USO DEL LENGUAJE SQL

AUTOR: MAX JOSÉ BERMÚDEZ LEÓN

DICIEMBRE: 2020

San Marcos

Introducción

El SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales.

Para utilizar SQL desde un lenguaje de programación necesitaremos sentencias especiales que nos permitan distinguir entre las instrucciones del lenguaje de programación y las sentencias de SQL. La idea es que trabajando básicamente con un lenguaje de programación anfitrión se puede cobijar SQL como si fuese un huésped. Por este motivo, este tipo de SQL se conoce con el nombre de SQL hospedado.

Tabla de contenido

Las formas normales proporcionan el criterio necesario para determinar el grado de vulnerabilidad a inconsistencias lógicas y anomalías en los datos. Cuanto más alto sea el número de la forma normal, menos vulnerable será nuestra tabla. Y sí, he dicho nuestra tabla, porque las formas normales se aplican a ellas, individualmente. Para poder decir que una base de datos cumple una determinada forma normal, todas sus tablas deberán cumplirla.

Uso del lenguaje SQL sobre una base de datos

Cuando te enfrentas al desarrollo de un proyecto muchas veces encuentras que el mismo tiene como requisito que domines bases de datos en SQL. Previamente se ha hablado sobre qué son las bases de datos y los sistemas de base de datos relacionales, donde se explica el origen de SQL.

Que es SQL

El Lenguaje de Consulta Estructurado popularmente conocido por sus siglas en inglés como SQL, es un tipo de lenguaje de programación que ayuda a solucionar problemas específicos o relacionados con la definición, manipulación e integridad de la información representada por los datos que se almacenan en las bases de datos.

Algunos aspectos de SQL están basados en el cálculo relacional, algunos en el álgebra relacional que provienen del modelo relacional y otros a ninguno de los dos, sino que son parte de SQL.

SQL tiene comandos referentes a:

- Un lenguaje de definición de datos o DDL en inglés, que permite:
  - La creación de la estructura o esquema de base de datos.
  - La modificación de dicha estructura.
- Un lenguaje de manipulación de datos o DML en inglés, que hace posible:
  - La inserción de datos en tablas.
  - Las consultas sobre los datos de estas tablas.
  - La actualización de los datos contenidos en estas tablas.
  - La eliminación de los registros de estas tablas.
  - Peticiones de información más complejas que incluyen JOINs y subconsultas.
- Integridad: el DDL incluye también comandos para especificar las restricciones de integridad que el DBMS debe hacer cumplir.

- Definición de vistas o tablas derivadas.
- Funciones de agrupamiento: que permiten hacer cálculos de resúmenes.
- Control de Transacciones: como unidad de trabajo lógica, unidad de recuperación y mecanismo de concurrencia.
- Autorización: incluye comandos para otorgar los privilegios de acceso a las tablas, vistas y otros elementos de base de datos.
- SQL incorporado y dinámico: esto quiere decir que se puede incorporar comandos SQL en lenguajes de programación como C++, PHP, Java, etc.

El estándar SQL permite el intercambio entre diferentes DBMS, Esto significa que puedes usar los mismos comandos entre un manejador y otro, ya que SQL es un lenguaje estándar, lo cual es una gran ventaja porque los comandos para crear la estructura de base de datos o para manipular los datos siguen conservando la misma forma.

No obstante, aunque exista un estándar definido por ANSI, existen particularidades entre los diferentes DBMS en la gestión SQL. Por ejemplo: el lenguaje SQL de Oracle no es exactamente el mismo que el de Microsoft SQL Server; normalmente las diferencias son mínimas, pero existen y el programador debe hacer las adaptaciones que crea convenientes.

El lenguaje SQL consta de los siguientes componentes:

- Lenguaje de manipulación de datos (DML)
- Lenguaje de definición de datos (DDL)
- Lenguaje de control de datos (DCL)

1. El lenguaje de manipulación de datos (DML) consta de 4 comandos principales:
    a. Selección de información de la base de datos - SELECCIONAR
    b. Insertar información en una tabla de base de datos - INSERTAR
    c. Actualizar (cambiar) información en tablas de bases de datos - ACTUALIZAR
    d. Eliminar información de una base de datos - BORRAR
2. El lenguaje de definición de datos (DDL):

a. Se emplea para crear y cambiar la estructura de la base de datos y sus componentes: tablas, índices, vistas (tablas virtuales), así como disparadores y procedimiento almacenamiento.
b. Estos son solo algunos de los comandos básicos del lenguaje:
c. Creación de base de datos - CREATE DATABASE
d. Creación de tabla - CREATE TABLE
e. Cambio de tabla (estructura) - ALTER TABLE
f. Eliminación de tabla - DROP TABLE

3. El lenguaje de control de datos (DCL):
   a. Se utiliza para controlar los derechos de acceso a datos y ejecutar procedimientos en un entorno multiusuario.

Definición de datos (DDL)

Crear base de datos

La sentencia CREATE DATABASE se utiliza para crear una nueva base de datos SQL.

Sintaxis:

```
CREATE DATABASE databasename;
```

Ejemplo:

```
CREATE DATABASE testDB;
```

Crear Tabla

La instrucción CREATE TABLE se utiliza para crear una nueva tabla en una base de datos.

