BASES DE DATOS

CREACIÓN DE TABLAS Y VISTAS

1 Introducción

El **SQL** (Structured Query Language), lenguaje de consulta estructurado, es un lenguaje surgido de un proyecto de investigación de IBM para el acceso a bases de datos relacionales. Actualmente se ha convertido en un estándar de lenguaje de bases de datos, y la mayoría de los sistemas de bases de datos lo soportan.

A partir del estándar cada sistema ha desarrollado su propio SQL que puede tener variaciones no significativas de un sistema a otro.

SQL además realiza funciones de definición, control y gestión de la base de datos. Las sentencias SQL se clasifican según su finalidad originando tres lenguajes:

- **DDL** (Data Description Language), lenguaje de definición de datos, que incluye órdenes para crear, modificar o borrar las tablas en las que se almacenan los datos y definir las relaciones entre estas. Es el que más variantes puede presentar entre sistemas.
- **DCL** (Data Control Language), lenguaje de control de datos, contiene los elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.
- **DML** (Data Manipulation Language), lenguaje de manipulación de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados. Es el más utilizado.

El lenguaje SQL consta de unas treinta sentencias. Cada sentencia solicita una acción específica por parte del SGBD como: creación de una nueva tabla, recuperación de datos o inserción de nuevos datos en la base de datos.

En la siguiente tabla podemos ver un resumen de estas sentencias.

Sentencias	Descripción
De manipulación de datos o LMD	
SELECT	Recupera datos de la BD
INSERT	Añade nuevas filas de datos de la BD
DELETE	Suprime filas de la BD
UPDATE	Modifica datos existentes en la BD
De definición de datos o LDD	
CREATE TABLE	Añade una base de datos a la BD
DROP TABLE	Suprime una tabla de la BD
ALTER TABLE	Modifica la estructura de una tabla existente
CREATE VIEW	Añade una nueva vista a la BD
DROP VIEW	Suprime una vista de la BD
CREATE INDEX	Construye un índice para una columna
DROP INDEX	Suprime el índice para una columna
CREATE SYNONYM	Define un alias para un nombre de tabla
DROP SYNONYM	Suprime un alias para un nombre de tabla
COMMENT	Define comentarios para una tabla
LABEL	Define el título de una columna
De control de acceso o LCD	
GRANT	Concede privilegios de acceso a usuarios
REVOKE	Suprime privilegios de acceso a usuarios
De control de transacciones o LCD	
COMMIT	Finaliza la transacción actual
ROLLBACK	Aborta la transacción actual
De SQL programático	
DECLARE	Define un cursor para una consulta
EXPLAIN	Describe el plan de acceso a datos para una consulta
OPEN	Abre un cursor para recuperar resultados de consulta
FETCH	Recupera una fila de resultados de consulta
CLOSE	Cierra un cursor
PREPARE	Prepara una sentencia SQL para ejecución dinámica
EXECUTE	Ejecuta dinámicamente una sentencia SQL
DESCRIBE	Describe una consulta preparada

1.1 Características del lenguaje SQL

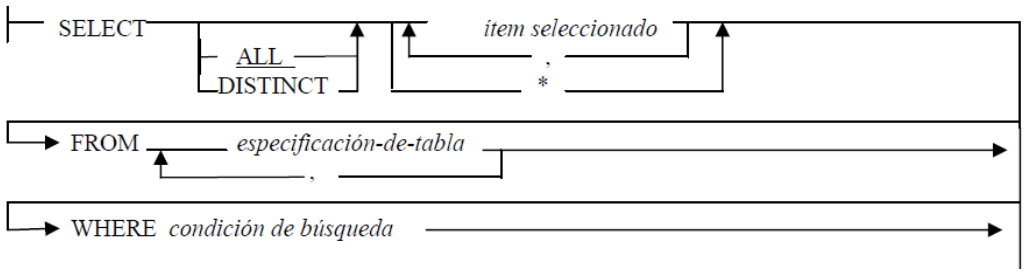
Es una aproximación al lenguaje natural, pues una sentencia SQL es una "frase" con las indicaciones de qué queremos y dónde está.

DELETE FROM *InfVentas* WHERE *VENTAS*<1000

Todas las sentencias SQL tienen la misma estructura: comienzan con un verbo que será una palabra reservada y que indica la acción a realizar CREATE, INSERT, DELETE y COMMIT. Después una serie de cláusulas que completan la sentencia. Estas cláusulas pueden ser obligatorias u opcionales.

Una cláusula puede especificar los datos sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la sentencia se supone que hace. Todas las cláusulas comienzan también con una palabra clave tal como WHERE, FROM, INTO y HAVING. La estructura y contenido específicos varían de una cláusula a otra. Muchas cláusulas contienen nombres de tablas o columnas; algunas pueden contener palabras clave adicionales, constantes o expresiones.

La sintaxis de las sentencias SQL se expresa gráficamente con diagramas como el siguiente:



Las palabras que aparecen en mayúsculas son palabras reservadas y se tienen que poner tal cual. No se pueden utilizar para otro fin. En el ejemplo aparecen las palabras reservadas SELECT, ALL, DISTINCT, FROM, WHERE.

Las palabras en minúsculas son variables que sustituiremos por un dato concreto. En el diagrama tenemos ítem-seleccionado, especificación-tabla y condición-de-búsqueda.

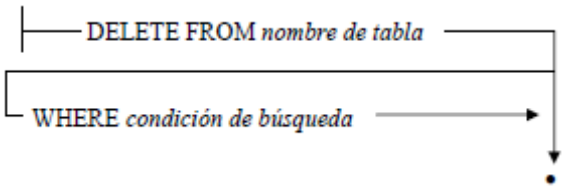
Una sentencia válida se construye siguiendo la línea a través del diagrama hasta el punto que marca el final. Las líneas se siguen de izquierda a derecha y de arriba abajo. Cuando se quiere alterar el orden normal se indica con una flecha.

Veamos el ejemplo de la figura: Empezamos por la palabra SELECT, después podemos poner ALL o bien DISTINCT o nada, a continuación un nombre de columna, o varios separados por comas, a continuación la palabra FROM y un nombre de tabla, y por último de forma opcional podemos incluir la cláusula WHERE con una condición-de-búsqueda.

```
SELECT ALL col1, col2, col3 FROM mitabla
SELECT col1, col2, col3 FROM mitabla
SELECT DISTINCT col1 FROM mitabla
SELECT col1, col2 FROM mitabla WHERE col2 = 0
```

Cuando en el esquema una palabra opcional está subrayada, esto indica que ese es el valor por defecto (el valor que se asume si no se pone nada). En el ejemplo anterior las dos primeras sentencias son equivalentes (en el diagrama ALL aparece subrayada).

Otros ejemplos:



DELETE y FROM son palabras clave o reservadas. Los ítems variables en el gráfico anterior son el nombre de una tabla y la condición de búsqueda.

Las cláusulas y palabras clave opcionales, tales como la cláusula WHERE del gráfico anterior se indican mediante caminos alternativos dentro del diagrama sintáctico.

La sintaxis de la sentencia DELETE de SQL en notación **BNF** es:

DELETE FROM <nombre tabla> [WHERE <condición>]

DELETE, FROM y WHERE son palabras clave o reservadas por aparecer en mayúsculas.

Los ítems variables o construcciones sintácticas, que el usuario deberá sustituir por un dato concreto, son el nombre de una tabla y la condición de búsqueda. La cláusula y palabra clave WHERE es opcional, por aparecer entre corchetes.

Notación BNF (Backus-Naur Form)

En este metalenguaje se utilizan ***metasímbolos*** (símbolos que se utilizan para explicar algo otros símbolos, los de un lenguaje).

Por ejemplo: <identificador> ::= <letra> { <letra> | <digito> }

	Significado
MAYÚSCULAS o cursiva (depende de la notación elegida)	Palabras reservadas
Minúsculas entre < >	Construcciones sintácticas (algo que tiene ya definida su explicación sintáctica)
[ ]	Opcionalidad (se puede poner o no, pero no se repite
{ }	Posible repetición (cero o más veces)
|	Alternativa (o una opción o la otra)
::=	"Se construye como…"

Más ejemplos:
<carácter> ::= A|B|C|….|Z|a|b|c|…|z
<cadena de caracteres> ::= '<carácter>{<carácter>}'|"<carácter>{<carácter>}"
<signo> ::= + | -
<identificador de archivo> ::= <identificador>
<identificador de variable> ::= <identificador>

1.2 Identificadores en SQL

Los objetos de una base de datos basada en SQL se identifican asignándoles nombres **únicos**. Los nombres se utilizan en las sentencias SQL para identificar el objeto de la base de datos sobre la que la sentencia debe actuar.

