet	13 julio	aeropuertobarce	t2
337	iberia	3 julio	barajas	t1
571	iberia	12 julio	aeropuertomadri	t1
795	easyjet	9 julio	aeropuertomalag	t2
815	spanair	11 julio	aeropuertomalag	t4
837	iberia	10 julio	aeropuertoniger	t1

**Aplicar una agrupación a una tabla base (vista agrupada)**

**Ejemplo.** Precio total de cada **reserva** calculado en base a la información de las líneas.

**Vista resultado****Ejemplo****Vista resultado**

```
CREATE VIEW PrecioReservas(reserva, precio)
```

```
AS (SELECT cod_reserva, SUM(((tarifa+tarifa*tasas+gastos_gestion)*n_pasajeros) FROM lineas GROUP BY cod_reserva);
```

reserva	precio
1	73
2	390
3	506
4	146
5	9785
6	1071
7	714
8	849
9	1473
10	94
11	643

**Combinar varias tablas base (vistas combinadas)**

**Ejemplo.** Vuelos y comidas especiales que han seleccionado los pasajeros del mismo. La vista sería:

```
CREATE VIEW comidasporvuelo(vuelo, menu) AS (SELECT c.cod_vuelo, a.comida_especial FROM compuesta_por c, aparece a WHERE c.cod_reserva=a.cod_reserva ORDER BY c.cod_vuelo);
```

## Eliminación de vistas

La sentencia que se utiliza para eliminar vistas es:

```
DROP VIEW <Vista> <acción>
```

Donde:


- **<vista>** es el nombre de la vista que queremos eliminar.
- **<acción>** define qué pasará con los objetos creados en la base de datos a partir de esta vista.  
Hay dos opciones de acción:
  - **CASCADE.** En este caso no sólo se elimina esta vista sino que también se eliminarán todas las vistas que se hayan definido en base a ésta.
  - **RESTRICT.** En este caso el SGBD no permitirá eliminar ninguna vista que tenga objetos en la base de datos pendiente de ella.


## Ejemplo

Con las siguientes sentencias, eliminamos las vistas incluidas en los ejemplos de la pantalla anterior (a la derecha de la tabla).

DROP VIEW vuelosBarcelona;	<pre>CREATE VIEW vuelosBarcelona AS (SELECT * FROM vuelos WHERE (salida_aeropuerto = 'aeropuertobarce' OR llegada_aeropuerto='aeropuertobarce'));</pre>
DROP VIEW salidas;	
DROP VIEW precioreservas;	
DROP VIEW comidasporvuelo;	

DROP VIEW vuelosBarcelona;	<pre>CREATE VIEW Salidas (vuelo, empresa, fecha, aeropuerto, terminal) AS (SELECT cod_vuelo, empresa, salida_fecha, salida_aeropuerto, salida_terminal FROM vuelos);</pre>
DROP VIEW salidas;	
DROP VIEW precioreservas;	
DROP VIEW comidasporvuelo;	

	<b>DROP VIEW vuelosBarcelona;</b>	<pre>CREATE VIEW PrecioReservas(reserva, precio) AS (SELECT                                cod_reserva, SUM((tarifa+tarifa*tasas+gastos_gestion)*n_pasajeros) FROM lineas GROUP BY cod_reserva);</pre>
<b>DROP VIEW salidas;</b>		
<b>DROP VIEW precioreservas;</b>		
<b>DROP VIEW comidasporvuelo;</b>		

<b>DROP VIEW vuelosBarcelona;</b>	<pre>CREATE VIEW comidasporvuelo(vuelo, menu) AS (SELECT c.cod_vuelo, a.comida_especial FROM compuesta_por c, aparece a WHERE c.cod_reserva=a.cod_reserva ORDER BY c.cod_vuelo);</pre>	
 <b>DROP VIEW salidas;</b>		
<b>DROP VIEW precioreservas;</b>		
<b>DROP VIEW comidasporvuelo;</b>		

### Actualización de vistas

Las actualizaciones realizadas sobre una tabla base afectan inmediatamente a la vista creada sobre dicha tabla. Sin embargo, el caso contrario no siempre es posible.