Sintaxis:

```
CREATE TABLE table_name (
  column1 datatype,
```

```
  column2 datatype,
  column3 datatype,
 ...
);
```

Los parámetros de columna especifican los nombres de las columnas de la tabla.

El parámetro de tipo de datos especifica el tipo de datos que puede contener la columna (por ejemplo, varchar, integer, date, etc.).

Ejemplo de SQL CREATE TABLE

El siguiente ejemplo crea una tabla llamada "Personas" que contiene cinco columnas: PersonID, LastName, FirstName, Address y City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

La columna PersonID es de tipo int y contendrá un número entero.

Las columnas Apellido, Nombre, Dirección y Ciudad son de tipo varchar y tendrán caracteres, y la longitud máxima de estos campos es 255 caracteres.

La tabla vacía "Personas" ahora se verá así:

PersonID	LastName	FirstName	Address	City

Crear tabla usando otra tabla

También se puede crear una copia de una tabla existente usando CREATE TABLE.

La nueva tabla obtiene las mismas definiciones de columna. Se pueden seleccionar todas las columnas o columnas específicas.

Si crea una nueva tabla utilizando una tabla existente, la nueva tabla se completará con los

valores existentes de la tabla anterior.

Sintaxis

```
CREATE TABLE new_table_name AS
    SELECT column1, column2, ...
    FROM existing_table_name
    WHERE …;
```

El siguiente SQL crea una nueva tabla llamada "TestTables" (que es una copia de la tabla "Clientes"):

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

Agregar columna

Para agregar una columna en una tabla, use la siguiente sintaxis:

```
ALTER TABLE table_name
ADD column_name datatype;
```

El siguiente SQL agrega una columna "Correo electrónico" a la tabla "Clientes":

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Modificar columna

Para cambiar el tipo de datos de una columna en una tabla, use la siguiente sintaxis, según MS SQL Server y Oracle.

SQL Server

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Oracle

ALTER TABLE table_name

MODIFY COLUMN column_name datatype;

Ejemplo de SQL ALTER TABLE

A partir de la tabla “Personas”

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Ahora queremos agregar una columna llamada "DateOfBirth" en la tabla "Personas".

Usamos la siguiente declaración SQL:

ALTER TABLE Persons

ADD DateOfBirth date;

Observe que la nueva columna, "DateOfBirth", es de tipo fecha y va a contener una fecha. El tipo de datos especifica qué tipo de datos puede contener la columna.

La tabla "Personas" ahora se verá así:

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Cambio tipos de dato en columna

Ahora queremos cambiar el tipo de datos de la columna denominada "DateOfBirth" en la tabla "Personas".

Usamos la siguiente declaración SQL:

ALTER TABLE Persons

ALTER COLUMN DateOfBirth date;

Eliminar columna

A continuación, queremos eliminar la columna llamada "DateOfBirth" en la tabla "Personas".

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Usamos la siguiente declaración SQL:

ALTER TABLE Persons

DROP COLUMN DateOfBirth;

Al eliminar la columna.

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Eliminar Tabla

La instrucción DROP TABLE se utiliza para eliminar una tabla existente en una base de datos.

Sintaxis

DROP TABLE table_name;

Eliminar Base de datos

La instrucción DROP DATABASE se utiliza para eliminar una base de datos SQL existente.

Sintaxis

DROP DATABASE databasename;

Índices

Crear Índice

La instrucción CREATE INDEX se utiliza para crear índices en tablas.

Los índices se utilizan para recuperar datos de la base de datos más rápidamente que de otra manera. Los usuarios no pueden ver los índices, solo se utilizan para acelerar las búsquedas / consultas.

Sintaxis

Crea un índice en una tabla. Se permiten valores duplicados:

CREATE INDEX index_name

ON table_name (column1, column2, ...);

Crear Índice Único

Crea un índice único en una tabla. No se permiten valores duplicados:

CREATE UNIQUE INDEX index_name

ON table_name (column1, column2, ...);

Eliminar Índice

SQL Server

DROP INDEX table_name.index_name;

Oracle

DROP INDEX index_name;

TRUNCATE

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DROP, es que, si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve

a crear y no ejecuta ninguna transacción.

Sintaxis:

TRUNCATE TABLE 'NOMBRE_TABLA';

Manipulación de datos (DML)

Como hemos visto, las instrucciones DML (Data Manipulation Language – Lenguaje de Manipulación de Datos) trabajan sobre los datos almacenados en nuestro SGBD, permitiendo consultarlos o modificarlos.

En general a las operaciones básicas de manipulación de datos que podemos realizar con SQL se les denomina operaciones CRUD (de Create, Read, Update and Delete, o sea, Crear, Leer, Actualizar y Borrar, sería CLAB en español, pero no se usa). Lo verás utilizado de esta manera en muchos sitios, así que apréndete ese acrónimo.