El estándar SQL especifica nombres de tabla, nombres de columna y nombres de usuario. También especifica que los nombres deben contener de **1 a 18 caracteres**, **comenzar con una letra**, y que **no pueden contener espacios o caracteres de puntuación especiales**. En la práctica los nombres contemplados por los distintos productos SGBD basados en SQL varían significativamente con identificadores de hasta 128 letras, dígitos o el símbolo _ o restricciones en los nombres de usuario a ocho caracteres. Los diferentes productos también se diferencian en los caracteres especiales que permiten en los nombres de tablas. Para conseguir portabilidad es mejor mantener los nombres relativamente breves y evitar el uso de caracteres especiales.

Nombres de tabla

Cuando un usuario de una base de datos especifica un nombre de tabla en una sentencia SQL, se refiere a una de las tablas de su propiedad. Si tiene el permiso adecuado, también puede hacer referencia a tablas de otros usuarios. En este caso se debe especificar el nombre del propietario de la tabla junto con el nombre de la tabla, separados por un punto (.). Por ejemplo, la tabla EMPLEADOS, propiedad del usuario de nombre ADMIN2, debe de ser nombrada ADMIN2.EMPLEADOS.

Nombre completo o totalmente cualificado de un objeto de la base de datos:

[[[servidor.] [base de datos] .] [nombre propietario] .] nombre objeto

Nombres de columna

Cuando se especifica un nombre de columna en una sentencia SQL, SQL puede determinar normalmente a qué columna se refiere a partir del contexto. Sin embargo, si la sentencia afecta a dos columnas con el mismo nombre correspondientes a dos tablas diferentes, debe utilizarse un nombre de columna cualificado para identificar sin ambigüedad la columna designada. Un nombre de columna cualificado especifica el nombre de la tabla y el nombre de la columna, separados por un punto (.). Por ejemplo, la columna NOMBRE de la tabla EMPLEADOS tiene el nombre de columna cualificado EMPLEADOS.NOMBRE: Esto la distinguiría de, por ejemplo, CLIENTES.NOMBRE.

Si la columna procede de una tabla propiedad de otro usuario, se utiliza un nombre de tabla cualificado en el nombre de columna cualificado. Por ejemplo ADMIN2.EMPLEADOS.NOMBRE.

SQL permite nombres de columnas que contengan blancos y otros caracteres especiales. Cuando estos caracteres aparecen como nombres en una sentencia SQL, deben ir entre corchetes. Por ejemplo, si la columna NOMBRE de la tabla EMPLEADOS fuera en realidad “NOMBRE COMPLETO” en una base de datos SQL, esta sentencia SELECT sería válida.:

```
SELECT [NOMBRE COMPLETO], PUESTO, SUELDO
   FROM EMPLEADOS
   WHERE [NOMBRE COMPLETO] = ‘Rodríguez, Juan Manuel’
```

1.3 Comentarios en SQL

Para poder añadir comentarios en SQL tenemos varias formas. La primera es la definida en el estándar ANSI/SQL mediante dos guiones al principio de una línea.

```
-- Esto es un comentario
```

Esta forma de poner comentarios nos sirve para anular partes de sentencias Por ejemplo, si tenemos la sentencia:

```
SELECT *
FROM libros
WHERE idautor = 1
```

Podemos anular el filtro del autor de una forma sencilla comentándolo mediante los guiones:

```
SELECT *
FROM libros
--WHERE idautor = 1
```

En este caso si los comentarios ocupan varias líneas, los dos guiones deben aparecer al principio de cada línea de comentario.

Otra forma de añadir comentarios, que en este caso pueden ser de varias líneas, es mediante la estructura /*comentario*/.

```
/*
 * Comentario en varias líneas.
 * Compatible con el estándar SQL99
 */
```

Esta forma fue añadida en la revisión SQL 99 del lenguaje.

Existen formas ya adaptadas a diferentes gestores de bases de datos. Es por ello que podemos encontrarnos con comentarios en sentencias SQL de las siguientes formas:

```
{Comentario con llaves}
# Esto es un comentario
REM Esto es un comentario
```

Aunque lo recomendable es definir los comentarios SQL mediante las dos formas estándar explicadas anteriormente.

2 Herramientas para la creación, modificación y borrado de elementos de bases de datos

2.1 Tablas

Tabla en las bases de datos, se refiere al tipo de modelado de datos, donde se guardan los datos recogidos por un programa. Una tabla es utilizada para organizar y presentar información. Las tablas se componen de filas y columnas de celdas que se pueden rellenar con textos y gráficos.

Las tablas se componen de dos estructuras:

Registro: es cada una de las filas en que se divide la tabla. Cada registro contiene datos de los mismos tipos que los demás registros. Ejemplo: en una tabla de nombres y direcciones, cada fila contendrá un nombre y una dirección.

Campo: es cada una de las columnas que forman la tabla. Contienen datos de tipo diferente a los de otros campos. En el ejemplo anterior, un campo contendrá un tipo de datos único, como una dirección, o un número de teléfono, un nombre, etc.

A los campos se les puede asignar, además, propiedades especiales que afectan a los registros insertados. El campo puede ser definido como índice o autoincrementable, lo cual permite que los datos de ese campo cambien solos o sean el principal a la hora de ordenar los datos contenidos.

Cada tabla creada debe tener un nombre único en la Base de Datos, haciéndola accesible mediante su nombre o su seudónimo (Alias) (dependiendo del tipo de base de datos elegida). Las tablas son los objetos principales de bases de datos que se utilizan para guardar datos.

2.2 Índices

Un índice es una estructura de datos definida sobre una columna de tabla (o varias) y que permite localizar de forma rápida las filas de la tabla en base a su contenido en la columna indexada además de permitir recuperar las filas de la tabla ordenadas por esa misma columna.

Funciona de forma parecida al índice de un libro donde tenemos el título del capítulo y la página donde empieza dicho capítulo, en un índice definido sobre una

determinada columna tenemos el contenido de la columna y la posición de la fila que contiene dicho valor dentro de la tabla.

La definición de los índices de la base de datos es tarea del administrador de la base de datos, los administradores más experimentados pueden diseñar un buen conjunto de índices, pero esta tarea es muy compleja, consume mucho tiempo y está sujeta a errores, incluso con cargas de trabajo y bases de datos con un grado de complejidad no excesivo.

Índice simple y compuesto

Un índice simple está definido sobre una sola columna de la tabla mientras que un índice compuesto está formado por varias columnas de la misma tabla (tabla sobre la cual está definido el índice.

Cuando se define un índice sobre una columna, los registros que se recuperen utilizando el índice aparecerán ordenados por el campo indexado. Si se define un índice compuesto por las columnas APELLIDOS y NOMBRE, las filas que se recuperen utilizando dicho índice aparecerán ordenadas por los valores de APELLIDOS y todas las filas que tengan el mismo valor de APELLIDOS se ordenarán a su vez por los valores contenidos en NOMBRE.

Por ejemplo si definimos un índice compuesto basado en las columnas (provincia, localidad), las filas que se recuperen utilizando este índice aparecerán ordenadas por provincia y dentro de la misma provincia por localidad.

Índice agrupado y no agrupado

El término índice agrupado no se debe confundir con índice compuesto, el significado es totalmente diferente. Un índice agrupado (CLUSTERED) es un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla. El nivel inferior, u hoja, de un índice agrupado contiene las filas de datos en sí de la tabla. Una tabla o vista permite un solo índice agrupado al mismo tiempo.

Los índices no agrupados existentes en las tablas se vuelven a generar al crear un índice agrupado, por lo que es conveniente crear el índice agrupado antes de crear los índices no agrupados.

Un índice no agrupado especifica la ordenación lógica de la tabla. Con un índice no agrupado, el orden físico de las filas de datos es independiente del orden indizado.

Índice único

Índice único es aquel en el que no se permite que dos filas tengan el mismo valor en la columna de clave del índice. Es decir que no permite valores duplicados.

