UNIVERSIDAD TÉCNOLOGICA NACIONAL
FACULTAD REGIONAL GENERAL PACHECO

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

Base de Datos II

Teoría SQL – Parte 1

ING. ARIEL HERRERA

Contenido

SQL

1.- INTRODUCCIÓN

SQL (Structured Query Language, Lenguaje de consultas estructurado), es un lenguaje que nos permitirá operar con nuestras bases de datos de una forma sencilla y potente. Se encuentra muy difundido y es soportado por todos los grandes sistemas gestores de bases de datos como pueden ser: Oracle, Interbase, Microsoft, Informix, Sybase, etc. La gran ventaja que nos ofrece SQL frente a otras formas de trabajar es su estandarización, no es necesario que cuando nos encontramos ante diferentes sistemas gestores tengamos que aprender como funciona cada uno de ellos de una forma más o menos detallada, simplemente debemos conocer como se trabaja con SQL, ya que su sintaxis y estructuración sufre muy pocas variaciones de un sistema otro. Por lo tanto SQL nos ofrece una única forma de trabajar con independencia del sistema con el quetengamos que hacerlo, con el consiguiente ahorro de tiempo y energías por partede programador.

Un poco de historia. SQL comenzó a ver la luz sobre el final de la década de los 70, implementado como un prototipo por IBM y en 1979 aparece el primer sistema gestor de bases de datos que lo soporta, Oracle. Poco tiempo después, otros sistemas como DB2, Interbase, etc, también quisieron adoptar a éste. Yaen 1987 se le confirió la estandarización a través de las normas ISO. El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo: En el año 1986 el SQL-86 alias SQL-87 confirmada por ISO en 1987, 1989 SQL-89, en 1992 SQL-92 conocida como SQL2, 1999 SQL:1999 alias SQL2000, en el 2003 el SQL:2003, 2005 el SQL:2005 y en el año 2008 el SQL:2008 en el que se le han añadido nuevas características como la creación de TADs (tipos abstractos de datos), reglas activas (triggers - disparadores), cursores sensitivos, operador de unión recursiva, etc., el SQL14 en 2014, y el SQL:2016 en 2016 que permite búsqueda de patrones, funciones de tabla polimórficas y compatibilidad con los ficheros JSON, utilizado hasta la actualidad.

Aunque todo el SQL está estandarizado, sí es cierto que encontramos pequeñas diferencias en la sintaxis de las sentencias en los diferentes sistemas gestores. Estas diferencias repito que son mínimas y una vez realizados los scripts de SQL que, por ejemplo, diseñan una base de datos, tendremos que hacer muy pocas variaciones si

queremos que esos scripts se ejecuten en diferentes sistemas gestores.

1.1.- Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

1.2.- Comandos

Existen dos tipos de comandos SQL:

· los DDL (Data Definition Language, Lenguaje de definición de datos) que permiten crear y definir nuevas bases de datos, campos e índices.

· los DML (Data Manipulation Language, Lenguaje de Manipulación de Datos) que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL

CREATE Utilizado para crear nuevas tablas, campos e índices.

DROP Empleado para eliminar tablas e índices.

ALTER Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML

SELECT Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.

INSERT Utilizado para cargar lotes de datos en la base de datos en una única operación.

DELETE Utilizado para eliminar registros de una tabla de una base de datos.

UPDATE Utilizado para modificar los valores de los campos y registros especificados.

1.3.- Cláusulas

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

FROM Utilizada para especificar la tabla de la cual se van a seleccionar los registros.

WHERE Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

GROUP BY Utilizada para separar los registros seleccionados en grupos específicos.

HAVING Utilizada para expresar la condición que debe satisfacer cada grupo.

ORDER BY Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

1.4.- Operadores Lógicos

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.

1.5.- Operadores de Comparación

< Menor que

> Mayor que

<> Distinto de

<= Menor ó Igual que

>= Mayor ó Igual que

= Igual que

BETWEEN Utilizado para especificar un intervalo de valores.

LIKE Utilizado en la comparación de un modelo.

IN Utilizado para especificar registros de una base de datos.

1.6.- 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.

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

2.- 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.

2.1.- Consultas básicas

La sintaxis básica de una consulta de selección es la siguiente:

SELECT Campos **FROM** Tabla;

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

SELECT Nombre, Telefono **FROM** Clientes;

Esta consulta devuelve el campo nombre y teléfono de la tabla clientes.