Hay cuatro instrucciones para realizar estas tareas:

- INSERT: Inserta filas en una tabla. Se corresponde con la “C” de CRUD.
- SELECT: muestra información sobre los datos almacenados en la base de datos. Dicha información puede pertenecer a una o varias tablas. Es la “R”.
- UPDATE: Actualiza información de una tabla. Es, obviamente, la “U”.
- DELETE: Borra filas de una tabla. Se corresponde con la “D”.

Seleccionar datos (SELECT)

Ahora nos vamos a centrar en la “R” de CRUD, es decir, en cómo recuperar la información que nos interesa de dentro de una base de datos, usando para ello el lenguaje de consulta o SQL. Ya nos preocuparemos luego de cómo llegamos a introducir los datos, primeramente.

Para realizar consultas sobre las tablas de las bases de datos disponemos de la instrucción SELECT. Con ella podemos consultar una o varias tablas. Es sin duda el comando más versátil del lenguaje SQL.

Existen muchas cláusulas asociadas a la sentencia SELECT (GROUP BY, ORDER,

HAVING, UNION). También es una de las instrucciones en la que con más frecuencia los motores de bases de datos incorporan cláusulas adicionales al estándar, que es el que veremos aquí.

Vamos a empezar viendo las consultas simples, basadas en una sola tabla. Veremos cómo obtener filas y columnas de una tabla en el orden en que nos haga falta.

El resultado de una consulta SELECT nos devuelve una tabla lógica. Es decir, los resultados son una relación de datos, que tiene filas/registros, con una serie de campos/columnas. Igual que cualquier tabla de la base de datos. Sin embargo, esta tabla está en memoria mientras la utilicemos, y luego se descarta. Cada vez que ejecutamos la consulta se vuelve a calcular el resultado.

La sintaxis básica de una consulta SELECT es la siguiente (los valores opcionales van entre corchetes):

*SELECT [ALL / DISTINC] [*] / [ListaColumnas_Expresiones] AS [Expresion]*
FROM Nombre_Tabla_Vista
WHERE Condiciones
ORDER BY ListaColumnas [ASC / DESC]

A continuación, analizaremos cada una de las partes de la consulta para entenderla mejor.

SELECT

Permite seleccionar las columnas que se van a mostrar y en el orden en que lo van a hacer. Simplemente es la instrucción que la base de datos interpreta como que vamos a solicitar información.

ALL / DISTINCT

ALL es el valor predeterminado, especifica que el conjunto de resultados puede incluir filas duplicadas. Por regla general nunca se utiliza.

DISTINCT especifica que el conjunto de resultados sólo puede incluir filas únicas. Es decir, si al realizar una consulta hay registros exactamente iguales que aparecen más de una vez, éstos se eliminan. Muy útil en muchas ocasiones.

Nombres de campos

Se debe especificar una lista de nombres de campos de la tabla que nos interesan y que

por tanto queremos devolver. Normalmente habrá más de uno, en cuyo caso separamos cada nombre de los demás mediante comas.

Se puede anteponer el nombre de la tabla al nombre de las columnas, utilizando el formato Tabla.Columna. Además de nombres de columnas, en esta lista se pueden poner constantes, expresiones aritméticas, y funciones, para obtener campos calculados de manera dinámica.

Si queremos que nos devuelva todos los campos de la tabla utilizamos el comodín “*” (asterisco).

Los nombres indicados deben coincidir exactamente con los nombres de los campos de la tabla, pero si queremos que en nuestra tabla lógica de resultados tengan un nombre diferente podemos utilizar:

AS

Permite renombrar columnas si lo utilizamos en la cláusula SELECT, o renombrar tablas si lo utilizamos en la cláusula FROM. Es opcional. Con ello podremos crear diversos alias de columnas y tablas. Enseguida veremos un ejemplo.

FROM

Esta cláusula permite indicar las tablas o vistas de las cuales vamos a obtener la información. De momento veremos ejemplos para obtener información de una sola tabla. Como se ha indicado anteriormente, también se pueden renombrar las tablas usando la instrucción “AS”.

WHERE

Especifica la condición de filtro de las filas devueltas. Se utiliza cuando no se desea que se devuelvan todas las filas de una tabla, sino sólo las que cumplen ciertas condiciones. Lo habitual es utilizar esta cláusula en la mayoría de las consultas.

Condiciones

Son expresiones lógicas a comprobar para la condición de filtro, que tras su resolución devuelven para cada fila TRUE o FALSE, en función de que se cumplan o no. Se puede utilizar cualquier expresión lógica y en ella utilizar diversos operadores como:

> (Mayor)

>= (Mayor o igual)

< (Menor)

<= (Menor o igual)

= (Igual)

<> o != (Distinto)

IS [NOT] NULL (para comprobar si el valor de una columna es o no es nula, es decir, si contiene o no contiene algún valor)

“Se dice que una columna de una fila es NULL si está completamente vacía. Hay que tener en cuenta que, si se ha introducido cualquier dato, incluso en un campo alfanumérico si se introduce una cadena en blanco o un cero en un campo numérico, deja de ser NULL.”