Ventajas

La utilización de índices puede mejorar el rendimiento de las consultas, ya que los datos necesarios para satisfacer las necesidades de la consulta existen en el propio índice. Es decir, sólo se necesitan las páginas de índice y no las páginas de datos de la tabla o el índice agrupado para recuperar los datos solicitados; por tanto, se reduce la E/S global en el disco. Por ejemplo, una consulta de las columnas a y b de una tabla que dispone de un índice compuesto creado en las columnas a, b y c puede recuperar los datos especificados del propio índice.

Inconvenientes

Las tablas utilizadas para almacenar los índices ocupan espacio. Los índices consumen recursos ya que cada vez que se realiza una operación de actualización, inserción o borrado en la tabla indexada, se tienen que actualizar todas las tablas de índice definidas sobre ella (en la actualización sólo es necesaria la actualización

de los índices definidos sobre las columnas que se actualizan). Por estos motivos no es buena idea definir índices indiscriminadamente.

Consideraciones a tener en cuenta

- Hay que evitar crear demasiados índices en tablas que se actualizan con mucha frecuencia y procurar definirlos con el menor número de columnas posible.
- Es conveniente utilizar un número mayor de índices para mejorar el rendimiento de consultas en tablas con pocas necesidades de actualización, pero con grandes volúmenes de datos. Un gran número de índices contribuye a mejorar el rendimiento de las consultas que no modifican datos, como las instrucciones SELECT, ya que el optimizador de consultas dispone de más índices entre los que elegir para determinar el método de acceso más rápido.
- La indización de tablas pequeñas puede no ser una solución óptima, porque puede provocar que el optimizador de consultas tarde más tiempo en realizar la búsqueda de los datos a través del índice que en realizar un simple recorrido de la tabla. De este modo, es posible que los índices de tablas pequeñas no se utilicen nunca; sin embargo, sigue siendo necesario su mantenimiento a medida que cambian los datos de la tabla.
- Se recomienda utilizar una longitud corta en la clave de los índices agrupados. Los índices agrupados también mejoran si se crean en columnas únicas o que no admitan valores NULL.
- Un índice único en lugar de un índice no único con la misma combinación de columnas proporciona información adicional al optimizador de consultas y, por tanto, resulta más útil.
- Hay que tener en cuenta el orden de las columnas si el índice va a contener varias columnas. La columna que se utiliza en la cláusula WHERE en una condición de búsqueda igual a (=), mayor que (>), menor que (<) o BETWEEN, o que participa en una combinación, debe situarse en primer lugar. Las demás columnas deben ordenarse basándose en su nivel de diferenciación, es decir, de más distintas a menos distintas.

Para crear índices se utiliza la orden **CREATE INDEX**, aunque también se pueden crear índices con las restricciones de seguridad **UNIQUE** y **PRIMARY KEY** de la sentencia CREATE TABLE, aunque estos últimos tienen las siguientes limitaciones:

- No permiten crear índices que no sean únicos.
- Para poder omitir posteriormente estos índices debemos utilizar ALTER TABLE.
- No se pueden utilizar las opciones de CREATE INDEX.

```
ALTER TABLE AUTORES
ADD CONSTRAINT PK_IdAutor PRIMARY KEY CLUSTERED (IdAutor)
```

La sentencia CREATE INDEX tiene muchas opciones:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH < index_option > [ ,...n] ]
```

Pero el formato básico es el siguiente:

CREATE INDEX nombreIndice ON nombreTabla (nombreColumna)

CREATE INDEX idx_IdAutor ON AUTORES (idAutor)

La sentencia CREATE INDEX es más compleja y requiere consultar sus opciones en el manual del gestor que estemos utilizando. Veremos ahora sus opciones principales.

Se pueden definir índices Compuestos.

- Un índice compuesto está formado por varias columnas.
- Las columnas no tienen por qué tener el mismo orden que en la tabla y se pueden combinar hasta 16 columnas, con un máximo de 256 bytes.
- Las columnas tienen que ser de la misma tabla.
- El orden de cada columna puede ser Ascendente (por defecto) o Descendente. Se representa con ASC o DESC respectivamente.

CREATE INDEX idx_Amigos ON AMIGOS (nombre ASC, apellido DESC)

que es lo mismo que:

CREATE INDEX idx_Amigos ON AMIGOS (nombre, apellido DESC)

Se pueden definir índices Únicos

- Un índice único es aquel índice que no permite que 2 filas tengan el mismo valor de índice.
- Para crear un índice único se debe especificar UNIQUE.

CREATE UNIQUE INDEX idx_DNI ON PERSONAS (DNI)

Se pueden crear índices compuestos únicos.

CREATE UNIQUE INDEX idx_Amigos ON AMIGOS (nombre, apellido)

Definir índices Agrupados/No-Agrupados

- Con un índice agrupado (CLUSTERED), se ordenan las filas de forma continuada, de forma que el orden físico de cada tupla o registro coincida con el orden lógico especificado.
- Si el índice es No-Agrupado (NONCLUSTERED), el orden físico es independiente del orden lógico, siendo el gestor el encargado de entregarnos posteriormente los registros según el orden lógico.

CREATE CLUSTERED INDEX idx_Titulo ON LIBROS (idTitulo)

CREATE NONCLUSTERED INDEX idx_CPostal ON CLIENTES (codPostal)

Se puede modificar un índice (ALTER INDEX)

```
ALTER INDEX PK_Employee_EmployeeID
ON HumanResources.Employee
REBUILD;
```

Se puede borrar un índice (DROP INDEX)

```
DROP INDEX tCLIENTES.UIX_CLIENTES_NIF;
```

Se puede desactivar/Activar un índice (DISABLE/ENABLE)

```
ALTER INDEX IX_Employee_ManagerID
ON HumanResources.Employee
DISABLE ;
```

2.3 Consultas

En bases de datos, una consulta es el método para acceder a los datos en las bases de datos. Con las consultas se puede modificar, borrar, mostrar y agregar datos en una base de datos. Para esto se utiliza un lenguaje de consultas. El lenguaje de consultas a base de datos más utilizado es el **SQL**.

Las consultas nos permitirán, entre otras cosas:

1. Recuperar datos de una o más tablas con los criterios especificados y después mostrar los datos en el orden que se desee.
2. Ver todos o algunos registros, todos o algunos campos, de una o varias tablas relacionadas.
3. Actualizar registros en una tabla.
4. Agrupar registros y calcular sumas, cuentas, promedios y otros tipos de totales.
5. Reunir datos de varias tablas y ordenarlos de una forma concreta.
6. Eliminar un registro o un grupo de registros de una o más tablas.
7. Realizar cambios globales en un grupo de registros de una o más tablas.
8. Anexar un grupo de registros de una a otra tabla.
9. Crear tablas que pueden exportarse a otras bases de datos.

2.4 Vistas

Una vista es una consulta que se presenta como una tabla (virtual) a partir de un conjunto de tablas en una base de datos relacional. Las tablas que originan la vista se denominan tablas base o *tablas subyacentes*.

Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo se almacena de ellas la definición, no los datos. Los datos que se recuperan mediante una consulta a una vista se presentarán igual que los de una tabla. De hecho, si no se sabe que se está trabajando con una vista, nada hace suponer que es así. Al igual que sucede con una tabla, se pueden insertar, actualizar, borrar y seleccionar datos en una vista. Aunque siempre es posible seleccionar datos de una vista, en algunas condiciones existen restricciones para realizar el resto de las operaciones sobre vistas.

Una vista se especifica a través de una expresión de consulta (una sentencia SELECT) que la calcula y que puede realizarse sobre una o más tablas. Sobre un conjunto de tablas relacionales se puede trabajar con un número cualquiera de vistas.

La mayoría de los SGBD soportan la creación y manipulación de vistas. Las vistas se crean cuando se necesitan hacer varias sentencias para devolver una tabla final.

Una vista es una tabla virtual derivada de las tablas reales de una base de datos. Las vistas no se almacenan en la base de datos, resultado de la cual se produce una tabla cuyos datos proceden de la base de datos o de otras vistas. Eso asegura que los datos sean coherentes al utilizar los datos almacenados en las tablas. Si los datos de las relaciones cambian, los de la vista que utiliza esos datos también cambian. Por todo ello, las vistas gastan muy poco espacio de disco.

Usos de las vistas

- Realizar consultas complejas más fácilmente: Las vistas permiten dividir la consulta en varias partes.
- Proporcionar tablas con datos específicos: Las vistas permiten ser utilizadas como tablas que resumen todos los datos, así como también permiten ocultar ciertos datos. Cuando ese se requiere un detalle que no corresponde precisamente a las relaciones.
- Modularidad de acceso a base de datos: las vistas se pueden pensar en forma de módulos que nos da acceso a partes de la base de datos. Cuando ese detalle que se requiere no corresponde precisamente a las relaciones.

Las aplicaciones reales tienden a usar un muchas vistas, por lo que cuanto más grande es la aplicación, más necesario es que haya modularidad, para facilitar determinadas consultas o para ocultar los datos. Las vistas entonces son el mecanismo para alcanzar dichos objetivos.

Creación de una vista

CREATE VIEW: Define una tabla lógica a partir de una o más tablas físicas o de otras vistas.

DROP VIEW: Elimina una definición de vista (y cualquier vista definida a partir de ella).

CREATE VIEW NombreVista(A1,A2,...,An) As <QuerySQLstandar>

NombreVista es el nombre que se le asigna a la vista, A1, A2,..., An son los nuevos nombres de los atributos que tendrá la vista.

Ejemplo 1: Se utiliza una base de datos con las siguientes relaciones:

Especies(nombreEspecie------,nombreComún,familia)

Esta tabla almacena los datos que caracterizan las especies animales. Almacena el nombre científico en nombreEspecie, el nombre común con el que se le conoce es guardado en nombreComún y en familia la familia a la que pertenece la especie.

Zoo(zID---,zooNombre,tamaño,presupuesto)

La relación Zoo almacena los datos de los zoológicos. Un zID que es la primary key, el nombre en zooNombre, tamaño es el tamaño en hectáreas y presupuesto en unidades monetarias.

Animal(zID, nombreEspecie, aNombre----------,país)

La tabla animal guarda los datos de los animales que habitan cada zoológico. El atributo zID es clave foránea a Zoo, se refiere al zoológico en el que se encuentra un animal, nombreEspecie es clave foránea a la especie a la que pertenece, país es el país de procedencia.

Se crea una vista:

```
CREATE VIEW View1 AS
    SELECT zID, nombreEspecie
    FROM Animal
    WHERE aNombre = 'Tony' and país = 'China';
```

Como ya se mencionó para crear una vista se usan las palabras clave CREATE VIEW especificando el nombre de la vista View1 . Luego se declara una consulta en SQL estándar. Dicha consulta selecciona zID y nombreEspecie de los animales que se llamen 'Tony' y procedan de 'China' .

2.5 Sinónimos

Un sinónimo es un nombre alternativo (alias) que identifica una tabla en la base de datos. Con un sinónimo se pretende normalmente simplificar el nombre original de la tabla, aunque también se suelen utilizar para evitar tener que escribir el nombre del propietario de la tabla.

No todas las bases de datos soportan los sinónimos.

Para crear un sinónimo hay que utilizar la sentencia **CREATE SYNONYM** especificando el nombre que deseamos utilizar como sinónimo y la tabla para la que estamos creando el sinónimo.

```
CREATE SYNONYM
    FOR ;
```

Por ejemplo: El siguiente ejemplo crea el sinónimo Coches para la tabla tCoches.

```
CREATE SYNONYM Coches
    FOR tCoches;
```

Para eliminar el sinónimo creado debemos emplear la sentencia **DROP SYNONYM**.

```
DROP SYNONYM Coches;
```

Es una forma de facilitar a los usuarios acceder a objetos de base de datos propiedad de otros usuarios y también de ocultar la identidad del objeto subyacente y hacer más difícil para un programa malicioso o usuario lo descubra. Porque un sinónimo es simplemente un nombre alternativo para un objeto, no se requiere ningún almacenamiento aparte de su definición. Cuando una aplicación utiliza un sinónimo, el DBMS reenvía la solicitud al objeto base subyacente de los sinónimos. Cifrando los programas para utilizar Sinónimos en lugar de nombres de objeto de base de datos, salvamos de cualquier cambio el nombre, propiedad u objeto.

Hay dos usos principales de Sinónimos:

Invisibilidad del objeto: Sinónimos pueden crearse para mantener el objeto original ocultado para el usuario.

Invisibilidad de ubicación: Sinónimos pueden crearse como alias para las tablas y otros objetos que no son parte de la base de datos local.

3 Tipos de datos

Los tipos de datos que manejan los SGBD mediante SQL nos permiten definir el formato de la información que usemos en nuestras columnas, variables y expresiones. Hay que aclarar que dependiendo del SGBD que usemos los tipos pueden cambiar, así que veremos los más utilizados.

Al igual que otros lenguajes de programación, para SQL podemos encontrar números enteros, flotantes, cadenas, booleanos y demás.

Tipos de datos enteros

Existen varios tipos de datos para manejar números enteros dentro de SQL. Todas esas opciones se diferencian en el tamaño de memoria asignado para un rango de valores enteros.

Tipo de dato	mínimo	máximo
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,648
Smallint	-32,768	32,768
Tiyint	0	255
Bit	0	1
Decimal	-10^38+1	10^38+1
Numeric	-10^38+1	10^38+1
Money	-922,337,203,685,477.5808	922,337,203,685,477.5808
Smallmoney	-214,748.3648	214,748.3648

NUMERIC (precisión, escala) o DECIMAL (precisión, escala) Ofrecen un alto grado de precisión, además de un amplio rango de valores. En ambos hay que especificar dos datos: la precisión, que es el número total de dígitos guardados; y la escala, que es el número máximo de dígitos que están a la derecha del punto decimal. Tanto en un tipo como en el otro, la escala<=precisión. Si no se especifica precisión y escala, son igual que INTEGER.

El tener un tipo monetario específico permite al SGBD dar formato adecuado a los importes monetarios cuando son visualizados. MONEY ocupa 8 bytes de almacenamiento y SMALLMONEY 4 bytes. Para ambos, no es necesario incluir los datos de moneda (€) entre comillas simples.

Dependiendo de nuestras necesidades y criterios para guardar información elegiremos un tipo de dato u otro.

Por ejemplo, si deseamos establecer una columna dentro de la tabla PELICULAS_ALQUILADAS, cuya función sea informar de dos estados posibles de la película: Alquilada o No Alquilada, podría ser buena idea usar el tipo numérico BIT. El valor 1 representaría el estado Alquilada y el 0 No Alquilada.

Tipos de datos flotantes

Para representar número de coma flotante usaremos los siguientes tipos.

Tipo de dato	mínimo	máximo
Float	-1.79E+308	1.79E+308
Real	-3.40E+38	3.40E+38

Números de punto flotante: Por qué son necesarios los números de punto flotante.
http://puntoflotante.org/formats/fp/

Tipos de datos fecha

Es seguro que en algún momento necesitemos guardar registros que contengan información sobre fechas de nacimiento, tiempo de llegada, etc. Para este tipo de ocasiones existen los datos fecha en SQL.

Tipo de dato	Formato	Rango
Date	AAAA-MM-DD	1000-01-01 **a** 9999-12-31
Datetime	AAAA-MM-DD HH:MM:SS	1000-01-01 00:00:00 **a** 9999-12-31 23:59:59
Timestam	AAAA-MM-DD HH:MM:SS	1970-01-01 00:00:00 **a** 2037-12-31 23:59:59
Time	HH:MM:SS	-838:59:59 **a** 838:59:59

Cadenas de caracteres

Guardar nombres, apellidos, direcciones y otros tipos de datos denominativos requiere que usemos de cadenas de caracteres para gestionar estos atributos en SQL. Veamos:

Tipo de dato	Rango
Char	Longitud máxima de 8000 caracteres. Su longitud esta predeterminada
Varchar	Longitud máxima de 8000 caracteres. A diferencia de **char**, su longitud es variable
Text	Longitud máxima de 2,147,483,647 caracteres
Nchar	Longitud máxima de 4000 caracteres. A diferencia de **char**, **Nchar** acepta Caracteres Unicode
NVarchar	Longitud máxima de 4000 caracteres. A diferencia de **Varchar**, **NVarchar** acepta Caracteres Unicode.
Ntext	Longitud máxima de 1,073,741,823 caracteres. A diferencia de **Text, Ntext** acepta caracteres Unicode.

De longitud fija, CHAR (número) o CHARACTER(número) siempre almacenan el número de caracteres indicado en "número", aún cuando sea necesario rellenar los espacios sobrantes con caracteres en blanco porque la longitud de la cadena de caracteres sea inferior a "número".

De longitud variable: VARYING CHARACTER o VARCHAR (número) almacenan los caracteres que se hayan introducido, hasta un máximo que viene dado por "número".

TEXT (equivalente de MEMO) permite almacenar cadenas de texto extensas, es decir, cadenas de caracteres ASCII de longitud variable, de más de 8 Kbytes. El SGBD restringe generalmente el uso de estas columnas en consultas y búsquedas interactivas.

Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas, además de textos clásicos de lenguas muertas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad.

Unicode especifica un nombre e identificador numérico único para cada carácter o símbolo, el ***code point*** ('punto de código'), además de otras informaciones necesarias para su uso correcto: direccionalidad, mayúsculas y otros atributos. Unicode trata los caracteres alfabéticos, ideográficos y símbolos de forma equivalente, lo que significa que se pueden mezclar en un mismo texto sin la introducción de marcas o caracteres de control.

El establecimiento de Unicode ha sido un ambicioso proyecto para reemplazar los esquemas de codificación de caracteres existentes, muchos de los cuales están muy limitados en tamaño y son incompatibles con entornos plurilingües. Unicode se ha vuelto el más extenso y completo esquema de codificación de caracteres, siendo el dominante en la internacionalización y adaptación local del software informático. El estándar ha sido implementado en un número considerable de tecnologías recientes, que incluyen XML, Java y sistemas operativos modernos.

La descripción completa del estándar y las tablas de caracteres están disponibles en la página web oficial de Unicode http://unicode-table.com/es/

Otros

Flujo de bits no estructurados: SQL Server y otros varios productos permiten almacenar y recuperar secuencias de bits de longitud variable sin estructurar. Son datos tipo **IMAGE**. Las columnas que contienen estos datos se utilizan para almacenar imágenes de vídeo comprimidas y otros tipos de datos sin estructurar. Por ejemplo, imágenes TIFF.

Constantes simbólicas

Además de las constantes suministradas por el usuario, el lenguaje SQL incluye constantes simbólicas especiales que devuelven valores de datos mantenidos por el propio SGBD.

USER y **CURRENT_USER** contienen el nombre de usuario bajo el cual se está accediendo actualmente a la base de datos.

CURRENT_TIMESTAMP contienen la fecha y hora actuales. En SQL Server se puede utilizar current_timestamp o getdate(). Un jemplo de su uso puede ser:

```
SELECT CURRENT_TIMESTAMP
```

Generalmente, una constante simbólica puede aparecer en una sentencia SQL en cualquier lugar en el que pudiera aparecer una constante ordinaria del mismo tipo de dato.

Algunos productos SQL, incluyendo SQL Server, proporcionan acceso a valores del sistema mediante funciones internas además de poder hacerlo con constantes simbólicas.

EXPRESIONES

Las expresiones se utilizan para hacer cálculos con valores que se recuperan de una base de datos y para calcular valores utilizados en la búsqueda en la base de datos. Por ejemplo, la siguiente consulta calcula las ventas de cada oficina como porcentaje de su objetivo:

```
SELECT CIUDAD, OBJETIVO, VENTAS, (VENTAS/OBJETIVO) * 100
   FROM OFICINAS
```

y esta consulta lista las ciudades de las oficinas cuyas ventas son superiores a 50.000 por encima del objetivo:

```
SELECT CIUDAD
   FROM OFICINAS
   WHERE VENTAS > Objetivo + 50000.00
```

El estándar SQL ANSI/ISO especifica cuatro operaciones aritméticas que pueden ser utilizadas en expresiones: suma (X+Y), resta (X-Y), multiplicación (X*Y) y división (X/Y). También se pueden utilizar paréntesis para formar expresiones más complicadas, como la siguiente:

```
(VENTAS * 1.05) – (OBJETIVO * 0.95)
```

Estrictamente hablando, los paréntesis no son necesarios en esta consulta, ya que el estándar especifica que la multiplicación y la división tienen una precedencia superior a la suma y la resta. Sin embargo, se recomienda utilizar siempre los paréntesis para lograr que las expresiones no sean ambiguas, ya que diferentes dialectos de SQL pueden utilizar reglas diferentes. Además los paréntesis incrementan la legibilidad de la sentencia y hacen que las sentencias de SQL programado sean más fáciles de mantener.

El estándar también especifica conversión automática de tipos de datos desde números enteros a decimales, y desde números decimales a números en coma flotante, según se precise.

Muchas implementaciones SQL incluyen otros operadores y permiten operaciones sobre datos de caracteres y de fechas como el operador concatenación de cadenas, escrito con el símbolo “+”. Por ejemplo, si dos columnas llamadas NOMBRE y APELLIDO contienen los valores “Jorge” y “Martínez”, entonces la

expresión: (NOMBRE +" "+ APELLIDO) da como resultado la cadena "Jorge Martínez".

FUNCIONES INTERNAS

Son funciones que ya están definidas en SQL. Algunas son:

Función	Valor devuelto
CAST (valor AS tipo_dato)	Convierte ***valor*** al tipo de dato especificado (por ejemplo, una fecha convertida a una cadena de caracteres).
CURRENT_TIMESTAMP	Fecha y hora actuales. (en SQL Server hace lo mismo que la función del sistema ***Getdate())***
LOWER(cadena)	Convierte una cadena de tipo VARCHAR a minúsculas
UPPER (cadena)	Convierte una cadena de tipo VARCHAR a mayúsculas
SUBSTRING (fuente, n, lon)	Extrae, de la cadena fuente, una subcadena comenzando en el carácter n-ésimo y con una longitud lon
LEFT(cadena, lon)	Devuelve, de la cadena, una subcadena comenzando por la izquierda y con una longitud lon
LEN(cadena)	Devuelve la longitud de la cadena
LTRIM(cadena)	Quita los blancos de la izquierda en la cadena
RTRIM(cadena)	Quita los blancos de la derecha en la cadena
RIGHT(cadena, nº elem)	Devuelve los elementos de la cadena que están a la derecha. Por ejemplo RIGHT(CASA,2) = 'SA'
ASCII(cadena)	Devuelve el código ASCII correspondiente al carácter más a la izquierda de la cadena
ISNULL(expresión, valor de sustitución)	Si la expresión es NULL la función devuelve el valor de sustitución. Sin embargo, si la expresión NO es NULL devuelve la propia expresión.

Es posible que en alguna ocasión estemos interesados en localizar un valor que nos aporte información interesante y que se obtenga a partir de una o varias columnas. Podría ser, por ejemplo, que necesitemos conocer:

- cuál es el artículo del que menos unidades tenemos para realizar el pedido con más urgencia

o

- cuál es la suma de unidades de todos los artículos del almacén.

Para resolver estas cuestiones SQL dispone de lo que se denominan funciones integradas de conjuntos o funciones resumen. Las más habituales son:

_ **MAX** () para obtener el máximo

_ **MIN** () para obtener el mínimo

_ **AVG** () para obtener la media aritmética
_ **SUM** () para obtener la suma
_ **COUNT** () para obtener cuántos.

Por ejemplo, para hallar cuántos artículos distintos hay en el almacén haríamos:

```
SELECT COUNT(ID_ARTICULO)
        FROM ARTICULOS
```

4 Creación de tablas. Claves primarias, alternativas y ajenas

- La instrucción que permite crear las tablas de una base de datos es **CREATE TABLE**.
- La sentencia define una nueva tabla en la base de datos y la prepara para aceptar datos.
- Las diferentes cláusulas de la sentencia especifican los elementos de la definición de la tabla.
- Cuando se ejecuta una sentencia CREATE TABLE, uno se convierte en el propietario de la tabla recién creada, a la cual se le da el nombre especificado en la sentencia.
- El nombre de la tabla debe respetar las reglas del gestor para nombrar tablas y el nombre no puede coincidir con otra tabla ya existente de la misma base de datos.
- La tabla recién creada está vacía, pero el SGBD la prepara para aceptar datos añadidos con la sentencia INSERT.
- La sentencia CREATE TABLE es muy amplia y sólo veremos las opciones más habituales. Tendremos que consultar el manual del gestor para ver otras opciones, según nuestras necesidades.

