DISEÑO FÍSICO DE BBDD. LENGUAJE DE DEFINICIÓN DE DATOS

Contenido

1. INTRODUCCIÓN

En la fase de análisis hemos realizado la E.R.S. A partir de dicha especificación de requisitos, en la unidad anterior, aprendimos a realizar el diseño conceptual de una BD mediante el Modelo E/R y el Modelo E/R extendido respectivamente.

También vimos cómo hacer el modelo lógico mediante el modelo relacional que se obtenía a partir del modelo E/R y aprendimos a comprobar que dicho modelo estaba normalizado.

Siguiendo con el proceso de desarrollo, lo que debemos hacer ahora es pasar al diseño físico de la BD. Es decir, implementar la Base de Datos. Para ello, se programarán las

diferentes tablas que constituirán la Base de Datos. Todo ello se hará programando en el lenguaje más extendido para la definición y manipulación de datos en SGBDR: **SQL**.

2. MOTORES DE ALMACENAMIENTO EN MySQL

MySQL soporta diferentes tipos de almacenamiento de tablas (motores de almacenamiento o storage engines, en inglés). Y cuando se crea una tabla hay que especificar en qué sistema de los posibles queremos crear.

Por defecto, MySQL a partir de la versión 5.5.5 crea las tablas de tipo InnoDB, que es un sistema transaccional, es decir, que soporta las características que hacen que una base de datos pueda garantizar que los datos se mantendrán consistentes.

Las propiedades que garantizan los sistemas transaccionales son las características llamadas ACID. ACID es el acrónimo inglés de atomicity, consistency, isolation, Durability:

Atomicidad: se dice que un SGBD garantiza atomicidad si cualquier transacción o bien finaliza correctamente (commit), o bien no deja ningún rastro de su ejecución (rollback).

Consistencia: se habla de consistencia cuando la concurrencia de diferentes transacciones no puede producir resultados anómalos.

Aislamiento (o aislamiento): cada transacción dentro del sistema se debe ejecutar como si fuera la única que se ejecuta en ese momento.

Definitividad: si se confirma una transacción, en un SGBD, el resultado de esta debe ser definitivo y no se puede perder.

Sólo el motor InnoDB permite crear un sistema transaccional en MySQL. Los otros tipos de almacenamiento no son transaccionales y no ofrecen control de integridad en las bases de datos creadas.

Evidentemente, este sistema (InnoDB) de almacenamiento es lo que a menudo interesará utilizar para las bases de datos que creamos, pero puede haber casos en que sea interesante considerar otros tipos de motores de almacenamiento. Por ello, MySQL también ofrece otros sistemas tales como, por ejemplo:

MyISAM: era el sistema por defecto antes de la versión 5.5.5 de MySQL. Se utiliza mucho en aplicaciones web y en aplicaciones de almacén de datos (datawarehousing).

Memory: este sistema almacena todo en memoria RAM y, por tanto, se utiliza para sistemas que requieran un acceso muy rápido a los datos.

Merge: agrupa tablas de tipo MyISAM para optimizar listas y búsquedas. Las tablas que hay que agrupar deben ser similares, es decir, deben tener el mismo número y tipo de columnas.

Para obtener una lista de los motores de almacenamiento soportados por la versión MySQL que tenga instalada, el comando SHOW ENGINES.

3. EL LENGUAJE SQL

3.1. Historia

El nacimiento del lenguaje SQL data de 1970 cuando E. F. Codd publica su libro: «Un modelo de datos relacional para grandes bancos de datos compartidos». Ese libro dictaría las direcrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el Standard English Query Language (Lenguaje Estándar Inglés para Consultas) al que se le llamó SEQUEL. Más adelante se le asignaron las siglas SQL (Standard Query Language, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando secuel. En español se pronuncia esecuele.

En 1979 Oracle presenta la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI. En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.