LIKE: para la comparación de un modelo. Para ello utiliza los caracteres comodín especiales: “%” y “_”. Con el primero indicamos que en su lugar puede ir cualquier cadena de caracteres, y con el segundo que puede ir cualquier carácter individual (un solo carácter). Con la combinación de estos caracteres podremos obtener múltiples patrones de búsqueda. Por ejemplo:

- El nombre empieza por A: Nombre LIKE ‘A%’
- El nombre acaba por A: Nombre LIKE ‘%A’
- El nombre contiene la letra A: Nombre LIKE ‘%A%’
- El nombre empieza por A y después contiene un solo carácter cualquiera: Nombre LIKE ‘A_’
- El nombre empieza una A, después cualquier carácter, luego una E y al final cualquier cadena de caracteres: Nombre LIKE ‘A_E%’

BETWEEN: para un intervalo de valores. Por ejemplo:

- Clientes entre el 30 y el 100: CodCliente BETWEEN 30 AND 100
- Clientes nacidos entre 1970 y 1979: FechaNac BETWEEN ‘19700101’ AND ‘19791231’
- IN( ): para especificar una relación de valores concretos. Por ejemplo: Ventas de los Clientes 10, 15, 30 y 75: CodCliente IN(10, 15, 30, 75)

Es posible combinar varias condiciones simples de los operadores anteriores utilizando los

operadores lógicos OR, AND y NOT, así como el uso de paréntesis para controlar la prioridad de los operadores (como en matemáticas). Por ejemplo: … (Cliente = 100 AND Provincia = 30) OR Ventas > 1000 … que sería para los clientes de las provincias 100 y 30 o cualquier cliente cuyas ventas superen 1000.

ORDER BY

Define el orden de las filas del conjunto de resultados. Se especifica el campo o campos (separados por comas) por los cuales queremos ordenar los resultados.

ASC / DESC

ASC es el valor predeterminado, especifica que la columna indicad en la cláusula ORDER BY se ordenará de forma ascendente, o sea, de menor a mayor. Si por el contrario se especifica DESC se ordenará de forma descendente (de mayor a menor).

Por ejemplo, para ordenar los resultados de forma ascendente por ciudad, y los que sean de la misma ciudad de forma descendente por nombre, utilizaríamos esta cláusula de ordenación:

… ORDER BY Ciudad, Nombre DESC …

Como a la columna Ciudad no le hemos puesto ASC o DESC se usará para la misma el valor predeterminado (que es ASC)

Algunos ejemplos

Para terminar este repaso a las consultas simples practicarlas un poco, veamos algunos ejemplos con la base de datos “Northwind” en SQL Server:

- Mostrar todos los datos de los Clientes de nuestra empresa:
  - SELECT * FROM Customers
- Mostrar apellido, ciudad y región (LastName, city, region) de los empleados de USA (nótese el uso de AS para darle el nombre en español a los campos devueltos):
  - SELECT E.LastName AS Apellido, City AS Ciudad, Region FROM Employees AS E WHERE Country = ‘USA’
- Mostrar los clientes que no sabemos a qué región pertenecen (o sea, que no tienen asociada ninguna región) :
  - SELECT * FROM Customers WHERE Region IS NULL

- Mostrar las distintas regiones de las que tenemos algún cliente, accediendo sólo a la tabla de clientes:
    - SELECT DISTINCT Region FROM Customers WHERE Region IS NOT NULL
- Mostrar los clientes que pertenecen a las regiones CA, MT o WA, ordenados por región ascendentemente y por nombre descendentemente.
    - SELECT * FROM Customers WHERE Region IN('CA', 'MT', 'WA') ORDER BY Region, CompanyName DESC
- Mostrar los clientes cuyo nombre empieza por la letra "W":
    - SELECT * FROM Customers WHERE CompanyName LIKE 'W%'
- Mostrar los empleados cuyo código está entre el 2 y el 9:
    - SELECT * FROM Employees WHERE EmployeeID BETWEEN 2 AND 9
- Mostrar los clientes cuya dirección contenga "ki":
    - SELECT * FROM Customers WHERE Address LIKE '%ki%'
- Mostrar las Ventas del producto 65 con cantidades entre 5 y 10, o que no tengan descuento:
    - SELECT * FROM [Order Details] WHERE (ProductID = 65 AND Quantity BETWEEN 5 AND 10) OR Discount = 0

En SQL Server, para utilizar nombres de objetos con caracteres especiales se deben poner entre corchetes. Por ejemplo, en la consulta anterior [Order Details] se escribe entre corchetes porque lleva un espacio en blanco en su nombre. En otros SGBDR se utilizan comillas dobles (Oracle, por ejemplo: "Order Details") y en otros se usan comillas simples (por ejemplo, en MySQL).

Insertar datos (INSERT)

La instrucción INSERT de SQL permite añadir registros a una tabla. Con ella podemos ir añadiendo registros uno a uno, o añadir de golpe tantos registros como nos devuelva una instrucción SELECT.

Veamos la sintaxis para cada uno de estos casos,

INSERT INTO NombreTabla [(Campo1, …, CampoN)] VALUES (Valor1, …, ValorN)

Siendo:

- NombreTabla: la tabla en la que se van a insertar las filas.
- (Campo1, …, CampoN): representa el campo o campos en los que vamos a introducir valores.
- (Valor1, …, ValorN): representan los valores que se van a almacenar en cada campo.