```
CREATE TABLE nombreTabla
(
        nombreColumna tipoColumna [ <restricciones> ]
        [, nombrecolumna tipoColumna [ <restricciones> ]]
[, <restricciones tabla> ] )
```

nombreTabla Nombre que le daremos a la tabla
nombreColumna Nombre de la columna que debe ser única en esa tabla
tipoColumna Tipo de datos predefinido o definido por el usuario.
restricciones Reglas de integridad (UNIQUE, IDENTITY, REFERENCES, NULL, NOT NULL, DEFAULT
RestriccionesTabla Reglas de integridad para la tabla. Cláusula CONSTRAINT

```
[CONSTRAINT nombreRestricción]
{
```

```
        [ { PRIMARY KEY | UNIQUE }
                { ( columna [,...n] ) }
        ]
        | FOREIGN KEY
                [(columna [,...n])]
                REFERENCES tablaReferencia [(columnaReferencia [,...n])]
        | CHECK] (expresiónLógica)
}
```

CONSTRAINT.- Es una palabra clave que indica el principio de la definición de una restricción.

nombreRestricción.- Es el nombre de una restricción. Los nombres de restricción deben ser únicos en una base de datos.

PRIMARY KEY.- Es una restricción de clave primaria para la tabla.

UNIQUE.- Se puede utilizar una restricción UNIQUE para asegurarse que no se escriben valores duplicados en una o varias columnas específicas que no forman parte de una clave primaria. Se puede utilizar para definir una Clave Alternativa. El SGBD asigna a dicha/s columna/s un índice único. Una tabla puede tener varias restricciones UNIQUE.

FOREIGN KEY...REFERENCES.- Es una restricción que proporciona integridad referencial para los datos de la columna o columnas. Las restricciones FOREIGN KEY requieren que cada valor de la columna exista en la columna de referencia correspondiente de la tabla a la que se hace referencia. Las restricciones FOREIGN KEY pueden hacer referencia sólo a columnas que sean restricciones PRIMARY KEY o UNIQUE en la tabla de referencia.

tablaReferencia.- Es el nombre de la tabla a la que hace referencia la restricción FOREIGN KEY.

(**columnaReferencia** [,...n]).- Es una columna o lista de columnas de la tabla a la que hace referencia la restricción FOREIGN KEY.

CHECK.- Es una restricción de chequeo que exige la integridad del dominio porque limita los valores posibles que se pueden escribir en una o varias columnas.

expresiónLógica.- Es una expresión lógica que debe ser TRUE para que se cumpla la restricción de chequeo.

5 El valor NULL

El valor NULL proporciona un modo sistemático de gestionar los datos que nunca han tenido un contenido, es decir, que no ha llegado a dárseles un valor en ningún momento.

NULL significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor 0, una cadena vacía o una cadena literal "null".

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos. Por ejemplo, en una tabla de libros, podemos tener valores nulos en el campo "precio" porque es posible que para algunos libros no le hayamos establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás, por ejemplo, los campos que identifican cada registro, como los códigos de identificación, que son clave primaria. Y hay otros que, aunque no sean clave, por la importancia de la información que soportan, tampoco deben estar vacios, lo cual deberá ser considerado en el diseño de la tabla.

Por defecto, los campos permiten valores nulos y entonces podremos introducir "NULL" (sin comillas) cuando no conocemos el valor.

6 Restricciones

Las restricciones de integridad son propiedades de la base de datos que se deben satisfacer en cualquier momento. Veremos diferentes tipos de restricciones que comprenden los siguientes puntos:

- Tratamiento de valores nulos.
- Valores por defecto.
- Integridad de clave primaria.
- Claves alternativas.
- Integridad referencial.
- Restricciones de integridad estáticas.

La integridad de los datos es la propiedad que asegura que información dada es correcta, al cumplir ciertas aserciones. Las restricciones de integridad aseguran que la información contenida en una base de datos es correcta y que la información contenida en la base de datos cumple ciertas restricciones para los diferentes estados.

Para incorporar el tratamiento de las restricciones de integridad en el sistema pueden realizarse:

- Añadiendo código adicional para verificar y asegurar que se cumplen las restricciones.
- Declarando las restricciones como parte del esquema de la base de datos.

La definición en la fase de diseño de las restricciones de integridad proporciona mayor número de ventajas, ya que:

- Reduce el coste de desarrollo de software.
- Es más confiable al ser centralizado y uniforme.
- Mantenimiento más fácil.

Como dijimos antes, estas restricciones las podemos implementar al momento de crear nuestras tablas o de modificarlas, también es necesario señalar que dichas restricciones son objetos propios de la base de datos y por lo tanto requieren de un nombre único compuesto del nombre del esquema al que pertenece y el nombre que lo identifica, un ejemplo sería *nombreEsquema.nombreRestriccion*.

Los diferentes tipos de restricción que existen son:

- PRIMARY KEY
- UNIQUE
- FOREIGN KEY
- CHECK
- DEFAULT

6.1 PRIMARY KEY

Es la más común de todas debido a que cada una de nuestras tablas debe ser completamente relacional y para lograr esto siempre debe existir una llave primaria dentro de cada tabla que identifique cada fila como única.

Para generar una llave primaria desde la creación de una tabla:

```
CREATE TABLE nombreEsquema.nombreTabla
(
        nombreColumna1 INT    NOT NULL,
        nombreColumna2 VARCHAR(100)  NOT NULL,
        nombreColumna3 NVARCHAR(200) NOT NULL,
        CONSTRAINT PK_nombreRestriccion PRIMARY KEY( nombreColumna1 )
);
```

Modificando una tabla:

```
ALTER TABLE nombreEsquema.nombreTabla
        ADD CONSTRAINT PK_nombreRestriccion PRIMARY KEY( nombreColumna1 );
```

Es posible agregar más columnas como parte de una llave primaria, se recomienda como buena práctica utilizar una nomenclatura en el nombre de la restricción que ayude a identificar de que tipo es, además de tener especial cuidado en nombrar las columnas que forman parte de la llave primaria ya que estás mismas serán utilizadas como referencia en una llave foránea en otra tabla. Cada vez que generamos una llave primaria, esta crea un índice tipo de clustered (índice agrupado) automáticamente.

Existen ciertos requerimientos para la creación de una llave primaria:

- La o las columnas utilizadas en una restricción PRIMARY KEY, no pueden aceptar NULL.
- No se pueden repetir valores en la o las columnas, deben ser únicos.
- Solamente puede existir una restricción de tipo PRIMARY KEY por cada tabla.

Para verificar las llaves primarias contenidas en nuestra base de datos podemos utilizar el siguiente código:

```
SELECT *
        FROM sys.key_constraints
        WHERE type = 'PK';
```

6.2 UNIQUE

Este tipo de restricción es muy parecida a PRIMARY KEY, las diferencias son las siguientes:

- También genera un índice automáticamente pero es de tipo de NON CLUSTERED.
- La tabla puede tener más de una restricción de tipo UNIQUE.
- Si puede aceptar NULL, pero solo una fila puede contenerlo ya que como su nombre lo indica, es de tipo UNIQUE o único.

```
CREATE TABLE nombreEsquema.nombreTabla
(
        nombreColumna1 INT    NULL,
        nombreColumna2 VARCHAR(100)  NOT NULL,
        nombreColumna3 NVARCHAR(200) NOT NULL,
        CONSTRAINT UQ_nombreRestriccion UNIQUE( nombreColumna1 ),
        CONSTRAINT UQ_nombreRestriccion2 UNIQUE( nombreColumna2 ),
        CONSTRAINT UQ_nombreRestriccion3 UNIQUE(  nombreColumna1,
                                                  nombreColumna2 )
);
```

Para consultar las restricciones UNIQUE se puede utilizar:

```
SELECT *
        FROM sys.key_constraints
        WHERE type = 'UQ';
```

6.3 FOREIGN KEY

Se forma de una columna o la combinación de varias columnas de una tabla que sirve como enlace hacia otra tabla donde en esta última, dicho enlace son la o las columnas que forman la PRIMARY KEY. En la primera tabla donde creamos la llave foránea es posible que existan valores duplicados de la/las columnas que conforman la llave primaria de la segunda tabla, además las columnas involucradas en la llave foránea deben tener el mismo tipo de datos que la llave primaria de la segunda tabla. Una llave foránea no crea un índice automáticamente, por lo que se recomienda generar uno para incrementar el rendimiento de la consulta.