2.2.- Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

SELECT CodigoPostal, Nombre, Telefono **FROM** Clientes **ORDER BY** Nombre;

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

SELECT CodigoPostal, Nombre, Telefono FROM Clientes **ORDER BY** CodigoPostal, Nombre;

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

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

2.3.- Consultas con Predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son:
ALL Devuelve todos los campos de la tabla
TOP Devuelve un determinado número de registros de la tabla
DISTINCT Omite los registros cuyos campos seleccionados coincidan totalmente

ALL (*)

El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

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**;

El valor que va a continuación de TOP debe ser un Integer sin signo. TOP no afecta a la posible actualización de la consulta.

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.
Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

SELECT DISTINCT Apellido **FROM** Empleados;

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

2.4.- Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho opor otras

circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

SELECT DISTINCT Apellido **AS** Empleado **FROM** Empleados;

3.- Criterios de Selección

En el capítulo anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este capítulo se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas.
Antes de comenzar el desarrollo de este capítulo hay que recalcar dos detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda hace referencia a las fechas. Las fechas en SqlServer se pueden escribir en distintos formatos y el servidor lo interpretará al formato que tenga configurado, Se pueden usar distintos separadores como guiones (-) y barras (/) (y hasta puntos en las versiones más avanzadas) y las fechas en código sql deben escribirse entre comillas simples (‘).Por ejemplo si deseamos referirnos al día 3 de Septiembre de 1995 podremos hacerlo de la siguiente forma; ‘09-03-95’, ‘9/3/95’ o sus variantes.

3.1.- Operadores Lógicos

Los operadores lógicos comprueban la veracidad de alguna condición. Al igual que los operadores de comparación, devuelven el tipo de datos **Boolean** con el valor TRUE, FALSE o UNKNOWN.

Operator	Significado
ALL	TRUE si el conjunto completo de comparaciones es TRUE.
AND	TRUE si ambas expresiones booleanas son TRUE.
ANY	TRUE si cualquier miembro del conjunto de comparaciones es TRUE.
BETWEEN	TRUE si el operando está dentro de un intervalo.
EXISTS	TRUE si una subconsulta contiene cualquiera de las filas.
IN	TRUE si el operando es igual a uno de la lista de expresiones.
LIKE	TRUE si el operando coincide con un patrón.
NOT	Invierte el valor de cualquier otro operador booleano.

Operator	Significado
OR	TRUE si cualquiera de las dos expresiones booleanas es TRUE.
SOME	TRUE si alguna de las comparaciones de un conjunto es TRUE.

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

Haciendo ctrl+click en cualquiera de los operadores de la tabla anterior, los redirigirá al sitio de Microsoft Sql Server (Transact-Sql), para profundizar si así lo quisieran.

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');

3.2.- Intervalos de Valores

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

campo [Not] Between valor1 And valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

SELECT * FROM Pedidos **WHERE** CodPostal **Between** 28000 **And** 28999; (Devuelve los pedidos realizados en la provincia de Madrid)

3.3.- Uso del operador LIKE

Determina si una cadena de caracteres dada coincide o no con un determinado modelo. Un modelo puede incluir caracteres normales y caracteres comodín. Durante la coincidencia de patrones, los caracteres regulares deben coincidir exactamente con los caracteres especificados en la cadena de caracteres. Sin embargo, los caracteres comodín pueden coincidir con fragmentos arbitrarios de la cadena de caracteres. Con los caracteres comodín el operador LIKE es más flexible que los operadores de comparación de cadenas = y <>.

Carácter comodín	Descripción	Ejemplo
%	Cualquier cadena de cero o más caracteres.	WHERE titulo LIKE '%compad%' busca todos los títulos de libros que contengan la palabra 'compad' en cualquier parte del título.
_ (subrayado)	Cualquier carácter individual	WHERE NOMBRE LIKE 'MARI_' busca todos los nombres de cuatro letras que comiencen con MARI (Maria, Mario, etc.).
[ ]	Cualquier carácter individual de intervalo ([a-f]) o del conjunto ([abcdef]) especificado.	WHERE NOMBRE LIKE '[P-Z]uarez buscaNombres de autores que terminen con uarez y comiencen con cualquier carácter individual entre P y Z, por ejemplo Suarez, Zuarez, Vuarez, Tuarez, etc.
[^]	Cualquier carácter individual que no se encuentre en el intervalo ([^a-f]) o el conjunto ([^abcdef]) especificado.	WHERE NOMBRE LIKE 'de[^s]%' busca todos los apellidos de autores que comienzan con **de** y en los que la siguiente letra no sea **s**.