En realidad, la lista de campos es opcional especificarla (por eso la hemos puesto entre corchetes en la sintaxis general). Si no se indica campo alguno se considera que por defecto vamos a introducir información en todos los campos de la tabla, y por lo tanto se deben introducir valores para todos ellos y en el orden en el que han sido definidos. En la práctica se suelen especificar siempre por claridad y para evitar errores.

Por otro lado, los valores se deben corresponder con cada uno de los campos que aparecen en la lista de campos, tanto en el tipo de dato que contienen como en el orden en el que se van a asignar. Es decir, si se indican una serie de campos en un orden determinado, la lista de valores debe especificar los valores a almacenar en dichos campos, en el mismo orden exactamente. Si un campo no está en la lista, se almacenará dentro de éste el valor NULL.

Si un campo está definido como NOT NULL (es decir, que no admite nulos o valores vacíos), debemos especificarlo siempre en la lista de campos a insertar. De no hacerlo así se producirá un error a la ejecución la correspondiente instrucción INSERT.

Veamos unos ejemplos con la base de datos de ejemplo Northwind:

- INSERT INTO Region VALUES(35, 'Madrid')
- INSERT INTO Region (RegionID, RegionDescription) VALUES(35, 'Madrid')
- INSERT INTO Shippers (ShipperID, CompanyName) VALUES(4, 'Mi Empresa')

En el tercer ejemplo no hemos asignado valores a la columna Phone, por tanto, tomará automáticamente el valor NULL, salvo que tenga un valor DEFAULT asignado.

En realidad, en la mayor parte de los casos los campos de identificación de los registros, también conocidos como claves primarias, serán campos de tipo numérico con auto-incremento. Esto quiere decir que el identificador único de cada registro (el campo que

normalmente contiene un ID en su nombre, como RegionID o ShipperID) lo genera de manera automática el gestor de datos, incrementando en 1 su valor cada vez que se inserta un nuevo registro en la tabla. Por ello en estos casos no es necesario especificar este tipo de campos en las instrucciones INSERT. Por ello, por ejemplo, para insertar una nueva región en la tabla de regiones de Northwind lo único que hay que hacer es indicar su nombre, así:

- INSERT INTO Region (RegionDescription) VALUES('Madrid')

Nota: Aunque las tablas estén relacionadas entre sí de manera unívoca (por ejemplo, cada cabecera de factura con sus líneas de factura), no es posible insertar de un golpe los registros de varias tablas. Es necesario siempre introducir los registros uno a uno y tabla a tabla.

Inserción masiva de filas partiendo de consultas

Una segunda variante genérica de la instrucción INSERT es la que nos permite insertar de golpe múltiples registros en una tabla, bebiendo sus datos desde otra tabla (o varias tablas) de nuestra base de datos o, incluso en algunos SGBDR, de otra base de datos externa. En cualquier caso obteniéndolos a partir de una consulta SELECT convencional.

La sintaxis genérica de la instrucción que nos permite insertar registros procedentes de una consulta SELECT es la siguiente:

INSERT INTO NombreTabla [(Campo1, …, CampoN)]
Select …

El SELECT se indica a continuación de a lista de campos, y en lugar de especificar los valores con VALUES, se indica una consulta de selección. Es indispensable que los campos devueltos por esta instrucción SELECT coincidan en número y tipo de datos con los campos que se han indicado antes, o se producirá un error de ejecución y no se insertará ningún registro.

Veámoslo mejor con unos ejemplos sencillos de Northwind. Supongamos que tenemos una tabla nueva llamada NewCustomers en la que hemos introducido previamente los datos de nuevos clientes que no están en el sistema (por ejemplo por que los hemos cargado desde

un programa externo) y queremos agregar a nuestra tabla de clientes pre-existente los nuevos clientes que son del país "España":

```
INSERT INTO Customers
SELECT * FROM NewCustomers WHERE Country = 'Spain'
```

Lo que se suele hacer en estos casos es crear primeramente la consulta de selección, que generalmente será más compleja y puede involucrar varias tablas que no tienen por qué tener los mismos campos. Esta consulta la única condición que debe cumplir es que los campos devueltos tienen que ser del mismo tipo y orden que los que hay en la tabla donde insertamos o que los que indiquemos opcionalmente en la consulta de inserción.

Modificar datos (UPDATE)

Esta instrucción nos permite actualizar los valores de los campos de una tabla, para uno o varios registros, o incluso para todos los registros de una tabla.

Su sintaxis general es:

```
UPDATE NombreTabla
SET Campo1 = Valor1, ..., CampoN = ValorN
WHERE Condición
```

Siendo:

- NombreTabla: el nombre de la tabla en la que vamos a actualizar los datos.
- SET: indica los campos que se van a actualizar y con qué valores lo vamos a hacer.
- WHERE: Selecciona los registros de la tabla que se van a actualizar. Se puede aplicar todo lo visto para esta cláusula anteriormente, incluidas las sub-consultas.