```
CREATE TABLE nombreEsquema.nombreTabla
(
        nombreColumna1 INT    NULL,
        nombreColumna2 VARCHAR(100)  NOT NULL,
        nombreColumna3 NVARCHAR(200) NOT NULL,
        CONSTRAINT FK_nombreRestriccion FOREIGN KEY (nombreColumna1)
       REFERENCES nombreEsquema.otraTabla (nombreColumna1)
);
```

Modificando la tabla:

```
ALTER TABLE nombreEsquema.nombreTabla
        ADD CONSTRAINT FK_nombreRestriccion FOREIGN KEY(nombreColumna1)
        REFERENCES nombreEsquema.otraTabla (nombreColumna1)
```

Algunos requerimientos para la restricción FOREIGN KEY:

- Los valores ingresados en la o las columnas de la llave foránea, deben existir en la tabla a la que se hace referencia en la o las columnas de la llave primaria.
- Solo se pueden hacer referencia a llaves primaria de tablas que se encuentren dentro de la misma base de datos.
- Puede hacer referencia a otra columnas de la misma tabla.
- Solo puede hacer referencia a columnas de restricciones PRIMARY KEY o UNIQUE.
- No se puede utilizar en tablas temporales.

Para consultar las restricciones FOREIGN KEY, se puede utilizar:

```
SELECT *
        FROM sys.foreign_keys
        WHERE name = 'nombreEsquema.nombreTabla';
```

6.4 CHECK

Con este tipo de restricción, se especifica que los valores ingresados en la columna deben cumplir la regla o formula especificada. Por ejemplo:

```
CREATE TABLE nombreEsquema.nombreTabla
(
        nombreColumna1 INT    NULL,
        nombreColumna2 VARCHAR(100)  NOT NULL,
        nombreColumna3 NVARCHAR(200) NOT NULL,
--valores positivos
        CONSTRAINT CH_nombreRestriccion CHECK (nombreColumna1>=0),
-- solo valores iguales a 10 20 30 40
        CONSTRAINT CH_nombreRestriccion2 CHECK (nombreColumna1 IN
                        (10,20,30,40)),
--valores contenidos en un rango
```

```
        CONSTRAINT CH_nombreRestriccion3 CHECK (nombreColumna1>=1 AND
                nombreColumna1 <=30)
);
```

Modificando una tabla:

```
ALTER TABLE nombreEsquema.nombreTabla
    ADD CONSTRAINT CH_nombreRestriccion CHECK (nombreColumna1>=0);

ALTER TABLE nombreEsquema.nombreTabla
    ADD CONSTRAINT CH_nombreRestriccion2 CHECK (nombreColumna1 IN
                (10,20,30,40));

ALTER TABLE nombreEsquema.nombreTabla
    ADD CONSTRAINT CH_nombreRestriccion3 CHECK (nombreColumna1>=1
                AND nombreColumna1 <=30);
```

Algunos requerimientos son:

- Una columna puede tener cualquier número de restricciones CHECK.
- La condición de búsqueda debe evaluarse como una expresión booleana y no puede hacer referencia a otra tabla.
- No se pueden definir restricciones CHECK en columnas de tipo text, ntext o image.

Ventajas:

- Las expresiones utilizadas son similares a las que se usan en la clausula WHERE.
- Pueden llegar a ser una mejor alternativa que los TRIGGERS o disparadores.

Tener siempre en mente:

- Al momento de crear nuestra expresión, tomar en cuenta si la columna acepta valores NULL, por ejemplo si definimos nuestra restricción que acepte solo valores positivos (nombreColumna1>=0), NULL es un valor desconocido por lo tanto se insertará en la columna.
- No es posible obtener el valor previo después de realizar un UPDATE, si esto es necesario se recomienda usar un TRIGGER.

Para consultar las restricciones CHECK se puede utilizar:

```
SELECT *
    FROM sys.check_constraints
    WHERE parent_object_id = OBJECT_ID('nombreEsquema.nombreTabla');
```

6.5 DEFAULT

Se puede decir que no es una restricción, ya que solo se ingresa un valor en caso de que ninguno otro sea especificado.

```
CREATE TABLE nombreTabla
(
        nombreColumna1 INT    NULL CONSTRAINT DF_nombreRestriccion
                                                    DEFAULT(0),
        nombreColumna2 VARCHAR(100)  NOT NULL,
        nombreColumna3 NVARCHAR(200) NOT NULL,
);
```

Para obtener una lista de las restricciones DEFAULT:

```
SELECT *
        FROM sys.default_constraints
        WHERE parent_object_id = OBJECT_ID('nombreEsquema.nombreTabla');
```

7 Modificar y eliminar tablas: ALTER DROP

7.1 Modificar una tabla

Se puede modificar la estructura de una tabla, mediante la instrucción **ALTER TABLE**

La sentencia ALTER TABLE puede:

- Añadir una definición de columna a una tabla.
- Añadir o eliminar una clave primaria para una tabla.
- Añadir o eliminar una nueva clave ajena para una tabla.
- Añadir o eliminar una restricción de unicidad para una tabla.
- Añadir o eliminar una restricción de comprobación para una tabla.

La sintaxis puede parecer algo complicada pero sabiendo el significado de las palabras reservadas la sentencia se aclara bastante; ADD (añade), ALTER (modifica), DROP (elimina), COLUMN (columna), CONSTRAINT (restricción).

La sintaxis de la restricción es idéntica a la utilizada en CREATE TABLE.

ADD COLUMN

La cláusula ADD COLUMN (la palabra COLUMN es opcional) permite añadir una columna nueva a la tabla. Como en la creación de tabla, hay que definir la columna indicando su nombre, tipo de datos que puede contener, y si lo queremos alguna restricción de valor no nulo, clave primaria, clave foránea, e índice único.

```
ALTER TABLE nombreTabla
        ADD COLUMN nombreColumna INT NOT NULL
        CONSTRAINT nombreIndice UNIQUE
```

Con este ejemplo estamos añadiendo a la tabla una columna de tipo entero, requerida (no admite nulos) y con un índice sin duplicados llamado c1.

Cuando añadimos una columna lo mínimo que se puede poner sería:

```
ALTER TABLE nombreTabla
        ADD nombreColumna tipoDato
```

En este caso la nueva columna admitiría valores nulos y duplicados

ADD CONSTRAINT

Se utiliza para añadir una nueva restricción en la tabla

```
ALTER TABLE nombreTabla ADD CONSTRAINT c1 UNIQUE (nombreColumna)
```

Con este ejemplo estamos añadiendo a la tabla nombreTabla un índice único (sin duplicados) llamado c1 sobre la columna col3.

DROP COLUMN

Para borrar una columna utilizamos la cláusula DROP COLUMN (COLUMN es opcional) y el nombre de la columna que queremos borrar. Se perderán todos los datos almacenados en la columna

```
ALTER TABLE nombreTabla DROP COLUMN nombreColumna
```

También podemos escribir:

```
ALTER TABLE nombreTabla DROP nombreColumna
```

El resultado es el mismo, la columna nombreColumna desaparece de la tabla nombreTabla.

DROP CONSTRAINT

Para borrar una restricción se usa la cláusula DROP CONSTRAINT y el nombre de la restricción que queremos borrar, en este caso sólo se elimina la definición de la restricción pero los datos almacenados no se modifican ni se pierden.