3.3.1.- Utilizar caracteres comodín como literales

Los caracteres comodín de concordancia de patrón se pueden utilizar como literales. Para utilizar como literal un carácter comodín, inclúyalo entre corchetes.La tabla muestra varios ejemplos del uso de la palabra clave LIKE y los caracteres comodín [].

Símbolo	Significado
LIKE '5[%]'	5%
LIKE '[_]n'	_n
LIKE '[a-cdf]'	a, b, c, d o f
LIKE '[-acdf]'	-, a, c, d o f
LIKE '[[ ]'	[
LIKE ']'	]
LIKE 'abc[_]d%'	abc_d y abc_de
LIKE 'abc[def]'	abcd, abce y abcf

3.4.- El Operador In

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

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

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

3.5.- La cláusula 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. Después de escribir esta cláusula se deben especificar las condiciones

expuestas en los apartados 3.1 y 3.2. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. 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;

SELECT * FROM Pedidos **WHERE** Fecha_Envio = ‘5/10/94’;

SELECT Apellidos, Nombre **FROM** Empleados **WHERE** Apellidos = 'King';

SELECT Apellidos, Nombre **FROM** Empleados **WHERE** Apellidos Like 'S%';

SELECT Apellidos, Salario **FROM** Empleados **WHERE** Salario **Between** 200 **And** 300;

SELECT Apellidos, Salario **FROM** Empl **WHERE** Apellidos **Between** 'Lon' **And** 'Tol';

SELECT Id_Pedido, Fecha_Pedido **FROM** Pedidos **WHERE** Fecha_Pedido **Between** ‘1-1-94’ And ‘30-6-94’;

SELECT Apellidos, Nombre, Ciudad **FROM** Empleados
WHERE Ciudad **In** ('Sevilla', 'Los Angeles', 'Barcelona');

4.- Agrupamiento de Registros

4.1.- GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluyeuna función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

SELECT campos **FROM** tabla **WHERE** criterio **GROUP BY** campos del grupo

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.
Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, yla cláusula HAVING para filtrar los registros una vez agrupados.
Un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no esta incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada.
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;

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING. 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%’;

4.2.- AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente

Avg(expr)

En donde expr representa el campo que contiene los datos **numéricos** para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

SELECT Avg(Gastos**) AS** Promedio **FROM** Pedidos **WHERE** Gastos > 100;

4.3.- Count

Calcula el número de registros devueltos por una consulta. Su sintaxis es lasiguiente

Count(expr)

En donde expr contiene el nombre del campo que desea contar. Los operandosde expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras delas funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto. Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

SELECT Count(*) AS Total **FROM** Pedidos;

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

```
SELECT Count([FechaEnvío & Transporte]) AS Total FROM Pedidos;
```

4.4.- 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)

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais = 'España';
```

```
SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais =
'España';
```

4.5.- StDev, StDevP

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es:

StDev(expr)
StDevP(expr)

En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL)
StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la

consulta contiene menos de dos registros (o ningún registro para StDevP), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

SELECT StDev(Gastos**) AS** Desviacion **FROM** Pedidos **WHERE** Pais = 'España';

SELECT StDevP(Gastos**) AS** Desviacion **FROM** Pedidos **WHERE** Pais= 'España';

4.6.- Sum

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

Sum(expr)

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definidapor el usuario pero no otras de las funciones agregadas de SQL).

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

4.7.- Var, VarP

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

Var(expr)
VarP(expr)

VarP evalúa una población, y Var evalúa una muestra de la población. Expr el nombre del campo que contiene los datos que desean evaluarse o una expresión

que realiza un cálculo utilizando los datos de dichos campos.
Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).
Si la consulta contiene menos de dos registros, Var y VarP devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar Var y VarP en una expresión de consulta o en una Instrucción SQL.

SELECT Var(Gastos**) AS** Varianza **FROM** Pedidos **WHERE** Pais = 'España';

SELECT VarP(Gastos**) AS** Varianza **FROM** Pedidos **WHERE** Pais = 'España';