Veamos unos ejemplos con la base de datos de ejemplo Northwind en SQL Server (serían idénticos en otros sistemas gestores de bases de datos relacionales como Oracle o MySQL):

Corrige el apellido del empleado con identificador 5. El correcto es "Smith":

- UPDATE Employees SET LastName = 'Smith' WHERE EmployeeID = 5

Elimina el valor del campo observaciones (Notes) de todos los empleados:

- UPDATE Employees SET Notes = NULL

Se puede notar como en este caso al carecer de una cláusula WHERE que restrinja la actualización se actualizan todos los registros.
“Es muy importante incluir una cláusula WHERE en las instrucciones UPDATE salvo en casos muy concretos como el del ejemplo anterior. De no hacerlo se actualizarán todos los registros de la tabla como acabamos de ver. Es habitual que, por error, se omita esta cláusula y se pierdan datos de forma irreversible (salvo que dispongamos de copias de seguridad o tengamos la posibilidad de hacer un ROLLBACK (cancelación de las instrucciones) si no hemos finalizado la transacción).”
Incrementa un 21% el precio de los productos pertenecientes la categoría que tiene el identificador 2:

- UPDATE Products SET UnitPrice = UnitPrice + (UnitPrice * 21 / 100) WHERE CategoryID = 2

Subconsultas como apoyo a la actualización
Podemos incluir una o varias subconsultas dentro una sentencia UPDATE. Éstas pueden estar contenidas en la cláusula WHERE o formar parte también de la cláusula SET. En este último caso se deben seleccionar el mismo número de campos y con los tipos de datos apropiados, para que cada uno de los valores pueda ser almacenado en la columna que hay a la izquierda del signo igual “=”, entre paréntesis, al lado de la instrucción SET.
Veamos unos ejemplos:

- Para todos los productos de la categoría bebidas ('beverages'), de la cual no conocemos su identificador, duplica su precio:

  UPDATE Products
  SET UnitPrice = UnitPrice * 2
  WHERE CategoryID = (SELECT CategoryID FROM Categories WHERE

CategoryName = 'Beverages')

Se puede observar que se utilizó una subconsulta en el filtro de modo que se averigua el identificador de la categoría que nos interesa, a partir de su nombre.

- Asignar a todos los productos de la categoría bebidas ('beverages'), el mismo precio que tiene el producto con número de identificador (clave primaria) igual a 5:

  UPDATE Products SET UnitPrice = (SELECT UnitPrice FROM Products WHERE ProductID = 5) WHERE CategoryID = (SELECT CategoryID FROM Categories WHERE CategoryName = 'Beverages')

Con el uso de esta sencilla instrucción UPDATE podemos actualizar cualquier dato o, lo que es más interesante, conjunto de datos que tengamos albergado en cualquier tabla de nuestra base de datos.

La única precaución que debemos tener es la de incluir una clausula WHERE apropiada que verdaderamente cualifique a los datos que queremos actualizar. El principal error en el uso de esta instrucción es precisamente ese, ya que podríamos acabar actualizando los registros equivocados, haciendo mucho daño en la base de datos. Para evitar problemas conviene realizar una consulta de selección precia que nos permita comprobar que las condiciones utilizadas en el WHERE son las apropiadas. También es conveniente tratar de lanzar la actualización dentro de una transacción que nos permita volver atrás todo el proceso en caso de producirse una equivocación y afectar al número equivocado de registros.

Borrar datos (DELETE)

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla.

Forma básica:

DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'

Ejemplo:

DELETE FROM My_table WHERE field2 = 'N';

Clausulas SQL

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Comando	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un

	orden específico
WHERE	Utilizada para determinar los registros seleccionados en la cláusula FROM

Operadores

Operadores Lógicos

Operador	Uso
AND	Es el “y” lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
BETWEEN	El operador BETWEEN se utiliza en la cláusula WHERE para seleccionar valores entre un rango de datos.
LIKE	El SQL Like es un tipo de operador lógico que se usa para poder determinar si una cadena de caracteres específica coincide con un patrón específico. Se utiliza normalmente en una sentencia Where para buscar un patrón específico de una columna.
In	Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista

Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Comando	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado

COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Operadores de Cadena

Las funciones de cadena realizan operaciones en valores de cadena y devuelven valores numéricos o de cadena.

Mediante las funciones de cadena, puede, por ejemplo, combinar datos, extraer una subcadena, comparar cadenas o convertir una cadena a todos los caracteres en mayúscula o minúscula.

Es importante indicar que según el SGDB que se utilice puede tener más funciones de manejo de cadenas.

En MSSQL Server tenemos las siguientes,

CONCAT (string_value1, string_value2 [, string_valueN])

LTRIM (expresión_caracteres)

RTRIM (expresión de caracteres)

SUBSTRING (expresión, inicio, longitud)

ASCII (expresión de caracteres)

REPLICATE (string_expression, integer_expression)

REVERSE (string_expression)

SUPERIOR (expresión de caracteres)

TRIM (cadena [caracteres de])
STRING_SPLIT (cadena, separador)
MATERIAL (expresión de caracteres, inicio, longitud, reemplazar con expresión)
REPLACE (string_expression, string_pattern, string_replacement)

TRIM

El recorte se utiliza para eliminar el espacio de escritura al principio o al final de la selección
En MSSQL no hay un solo uso del TRIM

- SELECT LTRIM(' Hello ') --returns 'Hello '
- SELECT RTRIM(' Hello ') --returns ' Hello'
- SELECT LTRIM(RTRIM(' Hello ')) --returns 'Hello'

MySql y Oracle

- SELECT TRIM(' Hello ') --returns 'Hello'

CONCAT

En SQL (ANSI / ISO estándar), el operador para la concatenación de cadenas es || . Esta sintaxis es compatible con todas las bases de datos principales, excepto SQL Server:

- SELECT 'Hello' || 'World' || '!'; --returns HelloWorld!