```
ALTER TABLE nombreTabla DROP CONSTRAINT c1
```

Con esta sentencia borramos el índice c1 creado anteriormente pero los datos de la columna nombreColumna no se ven afectados por el cambio.

Cada una de las cláusulas de la sentencia ALTER TABLE puede aparecer sólo una vez en la sentencia. Se puede añadir una columna y definir una clave ajena en una única sentencia ALTER TABLE, pero deben utilizarse dos sentencias ALTER TABLE para añadir dos columnas.

7.2 Eliminar una tabla

Se puede eliminar de la base de datos una tabla que ya no es necesaria con la sentencia **DROP TABLE**

DROP TABLE *nombreTabla*

- Cuando la sentencia DROP TABLE suprime una tabla de la base de datos, su definición, todos sus datos y los índices asociados se pierden. Debido a sus serias consecuencias, debe utilizarse la sentencia DROP TABLE con mucho cuidado.
- La supresión de una tabla no será posible si la tabla es referenciada por una clave ajena.
- Existe también la posibilidad de eliminar las filas de una tabla y los índices, manteniendo la definición de la tabla y los objetos asociados, si se utiliza la sentencia

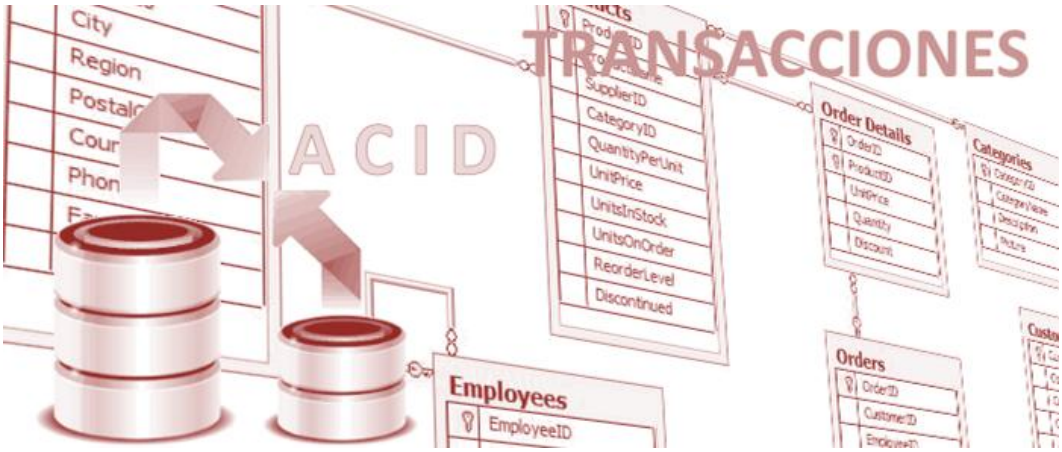
  TRUNCATE TABLE *nombreTabla*

8 Transacciones

8.1 Definición

Una transacción es un conjunto de operaciones que van a ser tratadas como una única unidad. Estas transacciones deben cumplir 4 propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

- **Atomicidad**. Es la forma de decir que una transacción es una unidad de trabajo. Cuando se acaba la transacción o se han ejecutado todas las instrucciones o no se ha hecho nada.
- **Coherencia**. Cuando una transacción se confirma o se deshace, debe dejarse la base de datos en un estado coherente. Si el gestor detecta que las instrucciones dejarían la base de datos en un estado incoherente la transacción no podría confirmarse.
- **Aislamiento**. Si dos o más transacciones se intentasen ejecutar a la vez, una de ellas comenzará primero y luego el resto. Nunca se podrán ejecutar a la vez. Cada transacción posterior podrá encontrar la base de datos modificada o no, según hubiese transcurrido la transacción anterior.
- **Durabilidad**. Una vez que se confirma una transacción sus efectos son permanentes.



No todas las operaciones SQL son transaccionales. Sólo son transaccionales las operaciones correspondientes al **DML** (Data Manipulation Language) es decir, sentencias **SELECT**, **INSERT**, **UPDATE** y **DELETE**

La transacción más simple es una única sentencia SQL. Por ejemplo una sentencia como esta:

```
UPDATE PRODUCTOS
        SET STOCKMINIMO=100, STOCKMAXIMO=200
        WHERE IDPRODUCTO=10
```

Es una transacción.

Esta es una transacción '***autocommit***', una transacción autocompletada. Para SQL la orden anterior incluye una transacción implícita, es decir, cuando se ejecuta la actualización, SQL asume que se deben hacer los dos cambios y, si por alguna circunstancia, se produjese un erro, no se actualizará ninguna de las dos columnas.

Este tipo de transacciones son las que SQL realiza por defecto.

Por supuesto este tipo de transacciones no requieren de nuestra intervención puesto que el sistema se encarga de todo. Sin embargo si hay que realizar varias operaciones y queremos que sean tratadas como una unidad tenemos que crear esas transacciones de manera explícita.

8.2 Transacciones explícitas

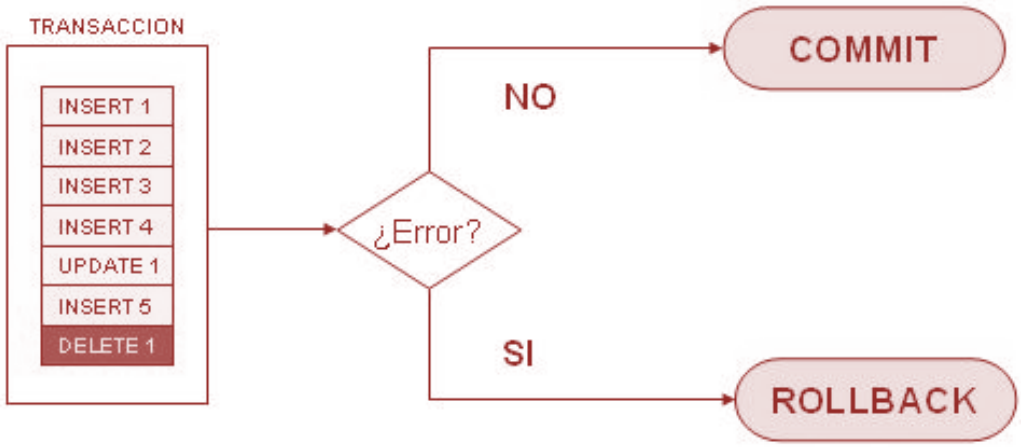
Una transacción explícita es un conjunto de operaciones que se ejecutan en una base de datos, y que son tratadas como una única unidad lógica por el SGBD. Es decir, una transacción es una o varias sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose en grupo. Este conjunto de operaciones debe marcarse como transacción para que todas las operaciones que la conforman tengan éxito o todas fracasen.

La sentencia que se utiliza para indicar el comienzo de una transacción es **BEGIN**.

Para confirmar una transacción se utiliza la sentencia **COMMIT**. Cuando realizamos COMMIT los cambios se escriben en la base de datos.

Para deshacer una transacción se utiliza la sentencia **ROLLBACK**. Cuando realizamos ROLLBACK se deshacen todas las modificaciones realizadas por la

transacción en la base de datos, quedando la base de datos en el mismo estado que antes de iniciarse la transacción.



Un ejemplo clásico de transacción son las transferencias bancarias. Para realizar una transferencia de dinero entre dos cuentas bancarias debemos descontar el dinero de una cuenta, realizar el ingreso en la otra cuenta y grabar las operaciones y movimientos necesarios, actualizar los saldos ... .Si en alguno de estos puntos se produce un fallo en el sistema podríamos hacer descontado el dinero de una de las cuentas y no haberlo ingresado en la otra. Por lo tanto, todas estas operaciones deben ser correctas o fallar todas. En estos casos, al confirmar la transacción (COMMIT) o al deshacerla (ROLLBACK) garantizamos que todos los datos quedan en un estado consistente.

En una transacción los datos modificados no son visibles por el resto de usuarios hasta que se confirme la transacción.

El siguiente ejemplo muestra una supuesta transacción bancaria:

```
DECLARE
        importe NUMBER;
        ctaOrigen VARCHAR2(23);
        ctaDestino VARCHAR2(23);
BEGIN
        importe := 100;
        ctaOrigen  := '2530 10 2000 1234567890';
        ctaDestino := '2532 10 2010 0987654321';
        UPDATE CUENTAS SET SALDO = SALDO - importe
                WHERE CUENTA = ctaOrigen;
        UPDATE CUENTAS SET SALDO = SALDO + importe
                WHERE CUENTA = ctaDestino;
        INSERT INTO MOVIMIENTOS(CUENTA_ORIGEN, CUENTA_DESTINO,IMPORTE,
                                FECHA_MOVIMIENTO)
                VALUES(ctaOrigen, ctaDestino, importe*(-1), SYSDATE);
        INSERT INTO MOVIMIENTOS(CUENTA_ORIGEN, CUENTA_DESTINO,IMPORTE,
                                FECHA_MOVIMIENTO)
                VALUES(ctaDestino,ctaOrigen, importe, SYSDATE);
```

```
        COMMIT;
EXCEPTION
WHEN OTHERS THEN
        dbms_output.put_line('Error en la transaccion:'||SQLERRM);
        dbms_output.put_line('Se deshacen las modificaciones);
        ROLLBACK;
END;
```

Si alguna de las tablas afectadas por la transacción tiene ***triggers***, las operaciones que realiza el trigger están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción.

Durante la ejecución de una transacción, una segunda transacción no podrá ver los cambios realizados por la primera transacción hasta que esta se confirme.

Fuentes

SQL y su gramática. Pilar García López. IES Nº 1 DE Gijón

Lenguaje Transact-SQL: creación y actualización de tablas. Joaquín Álvarez García/ Pilar García López. IES Nº 1 DE Gijón

www.aulaclic.es

chancrovsky.blogspot.com.es

Y otros.

Índice de contenidos