En general, se debe cumplir la siguiente regla básica: para que una vista sea actualizable, el SGBD debe poder localizar la fila y columna de la tabla base que correspondan a cada fila o columna de la vista.

#### Ejemplo

Dada la **tabla cliente** (DNI, nombre, apellido, dirección, ciudad) y las siguientes vistas que contiene el nombre, apellido y calle de los clientes de Murcia:

```
CREATE VIEW clientesMurcia (nombre, apellido, calle) AS SELECT
nombre, apellido, dirección FROM cliente WHERE ciudad =
'Murcia';
```

Dado que una vista se puede utilizar en todos aquellos lugares dentro de sentencias de SQL donde se utilizan tablas, se puede utilizar la siguiente sentencia para insertar los datos de un cliente:

```
INSERT INTO clientesMurcia VALUES ('Alejandro', 'Beltrán',
'Fuentes e');
```

¿Qué ocurre, por ejemplo, con el **dni**, que no puede ser nulo?

Ninguna fila que se añada utilizando la vista puede violar la integridad de la tabla base, por tanto, al insertar estos datos se intentará insertar la tupla correspondiente en **clientes**, estableciendo el **dni** a **NULL**. Como esto no es posible por ser la clave de la relación, el SGBD nos dará un error. No se puede, por tanto, violar ninguna restricción de **NOT NULL**.

### El problema de las filas migratorias

Las filas que contiene una vista deben cumplir la condición **WHERE** de la consulta de definición de la vista. Al modificarse una fila puede ocurrir que:

- Deje de cumplir la condición y abandone la vista.
- Pase a satisfacer la condición y entre en la vista.

Estas filas son llamadas **filas migratorias**.

Se puede añadir la opción **WITH CHECK OPTION** a la sentencia de creación de consultas para evitar que una fila migre **fuera** de la vista. La sintaxis de la sentencia en este caso sería:

```
CREATE VIEW <NombreVista> (<NombreColumnal>,..., <nombreColumnan>)  
AS <subconsulta> WITH CHECK OPTION
```

Esto hace que cualquier **INSERT** o **UPDATE** sobre una vista que viole la cláusula **WHERE** sea cancelado.

### Ejemplo

```
CREATE VIEW vuelosBarcelona AS (SELECT * FROM vuelos WHERE  
(salida_aeropuerto = 'aeropuertobarce' OR  
llegada_aeropuerto = 'aeropuertobarce')) WITH CHECK OPTION;  
  
UPDATE vuelosbarcelona SET salida_aeropuerto = 'barajas' WHERE  
cod_vuelo = '309';
```

La respuesta del SGBD es:

```
CHECK OPTION failed 'aerolinea.vuelosbarcelona'
```



## Resumen

En este tema hemos visto que el LDD de SQL permite la creación y eliminación de índices y vistas.

- Un **índice** es una estructura de datos que permite localizar rápidamente la situación física en memoria de los datos que queremos recuperar en una consulta, y así acelerar el tiempo de ejecución de la misma.
- Los índices se definen a través de la siguiente sintaxis, donde **ordenación** puede ser **ascendente** ASC ó **descendente** DSC:

```
CREATE INDEX <indice> ON <tabla> (<coll> <Ordenación>, ..., <coln>  
<Ordenación>)
```

- Para **eliminar un índice** se usa la sintaxis:

```
DROP INDEX <indice>
```

- Una vista es una tabla virtual que contiene el conjunto de datos permitidos para un usuario. Se utilizan para definir autorizaciones. En la siguiente sintaxis de la sentencia de creación de vistas, subconsulta es la **consulta de definición** de la vista y WITH CHECK OPTION evita la aparición de **filas migratorias**.

```
CREATE VIEW <vista>(<coll>, ..., <coln>) AS <subconsulta> WITH  
CHECK OPTION
```

- Para **eliminar una vista** se utiliza la siguiente sentencia. Acción puede ser **RESTRICT** ó **CASCADE**.

```
DROP VIEW <vista> <acción>
```