En 1989 aparece el estándar ISO (y ANSI) llamado SQL89 o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada SQL92. En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,… y otras características de las bases de datos objeto-relacionales; dicho estándar se conoce como SQL99. El último estándar es el del año 2011 (SQL2011) Elementos de SQL

SQL se basa en la Teoría Matemática del Álgebra Relacional. El lenguaje SQL consta de varios elementos:

- **Lenguaje de definición de datos (DDL)**: proporciona órdenes para definir, modificar o eliminar los distintos objetos de la base de datos (tablas, vistas, índices…).
- **Lenguaje de Manipulación de Datos (DML)**: proporciona órdenes para insertar, suprimir y modificar registros o filas de las tablas. También contempla la realización de consultas sobre la BD.
- **Lenguaje de Control de Datos (DCL)**: permite establecer derechos de acceso de los usuarios sobre los distintos objetos de la base de datos. Lo forman las instrucciones **GRANT** y **REVOKE**.
- Oracle contempla además **sentencias para transacciones**. Administran las modificaciones creadas por las instrucciones DML. Lo forman las instrucciones **ROLLBACK**, **COMMIT** y **SAVEPOINT**.

3.2. Proceso de ejecución de sentencia SQL

El proceso de una instrucción SQL es el siguiente:

1. Se analiza la instrucción. Para comprobar la sintaxis de la misma
2. Si es correcta se valora si los metadatos de la misma son correctos. Se comprueba esto en el diccionario de datos.
3. Si es correcta, se optimiza, a fin de consumir los mínimos recursos posibles.
4. Se ejecuta la sentencia y se muestra el resultado al emisor de la misma.

4. LENGUAJE DE DEFINICIÓN DE DATOS: DDL

4.1. Tipos de datos

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

- **Tipos de Fecha**
- **Tipos numéricos**
- **Tipos de Cadena**

Puedes buscar información acerca de los tipos se datos y sus características en **MySQL 5.0 Reference Manual** (Capítulo 11)

4.1.1 Tipos de Fecha

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

4.1.2. Tipos numéricos

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0

NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

Una columna entera puede tener el atributo adicional AUTO_INCREMENT. Cuando inserta un valor de NULL (recomendado) o 0 en una columna AUTO_INCREMENT autoindexada, la columna se asigna al siguiente valor de secuencia. Típicamente esto es value+1, donde value es el mayor valor posible para la columna en la tabla. Secuencias AUTO_INCREMENT comienzan con 1. Por ejemplo:

```
CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,…)
```

4.1.3 Tipos de cadena

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

4.2. Creación, Modificación y Eliminación de tablas

En este apartado veremos los comandos SQL que se utilizarán para crear y modificar la definición de una tabla, así como para eliminarla de la base de datos.

Nos referiremos al sistema MySQL pudiendo haber diferencias respecto a otros SGBD.

4.2.1. Creación de Tablas

El nombre de las tablas debe cumplir las siguientes reglas:

- Deben comenzar con una letra
- No deben tener más de 30 caracteres
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo $ y #, pero esos se utilizan de manera especial por lo que no son recomendados)
- No puede haber dos tablas con el mismo nombre para el mismo usuario (pueden coincidir los nombres si están en distintos esquemas)
- No puede coincidir con el nombre de una palabra reservada de SQL

Para la creación de tablas con SQL se utiliza el comando **CREATE TABLE**. Este comando tiene una sintaxis más compleja de la que aquí se expone, pero vamos a comenzar por la sintaxis básica. Sintaxis básica de creación de tablas:

```
CREATE TABLE nombre_tabla [IF NOT EXISTS] (
  columna1  tipo_dato  [ restricciones de columna1 ],
  columna2  tipo_dato  [ restricciones de columna2 ],
  columna3  tipo_dato  [ restricciones de columna3 ],
  ...
  [ restricciones de tabla ]
);
```

Para realizar las separaciones se utiliza la coma. **La última línea, antes del paréntesis de cierre, no lleva coma**.

Donde las restricciones de columna tienen la siguiente sintaxis:

```
{[NOT] NULL | UNIQUE | PRIMARY KEY | DEFAULT valor | CHECK
(condición)}
```

Nota: La cláusula **CHECK** se parsea pero se ignora en todos los motores de almacenamiento

Y las restricciones de tabla tienen la siguiente sintaxis:

```
CONSTRAINT [nombre_restricción] {
  PRIMARY KEY (columna1 [,columna2] ... )
| UNIQUE (columna1 [,columna2] ... )
| FOREIGN KEY (columna1 [,columna2] ... )    REFERENCES nombre_tabla
(columna1 [,columna2] ... )    [ON DELETE {CASCADE | SET NULL}]
| CHECK (condición)}
```

Obligatoriamente debemos crear una restricción de tabla cuando una misma restricción afecte a varias columnas. Por ejemplo, si tenemos una clave primaria compuesta por varios campos, debemos establecer una restricción de tabla, no de columna.