Muchas bases de datos admiten una función CONCAT para unir cadenas:

- SELECT CONCAT('Hello', 'World'); --returns 'HelloWorld'

Algunas bases de datos admiten el uso de CONCAT para unir más de dos cadenas (Oracle no lo hace):

- SELECT CONCAT('Hello', 'World', '!'); --returns 'HelloWorld!'

En algunas bases de datos, los tipos sin cadena deben ser convertidos o convertidos:

- SELECT CONCAT('Foo', CAST(42 AS VARCHAR(5)), 'Bar'); --returns 'Foo42Bar'

Algunas bases de datos (por ejemplo, Oracle) realizan conversiones sin pérdida implícitas.

Por ejemplo, un CONCAT en un CLOB y NCLOB produce un NCLOB . Un CONCAT en un número y un varchar2 da como resultado un varchar2 , etc .:

- SELECT CONCAT(CONCAT('Foo', 42), 'Bar') FROM dual; --returns Foo42Bar

Algunas bases de datos pueden usar el operador + no estándar (pero en la mayoría, + solo funciona con números):

- SELECT 'Foo' + CAST(42 AS VARCHAR(5)) + 'Bar';

UPPER / LOWER

Para convertir la cadena en Mayúscula o Minuscula.

- SELECT UPPER('HelloWorld') --returns 'HELLOWORLD'
- SELECT LOWER('HelloWorld') --returns 'helloworld'

SubString

La sintaxis es: SUBSTRING (string_expression, start, length) . Tenga en cuenta que las cadenas SQL son 1-indexadas.

Sintaxis:

- SELECT SUBSTRING('Hello', 1, 2) --returns 'He'
- SELECT SUBSTRING('Hello', 3, 3) --returns 'llo'

Esto se usa a menudo junto con la función LEN() para obtener los últimos n caracteres de una cadena de longitud desconocida.

- DECLARE @str1 VARCHAR(10) = 'Hello', @str2 VARCHAR(10) = 'FooBarBaz';
- SELECT SUBSTRING(@str1, LEN(@str1) - 2, 3) --returns 'llo'
- SELECT SUBSTRING(@str2, LEN(@str2) - 2, 3) --returns 'Baz'

DIV

Divide una expresión de cadena usando un separador de caracteres. Tenga en cuenta que STRING_SPLIT() es una función con valores de tabla.

Sintaxis

- SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');

- Resultado:

value

Lorem

ipsum

dolor

sit

amet.

LEN

Sql Server

El LEN no cuenta el espacio al final.

Sintaxis:

- SELECT LEN('Hello') -- returns 5

El DATALENGTH cuenta el espacio al final.

- SELECT DATALENGTH('Hello') -- returns 5
- SELECT DATALENGTH('Hello '); -- returns 6

Sin embargo, se debe tener en cuenta que DATALENGTH devuelve la longitud de la representación de bytes subyacente de la cadena, que depende, entre otras cosas, del conjunto de caracteres utilizado para almacenar la cadena.

- DECLARE @str varchar(100) = 'Hello ' --varchar is usually an ASCII string, occupying 1 byte per char
- SELECT DATALENGTH(@str) -- returns 6
- DECLARE @nstr nvarchar(100) = 'Hello ' --nvarchar is a unicode string, occupying 2 bytes per char
- SELECT DATALENGTH(@nstr) -- returns 12

Oracle

- Sintaxis: Longitud (char)

Ejemplos:

- SELECT Length('Bible') FROM dual; --Returns 5
- SELECT Length('righteousness') FROM dual; --Returns 13
- SELECT Length(NULL) FROM dual; --Returns NULL

Replace

Sintaxis:

- REPLACE(Cadena para buscar , Cadena para buscar y reemplazar , Cadena para colocar en la cadena original)

Ejemplo:

- SELECT REPLACE('Peter Steve Tom', 'Steve', 'Billy') --Return Values: Peter Billy Tom

LEFT / RIGHT

La sintaxis es:

- IZQUIERDA (expresión-cadena, entero)
- DERECHA (string-expresión, entero)

Ejemplos

- SELECT LEFT('Hello',2) --return He
- SELECT RIGHT('Hello',2) --return lo

Oracle no tiene funciones LEFT / RIGHT. Se pueden emular con SUBSTR y LONGITUD.

- SUBSTR (string-expresión, 1, entero)
- SUBSTR (expresión-serie, longitud (expresión-serie) -integer + 1, entero)

Ejemplos

- SELECT SUBSTR('Hello',1,2) --return He

- SELECT SUBSTR('Hello',LENGTH('Hello')-2+1,2) --return lo

Trabajando con consultas

Consultas de selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros. Este conjunto de registros es modificable.