El significado de las distintas opciones que aparecen en la sintaxis CREATE TABLE es:

- **PRIMARY KEY**: establece ese atributo o conjunto de atributos como la clave primaria de la tabla. Esta restricción ya implica las restricciones UNIQUE y NOT NULL.

- **UNIQUE**: impide que se introduzcan valores repetidos para ese atributo o conjunto de atributos. No se puede utilizar junto con PRIMARY KEY. Se utiliza para claves alternativas.
- **NOT NULL**: evita que se introduzcan filas en la tabla con valor NULL para ese atributo. No se utiliza con PRIMARY KEY.
- **DEFAULT** valor_por_defecto: permite asignar un valor por defecto al campo que se está definiendo. El valor no puede ser él devuelto por una función.
- **CHECK** (condición): permite establecer condiciones que deben cumplir los valores de la tabla que se introducirán en dicha columna.

  Nota: La cláusula **CHECK** se parsea pero se ignora en todos los motores de almacenamiento

- **FOREIGN KEY**: define una clave externa de la tabla respecto de otra tabla. Esta restricción especifica una columna o una lista de columnas como clave externa de una tabla referenciada. **No se puede definir una restricción de integridad referencial que se refiere a una tabla antes de que dicha tabla haya sido creada.** Es importante resaltar que una clave externa debe referenciar a una clave primaria completa de la tabla padre, y nunca a un subconjunto de los atributos que forman esta clave primaria.
  - **ON DELETE CASCADE**: especifica que se mantenga automáticamente la integridad referencial borrando los valores de la llave externa correspondientes a un valor borrado de la tabla referenciada (tabla padre). Si se omite esta opción no se permitirá borrar valores de una tabla que sean referenciados como llave externa en otras tablas.
  - **ON DELETE SET NULL**: especifica que se ponga a NULL los valores de la llave externa correspondientes a un valor borrado de la tabla referenciada (tabla padre).
  - **ON DELETE NO ACTION**. Opción por defecto. Un intento de borrar una fila o actualizar un valor de clave primaria no será permitido si la fila está referenciada. **RESTRICT** sería equivalente en MySQL

Nota

El valor predeterminado es que las filas de la tabla principal no se pueden eliminar si existe una fila en la tabla secundaria que se refiere a esta fila principal, si no indicamos ON DELETE CASCADE o ON DELETE SET NULL. El estándar SQL define muchas más opciones.

Nota

El estándar SQL define 5 opciones para manejar esta situación de tablas principal/secundaria de varias maneras. Estas opciones son:

- **ON DELETE CASCADE**: si se elimina una fila de la tabla principal, se eliminan todas las filas coincidentes en la tabla secundaria.
- **ON DELETE SET NULL**: si se elimina una fila de la tabla principal, todas las columnas de referencia en todas las filas coincidentes de la tabla secundaria se establecen en NULL.
- **ON DELETE SET DEFAULT**: si se elimina una fila de la tabla principal, todas las columnas de referencia en todas las filas coincidentes de la tabla secundaria se configuran en el valor predeterminado de la columna.

- **ON DELETE RESTRICT**: está prohibido eliminar una fila de la tabla principal si esa fila tiene alguna fila coincidente en la tabla secundaria. El punto en el tiempo cuando realiza la comprobación se puede aplazar hasta que se realice COMMIT.
- **ON DELETE NO ACTION** (el valor predeterminado): se prohíbe eliminar una fila de la tabla primaria si esa fila tiene filas coincidentes en la tabla secundaria.

Análoga a la opción ON DELETE hay una opción ON UPDATE. Define las mismas 5 opciones para el caso de cambiar una columna en la tabla principal a la que hace referencia la columna de una tabla secundaria.

- **ON UPDATE CASCADE**: Cualquier cambio en una columna referenciada en la tabla primaria provoca el mismo cambio en la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria.
- **ON UPDATE SET NULL**: Cualquier cambio en una columna referenciada en la tabla primaria provoca que la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria se establezca como nula.
- **ON UPDATE SET DEFAULT**: Cualquier cambio en una columna referenciada en la tabla principal provoca que la columna de referencia correspondiente en las filas coincidentes de la tabla de secundaria se establezca en su valor predeterminado.
- **ON UPDATE RESTRICT**: está prohibido cambiar una fila de la tabla principal si esa fila tiene filas coincidentes en la tabla secundaria. El punto en el tiempo cuando se realiza la comprobación se puede aplazar hasta que se realice COMMIT.
- **ON UPDATE NO ACTION** (valor predeterminado): está prohibido cambiar una fila de la tabla principal si esa fila tiene alguna fila coincidente en la tabla secundaria.

Si ON DELETE o ON UPDATE no están especificados, se producirá la acción predeterminada NO ACTION. En algunos sistemas, NO ACTION se implementa en el sentido de la opción RESTRICT. **Como veis NO hay diferencia a efectos prácticos entre una y otra opción.**

En la definición de una tabla pueden aparecer varias cláusulas FOREIGN KEY, tantas como claves externas tenga la tabla, sin embargo sólo puede existir una llave primaria, si bien esta llave primaria puede estar formada por varios atributos.

Ejemplos:

```
CREATE TABLE usuarios (
  id            INTEGER       PRIMARY KEY,
  dni           CHAR(9)       UNIQUE,
  nombre         VARCHAR(50) NOT NULL,
  edad          INTEGER       CHECK (edad>=0 and edad<120)
);
```

En el caso anterior de que queramos asignar un nombre a las restricciones deberemos definirlas después de definir los campos. Así, si deseamos modificar posteriormente el diseño de la tabla, será más fácil gestionar las restricciones.

Nota

El comando **DESCRIBE**, permite obtener la estructura de una tabla.

Ejemplo:

```
DESCRIBE COCHES;
```

Y aparecerán los campos de la tabla COCHES

Criterios de notación para los nombres de restricciones (Aconsejables)

Para la Restricción de Clave principal (solo una en cada tabla):

```
CONSTRAINT tabla_campo_pk PRIMARY KEY ...
```

Para Restricciones de Clave foránea (puede haber varias en cada tabla):

```
CONSTRAINT tabla_campo_fk1 FOREING KEY ...
CONSTRAINT tabla_campo_fk2 FOREING KEY ...
CONSTRAINT tabla_campo_fk3 FOREING KEY ...
```

Para Restricciones de tipo UNIQUE (puede haber varias en cada tabla)

```
CONSTRAINT tabla_campo_uq1 UNIQUE ...
CONSTRAINT tabla_campo_uq2 UNIQUE ...
...
```

Ejemplo:

```
CREATE TABLE COCHES (
  matricula              VARCHAR(8),
  marca                  VARCHAR(15) NOT NULL,
  color                  VARCHAR(15),
  codTaller              VARCHAR(10),
  codProp                VARCHAR(10),
```

```
  CONSTRAINT coches_mat_pk PRIMARY KEY (matricula),
  CONSTRAINT coches_codtaller_fk1 FOREIGN KEY (codTaller)
      REFERENCES TALLER(codTaller),
  CONSTRAINT coches_codprop_fk2 FOREIGN KEY (codProp)
      REFERENCES PROPIETARIO(codProp),
  CONSTRAINT coches_color_ck1
      CHECK (color IN ('ROJO','AZUL','BLANCO','GRIS','VERDE','NEGRO'))
);
```

4.2.2. Eliminación de Tablas

La sentencia en SQL para eliminar tablas es **DROP TABLE**. Su sintaxis es:

```
DROP TABLE nombre_tabla
[ CASCADE|RESTRICT];
```

`RESTRICT` y `CASCADE` se permiten para hacer la portabilidad más fácil. De momento, no hacen nada.

El borrado de una tabla es irreversible y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación. Al borrar una tabla se borran todos los datos que contiene.

Ejemplos:

```
DROP TABLE COCHES;
```

Se eliminará la tabla COCHES, siempre que su clave principal no sea clave foránea de ninguna tabla de la BD.

4.2.3. Modificación de Tablas

Cambiar de nombre una tabla

La orden RENAME permite el cambio de nombre de cualquier objeto. Sintaxis:

```
RENAME nombre  TO nombre_nuevo;
```

Ejemplo:

```
RENAME COCHES TO AUTOMOVILES;
```

Cambia el nombre de la tabla COCHES y a partir de ese momento se llamará AUTOMOVILES

Borrar el contenido de una tabla

La orden TRUNCATE TABLE seguida del nombre de una tabla, hace que se elimine el contenido de la tabla, pero no la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla. (**Esta orden no puede anularse con un ROLLBACK**)

Ejemplo:

```
TRUNCATE TABLE AUTOMOVILES;
```

Borra los datos de la tabla AUTOMOVILES.

4.2.3.1 Trabajo con columnas y restricciones

La cláusula **ALTER TABLE** permite hacer cambios en la estructura de una tabla: añadir columna, borrar columna, modificar columna.

Añadir Columnas

```
ALTER TABLE nombre ADD (
  columna1  tipo  [ restricciones ][,
  columna2  tipo  [ restricciones ]
  ... ]
);
```

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos y sus propiedades si es necesario (al estilo de CREATE TABLE). Las nuevas columnas se añaden al final, no se puede indicar otra posición.

Ejemplos:

Añadimos la columna “fechaMatric” a la tabla VEHÍCULOS:

```
ALTER TABLE VEHICULOS ADD ( fechaMatric DATE );
```

Añadimos las columnas “fechaMatric” y “tipoFaros” a la tabla VEHÍCULOS:

```
ALTER TABLE VEHICULOS ADD (
```

```
  fechaMatric           DATE,
  tipoFaros              VARCHAR(20) NOT NULL
);
```

Borrar Columnas

```
ALTER TABLE nombre_tabla DROP (nombre_columna, nombre_columna2, ...);
```

Elimina la columna indicada de manera irreversible e incluyendo los datos que contenía. No se pueden eliminar todas las columnas, para la última columna habrá que usar DROP TABLE.

Ejemplo:

```
ALTER TABLE VEHICULOS DROP (tipoFaros);
```

Borra la columna “tipoFaros” de la tabla VEHICULOS y los datos que contuviera de manera irreversible.

Modificar columnas

Permite cambiar el tipo de datos y propiedades de una determinada columna. Sintaxis:

```
ALTER TABLE nombre_tabla MODIFY (
  columna1  tipo_dato  [ restricciones de columna1 ][,
  columna2  tipo_dato  [ restricciones de columna2 ]
  ... ]
);
```

Ejemplo:

```
ALTER TABLE AUTOMOVILES
MODIFY (color  VARCHAR(20) NOT NULL, codTaller  VARCHAR(15));
```

Modifica dos campos o columnas de la tabla AUTOMOVILES cambiando su tamaño y además en Color, añadiendo la condición de que sea no nulo.

Los cambios que se permiten son:

- Incrementar precisión o anchura de los tipos de datos

- Sólo se puede reducir la anchura máxima de un campo si esa columna posee nulos en todos los registros, o no hay registros.
- Se puede pasar de CHAR a VARCHAR y viceversa (si no se modifica la anchura).

Añadir o Modificar Restricciones

Sabemos que una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla. A cada restricción se le pone un nombre, en el caso de no poner un nombre (en las que eso sea posible) entonces el propio le coloca el nombre que es un nemotécnico con el nombre de tabla, columna y tipo de restricción.

Hemos visto que se pueden añadir al crear la tabla, o bien, podemos hacerlo mediante modificación posterior de la tabla. También se puede modificar una restricción creada. Su sintaxis general es:

```
ALTER TABLE nombre_tabla { ADD | MODIFY } (
  CONSTRAINT nombre_restricción1   tipo_restricción  (columnas)
  [, CONSTRAINT nombre_restricción2   tipo_restricción  (columnas) ]
...
);
```

Borrar Restricciones

Su sintaxis es la siguiente:

```
ALTER TABLE nombre_tabla
DROP {
  PRIMARY KEY
| UNIQUE  (columnas)
| CONSTRAINT nombre_restricción
}
```

Cambiar de nombre la Restricciones

Para hacerlo se utiliza este comando:

```
ALTER TABLE nombre_tabla
RENAME CONSTRAINT nombre_restricción TO nombre_restricción_nuevo;
```

4.3. Creación, Modificación y Eliminación de 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 la instrucción SELECT necesaria para crear la vista, eso asegura que los datos sean coherentes al utilizar los datos almacenados en las tablas.

Las vistas se emplean para:

- Realizar consultas complejas más fácilmente
- Proporcionar tablas con datos completos
- Utilizar visiones especiales de los datos

Hay dos tipos de vistas:

- **Simples**. Las forman una sola tabla y no contienen funciones de agrupación. Su ventaja es que permiten siempre realizar operaciones DML sobre ellas.
- **Complejas**. Obtienen datos de varias tablas, pueden utilizar funciones de agrupación. No siempre permiten operaciones DML.

4.3.1. Creación de Vistas

Sintaxis:

```
CREATE [ OR REPLACE ] VIEW nombre_vista [ (alias1 [, alias2] ...) ]
AS SELECT ...
```

- **OR REPLACE**. Especifique OR REPLACE para volver a crear la vista si ya existe. Puede utilizar esta cláusula para cambiar la definición de una vista existente sin eliminar, volver a crear y volver a conceder los privilegios de objeto previamente concedidos.
- alias. Lista de alias que se establecen para las columnas devueltas por la consulta SELECT en la que se basa esta vista. El número de alias debe coincidir con el número de columnas devueltas por SELECT. La sentencia SELECT la trataremos en profundidad en el tema siguiente.

Lo bueno de las vistas es que tras su creación se utilizan como si fueran una tabla. La **vista USER_VIEWS** del diccionario de datos permite mostrar una lista de todas las vistas que posee el usuario actual. La columna TEXT de esa vista contiene la sentencia SQL que se utilizó para crear la vista (sentencia que es ejecutada cada vez que se invoca a la vista).

4.3.2. Eliminación de Vistas

Se utiliza el comando **DROP VIEW**:

```
DROP VIEW nombre_vista;
```

4.3.3. Modificación de Vistas

Sólo se utiliza la instrucción **ALTER VIEW** para recompilar explícitamente una vista que no es válida. Si desea cambiar la definición de una vista se debe ejecutar la sentencia **CREATE OR REPLACE nombre_vista**.

La sentencia ALTER VIEW le permite localizar errores de recompilación antes de la ejecución. Para asegurarse de que la alteración no afecta a la vista u otros objetos que dependen de ella, puede volver a compilar explícitamente una vista después de alterar una de sus tablas base.

Para utilizar la instrucción ALTER VIEW, la vista debe estar en su esquema, o debe tener el privilegio del sistema ALTER ANY TABLE.

4.4. Creación, Modificación y Eliminación de índices

Los índices son objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia.

Se almacenan aparte de la tabla a la que hace referencia, lo que permite crearles y borrarles en cualquier momento.

Lo que realizan es una lista ordenada por la que puede acceder para facilitar la búsqueda de los datos. cada vez que se añade un nuevo registro, los índices involucrados se actualizan a fin de que su información esté al día. De ahí que cuantos más índices haya, más le cuesta a añadir registros, pero más rápidas se realizan las instrucciones de consulta.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por los que los crea el propio SGBD.

4.4.1. Creación de Índices

Aparte de los índices obligatorios comentados anteriormente, se pueden crear índices de forma explícita. Éstos se crean para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

Sintaxis:

```
CREATE INDEX nombre
ON tabla (columna1 [,columna2] ...)
```

Ejemplo:

```
CREATE INDEX nombre_completo
```

```
ON clientes (apellido1, apellido2, nombre);
```

El ejemplo crea un índice para los campos apellido1, apellido2 y nombre. Esto no es lo mismo que crear un índice para cada campo, este índice es efectivo cuando se buscan u ordenan clientes usando los tres campos (apellido1, apellido2, nombre) a la vez.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores
- Contengan una gran cantidad de nulos
- Sean parte habitual de cláusulas WHERE, GROUP BY u ORDER BY
- Sean parte de listados de consultas de grandes tablas sobre las que casi siempre se muestran como mucho un 4% de su contenido.

No se aconseja en campos que:

- Pertenezcan a tablas pequeñas
- No se usen a menudo en las consultas
- Pertenecen a tablas cuyas consultas muestran menos de un 4% del total de registros
- Pertenecen a tablas que se actualizan frecuentemente
- Se utilizan en expresiones

Los índices se pueden crear utilizando expresiones complejas:

```
CREATE INDEX nombre_complejo

ON clientes (UPPER(nombre));
```

Esos índices tienen sentido si en las consultas se utilizan exactamente esas expresiones. Para ver la lista de índices en se utiliza la vista USER_INDEXES. Mientras que la **vista USER_IND_COLUMNS** muestra la lista de columnas que son utilizadas por índices.

4.4.2. Eliminación de Índices

La instrucción DROP INDEX seguida del nombre del índice permite eliminar el índice en cuestión.

```
DROP INDEX nombre_indice;
```