Básicas

La sintaxis básica de una consulta de selección es:

- # SELECT Campos FROM Tabla;
- # SELECT Nombre, Telefono FROM Clientes;

Ordenar los registros

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY:

- # SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;

Se pueden ordenar los registros por más de un campo:

- # SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;

Y se puede especificar el orden de los registros: ascendente mediante la claúsula (ASC -se toma este valor por defecto) ó descendente (DESC):

- # SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre ASC;

Consultas con predicado

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos

selecciona todos los registros que cumplen las condiciones de la instrucción SQL:

- # SELECT ALL FROM Empleados;
- # SELECT * FROM Empleados;

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

- # SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes .El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

- # SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos:

- # SELECT DISTINCT Apellido FROM Empleados;

DISTINCTROW

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campo indicados en la cláusula SELECT:

- # SELECT DISTINCTROW Apellido FROM Empleados;

Criterios de selección

Operadores Lógicos

Los operadores lógicos soportados por SQL son:

- AND, OR, XOR, Eqv, Imp, Is y Not.

A excepción de los dos últimos todos poseen la siguiente sintaxis:

- <expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico:

- # SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
- # SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;
- # SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
- # SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR (Provincia = 'Madrid' AND Estado = 'Casado');

Operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between:

- # SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;

(Devuelve los pedidos realizados en la provincia de Madrid)

- # SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional') FROM Editores;

(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo,'Nacional' en caso contrario)

Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión LIKE modelo

Operador IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los indicados en una lista. Su sintaxis es:

expresión [Not] In(valor1, valor2, . . .)

- # SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');

Clausula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM:

- # SELECT Apellidos, Salario FROM Empleados

WHERE Salario > 21000;

- # SELECT Id_Producto, Existencias FROM Productos WHERE Existencias <= Nuevo_Pedido;

Agrupamiento de registros (Agregación)

AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta:

Avg(expr)

La función Avg no incluye ningún campo Null en el cálculo. Un ejemplo del funcionamiento de AVG:

- # SELECT Avg(Gastos) AS Promedio FROM

Pedidos WHERE Gastos > 100;

MAX, MIN

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

- Min(expr)
- Max(expr)

Un ejemplo de su uso:

- # SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais = 'Costa Rica';
- # SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais = 'Costa Rica';

SUM

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(expr)

Por ejemplo:

- # SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM DetallePedido;

GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro:

- # SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada:

- # SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;

HAVING

Es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

- # SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia

  HAVING Sum(Stock) > 100 AND NombreProducto Like BOS*;

Conclusiones y recomendaciones

Ademas de conocer sobre el diseño de bases de datos, es muy importante conocer sobre el lenguaje SQL. Ambos procesos van de la mano, no es posible tener una base de datos eficiente si no se hace un análisis y diseño correcto, además de esto si el modelo de base de datos es igual de correcto, al momento de utilizar el lenguaje para las consultas se hace mucho mas sencillo el construir las sentencias de consulta.

La practica hace el maestro con esto se quiere decir que el llegar a manejar el lenguaje sql es mucha practica y tener muy claro los conceptos de construcción de sentencias.

Si bien es cierto SQL es un estándar de la industria, los SGDB, ya sea Oracle, SQL, MySQL, PostgreSQL, han aplicado las variantes en la construcción de las sentencias.

En resumen, SQL es un lenguaje muy poderoso y el mismo debe ser practicado para lograr dominarlo.

Recomendaciones:

Consultar la documentación de los SGDB para saber de las variantes del SQL.
Investigar sobre las mejores practicas para la construcción de consultas.
Investigar sobre las bondades que ofrece el SGDB y explotar las mismas.

Referencias bibliográficas

- Bertone, N. (2017). Introducción a las bases de datos: fundamentos y diseño. Pearson Educación
- Silberschatz, A. & Sudarshan, S. (2018). Fundamentos de bases de datos (6a. ed.). McGraw-Hill Interamericana.
- Bernal, Nieto. (2017). Diseño de base de datos. Universidad del Norte
- Pulido Romero, E. Escobar Domínguez, Ó. y Núñez Pérez, J. Á. (2019). Base de datos. Grupo Editorial Patria.

- W3SCHOOL. (2019). Tutorial de SQL. Sitio donde se descargo la información: https://www.w3schools.com/
- CampusMVP. (2015). Fundamentos de SQL: Como realizar consultas simples son SELECT. Sitio donde se descargo la información: https://www.campusmvp.es/recursos
- El Blog de Ceupe. (2020). Lenguaje de Consulta SQL. Sitio donde se descargo la informacion: https://www.ceupe.com/blog/lenguaje-de-consulta-sql.html
- GeoTalleres. (2014). Conceptos basicos de SQL. Sitio donde se descargo la información: https://geotalleres.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html
- Stack Overflow Documentation. (2016). Empezando con SQL. Sitio donde se descargo la información: https://sodocumentation.net/es/sql/topic/184/empezando-con-sql

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica