Transact-SQL y Consultas simples
Transact-SQL - Simple Queries

Índice

1 Conceptos básicos de SQL

SQL (*STRUCTURED QUERY LANGUAGE*), Lenguaje Estructurado de Interrogación/Consulta es el lenguaje utilizado para definir, controlar y acceder a los datos almacenados en una base de datos relacional.

Como ejemplos de sistemas gestores de bases de datos que utilizan SQL podemos citar DB2, SQL Server, Oracle, MySql, Sybase, PostgreSQL, Foxpro, Access…

El SQL es un lenguaje universal que se emplea en cualquier sistema gestor de bases de datos relacional. Tiene un estándar definido, a partir del cual cada sistema gestor ha desarrollado su versión propia.

En SQL Server la versión de SQL que se utiliza se llama TRANSACT-SQL.

El SQL en el principio era un lenguaje orientado únicamente a la definición y al acceso a los datos por lo que no se podía considerar un lenguaje de programación como tal ya que no incluía funcionalidades como son estructuras condicionales, bucles, formateo de la salida, etc.… Pero como veremos durante el curso eso ha pasado a la historia y realmente ahora solo le falta la parte de presentación.

Se puede ejecutar directamente en modo interactivo (consola), pero también se suele emplear embebido en programas escritos en lenguajes de programación convencionales. En estos programas se mezclan las instrucciones del propio lenguaje (denominado anfitrión) con llamadas a procedimientos de acceso a la base de datos que utilizan el SQL como lenguaje de acceso. Como por ejemplo en Visual Basic, Java, C#, PHP, .NET, etc.

Incluso ahora tenemos lenguajes que tienen sus propias instrucciones para acceder a una base de datos, pero éstas también recuerdan las instrucciones SQL.

Las instrucciones SQL se clasifican según su propósito en tres grupos:

- El DDL (*DATA DESCRIPTION LANGUAGE*) Lenguaje de Descripción de Datos.
- El DML (*DATA MANIPULATION LANGUAGE*) Lenguaje de Manipulación de Datos.
- El DCL (*DATA CONTROL LANGUAGE*) Lenguaje de Control de Datos.

● El **DDL**, es la parte del SQL dedicada a la definición de la base de datos, consta de sentencias para definir la **estructura** de la base de datos, estas sentencias permiten crear la base de datos, crear, modificar o eliminar la estructura de las tablas, crear índices, definir reglas de validación de datos, relaciones entre las tablas, etc. Permite definir gran parte del nivel interno de la base de datos. Por este motivo estas sentencias serán utilizadas normalmente por el administrador de la base de datos. Esta parte tiene que ver con el nivel físico de la base de datos, el nivel más cercano a la máquina y por lo tanto es la parte que más varía de un sistema a otro ya que cada sistema organiza sus datos de una forma u otra y maneja conceptos que a lo mejor otro sistema no emplea. El ANSI/ISO define una serie de sentencias pero luego cada sistema ha realizado una serie de modificaciones tanto a nivel de cláusula dentro de la sentencia como a nivel de implementar nuevas sentencias no contempladas en el estándar.

● El **DML** se compone de las instrucciones para el manejo de los datos, para insertar nuevos datos, modificar datos existentes, para eliminar datos y la más utilizada, para recuperar datos de la base de datos. Veremos que una sola instrucción de recuperación de datos es tan potente que permite recuperar datos de varias tablas a la vez, realizar cálculos sobre estos datos y obtener resúmenes.

El DML interactúa con el nivel externo de la base de datos por lo que sus instrucciones son muy parecidas, por no decir casi idénticas, de un sistema a otro, el usuario sólo indica lo que quiere recuperar no cómo se tiene que recuperar, no influye el cómo están almacenados los datos.

Es el lenguaje que utilizan los programadores y los usuarios de la base de datos.

● El **DCL** (Data Control Language) se compone de instrucciones que permiten:

- Ejercer un control sobre los datos tal como la asignación a usuarios de privilegios de acceso a los datos (GRANT/REVOKE).
- La gestión de transacciones (COMMIT/ROLLBACK).

Una transacción se puede definir como un conjunto de acciones que se tienen que realizar todas o ninguna para preservar la integridad de la base de datos.

Por ejemplo supongamos que tenemos una base de datos para las reservas de avión. Cuando un usuario pide reservar una plaza en un determinado vuelo, el sistema tiene que comprobar que queden plazas libres, si quedan plazas reservará la que quiera el usuario generando un nuevo billete y marcando la plaza como ocupada. Aquí tenemos un proceso que consta de dos operaciones de actualización de la base de datos (crear una nueva fila en la tabla de billetes, actualizar la plaza reservada en el vuelo, poniéndola como ocupada) estas dos operaciones se tienen que ejecutar o todas o ninguna, si después de crear el billete no se actualiza la plaza porque se cae el sistema, por ejemplo, la base de datos quedaría en un estado inconsistente ya que la plaza constaría como libre cuando realmente habría un billete emitido para esta plaza. En este caso el sistema tiene el mecanismo de transacciones para evitar este error. Las operaciones se incluyen las dos en una misma transacción y así el sistema sabe que las tiene que ejecutar las dos, si por lo que sea no se pueden ejecutar las dos, se encarga de deshacer los cambios que se hubiesen producido para no ejecutar ninguna.

Las instrucciones que gestionan las autorizaciones serán utilizadas normalmente por el administrador mientras que las otras, referentes a proceso de transacciones serán utilizadas también por los programadores.

No todos los sistemas disponen de ellas.

Algunos autores separan las instrucciones relativas a transacciones en el **TCL** (Transaction Control Language.

2 Introducción al TRANSACT-SQL.

Como hemos dicho, el sistema gestor de base de datos SQL-Server utiliza su propia versión de SQL, el TRANSACT_SQL.

TRANSACT-SQL es un lenguaje muy potente que nos permite definir casi cualquier tarea que queramos efectuar sobre la base de datos. A lo largo del curso veremos que TRANSACT-SQL incluye características propias de cualquier lenguaje de programación, características que nos permiten definir la lógica necesaria para el tratamiento de la información a través de procedimientos almacenados:

- Tipos de datos.
- Definición de variables.
- Estructuras de control de flujo.
- Gestión de excepciones.
- Funciones predefinidas.

Sin embargo no permite:

- Crear interfaces de usuario.
- Crear aplicaciones ejecutables, sino elementos que en algún momento llegarán al servidor de datos y serán ejecutados.

Debido a estas restricciones se emplea generalmente para crear procedimientos almacenados, triggers (procedimientos que se ejecutan automáticamente cuando se produce una operación de actualización sobre la base de datos) y funciones de usuario.

Puede ser utilizado como cualquier SQL como lenguaje embebido en aplicaciones desarrolladas en otros lenguajes de programación como Visual Basic, C, Java, etc. Y por supuesto los lenguajes incluidos en la plataforma .NET.

Puede ser ejecutado directamente de manera interactiva, por ejemplo desde el editor de consultas de SSMS (SQL Server Management Studio) el entorno de gestión que ya conocemos. Esta es la forma en que lo utilizaremos nosotros.

Como cualquier lenguaje de comunicación o cualquier lenguaje de programación, las instrucciones deben escribirse siguiendo una sintaxis determinada. Cuando cometemos errores de sintaxis al hablar puede que nuestro interlocutor nos entienda igualmente pero esto no pasa con el ordenador, debemos formar las frases siguiendo exactamente la sintaxis correcta si no queremos que la ejecución acabe en un error. Para saber cuál es la sintaxis correcta de las instrucciones SQL se utiliza un determinado tipo de formato descrito en el anexo I *Reglas de formato* que conviene conocer.

3 Características generales del lenguaje Transact-SQL

3.1 Patrón de las instrucciones

El lenguaje SQL se creó con la finalidad de ser un **lenguaje muy potente** y a la vez muy fácil de utilizar, se ha conseguido en gran medida ya que con una sola frase (instrucción) *STATEMENT/SENTENCE* podemos recuperar datos complejos (por ejemplo datos que se encuentran en varias tablas, combinándolos y calculando resúmenes), y utilizando un lenguaje **muy cercano al lenguaje hablado** (¡suponiendo que hablamos inglés, claro!).

Por ejemplo:

```
SELECT codigo, nombre FROM Clientes WHERE localidad='Valencia';
```

Esta instrucción nos permite SELECCIONAR el código y nombre DE los Clientes DONDE localidad sea igual a Valencia.

La sencillez también radica en que lo que indicamos es lo que queremos obtener, no el cómo lo tenemos que obtener, de eso se encargará el sistema automáticamente.

Las sentencias SQL además siguen todas el mismo patrón:

- Empiezan por un verbo que indica la acción a realizar,
- completado por el objeto sobre el cual queremos realizar la acción,
- seguido de una serie de cláusulas *CLAUSES* (unas obligatorias, otras opcionales) que completan la frase, y proporcionan más detalles acerca de lo que se quiere hacer.

Si sabemos algo de inglés nos será más fácil interpretar a la primera lo que quiere decir la instrucción, y de lo contrario, como el número de palabras que se emplean es muy reducido, enseguida nos las aprenderemos.

Por ejemplo en el DDL tenemos 3 verbos básicos:
CREATE (Crear)
DROP (Eliminar)
ALTER (Modificar)
Completados por el tipo de objeto sobre el que actúan y el objeto concreto:

```
CREATE DATABASE mibase .......;
```

Permite crear una base de datos llamada mibase, a continuación escribiremos las demás cláusulas que completarán la acción, en este caso dónde se almacenará la base de datos, cuánto ocupará, etc...

```
CREATE TABLE mitabla (.....);
```

Permite crear una nueva tabla llamada mitabla, entre paréntesis completaremos la acción indicando la definición de las columnas de la tabla.

```
CREATE INDEX miindex...;
```

Lo mismo para crear un índice (a que lo habías adivinado...).

```
DROP DATABASE  mibase;
```

Permite borrar, eliminar la base de datos mibase.

```
DROP TABLE mitabla;
```

Elimina la tabla mitabla

```
ALTER TABLE mitabla.....;
```

Permite modificar la definición de la tabla "mitabla".

Como se ve, todas estas instrucciones realizan acciones sobre la definición de la base de datos (estamos en el DDL).

En el DML utilizaremos los verbos:

INSERT	(crear, insertar una nueva fila de datos)
DELETE	(eliminar filas de datos)
UPDATE	(modificar filas de datos.
SELECT	(seleccionar, obtener...).

Estas acciones actúan sobre los datos almacenados.

Por ejemplo:

```
INSERT INTO mitabla .....
```

Inserta nuevas filas en mitabla

```
DELETE FROM mitabla
```

Eliminar filas de mitabla

```
UPDATE mitabla .......
```

Actualiza filas de mitabla, cambia los datos almacenados en mitabla.

Como ejemplo de cláusula dentro de una instrucción tenemos:

```
SELECT codigo, nombre
    FROM Clientes
    WHERE localidad='Valencia';
```

En esta sentencia nos aparecen dos cláusulas, la cláusula FROM que nos permite indicar de dónde hay que coger los datos y la cláusula WHERE que permite indicar una condición de selección.

Otra característica de una sentencia SQL es que acaba con un punto y coma (;) originalmente éste era obligatorio y servía para indicar el fin de la instrucción y que se ejecutara, pero ahora se puede omitir aunque se recomienda su uso. Cuando ejecutamos las instrucciones en modo consola es obligatorio.

En una sentencia utilizaremos palabras reservadas *RESERVED WORDS/KEYWORDS* (las fijas del lenguaje), y nombres de objetos y variables (identificadores *IDENTIFIERS*).
Las palabras reservadas no se pueden utilizar para otro propósito, por ejemplo una tabla no se puede llamar FROM, y los nombres (los identificadores) siguen las reglas detalladas en el punto siguiente. Escribiremos las palabras reservadas en mayúsculas y los identificadores en minúsculas, no es una regla obligatoria pero viene bien sobre todo al principio cuando estamos aprendiendo a escribir SQL.

3.2 Nombres cualificados.

En ocasiones deberemos utilizar nombres cualificados *QUALIFIED NAMES*, por ejemplo cuando se escribe un nombre de tabla, SQL presupone que se está refiriendo a una de las tablas de la base de datos activa, si queremos hacer referencia a una tabla de otra base de datos utilizamos su nombre cualificado *nombrebasedatos.nombredeesquema.nombretabla*, utilizamos el punto para separar el nombre del objeto y el nombre de su contenedor. O por ejemplo si en una consulta cuyo origen son dos tablas, queremos hacer referencia a un campo y ese nombre de campo es un nombre de campo en las dos tablas, pues utilizaremos su nombre cualificado *nombretabla.nombrecampo*.

El esquema *SCHEMA* en SQL Server es una estructura que permite organizar las tablas dentro de una base de datos, es equivalente a un esquema externo en la arquitectura de tres niveles que ya vimos. En SQL Server, por defecto se crea un esquema llamado dbo donde se crean todas las tablas de la base de datos si no se dice lo contrario. Como ejemplo, abre el SSMS y despliega la carpeta *Tablas* de la base de datos *GestionA* del tema anterior. Verás que las tablas se llaman dbo.empleados, dbo.oficinas, etc. Esto indica que todas pertenecen al esquema dbo.

3.3 El valor NULL.

Puesto que una base de datos es un modelo de una situación del mundo real, ciertos datos pueden inevitablemente faltar, ser desconocidos o no ser aplicables, esto se debe de indicar de alguna manera especial para no confundirlo con un valor conocido pero que sea cero por ejemplo. SQL tiene para tal efecto el valor NULL que indica precisamente la ausencia de valor.

Ejemplo: no es lo mismo que el alumno no tenga nota a que tenga la nota cero, esto afectaría también a todos los cálculos que se pueden realizar sobre la columna nota.

3.4 Reglas de formato de los identificadores

Los identificadores son los nombres de los objetos de la base de datos. Cualquier elemento de Microsoft SQL Server debe tener un identificador. Servidores, bases de datos, tablas, vistas, columnas, índices, desencadenadores, procedimientos, restricciones, reglas, etc. tienen identificadores.

Las reglas de formato de los identificadores normales dependen del nivel de compatibilidad de la base de datos, que se establecía con el parámetro sp_dbcmptlevel pero que ahora Microsoft aconseja no utilizar ya que desaparecerá en versiones posteriores en vez de eso se tiene que utilizar la cláusula SET COMPATIBILITY_LEVEL de la instrucción ALTER TABLE.

Cuando el nivel de compatibilidad es **90**, (el asignado por defecto) se aplican las reglas siguientes para los nombres de los identificadores:

- No puede ser una palabra reservada.
- El nombre debe tener entre 1 y 128 caracteres, excepto para algunos tipos de objetos en los que el número es más limitado.
- El nombre debe empezar por:
  - Una letra, como aparece definida por el estándar Unicode 3.2. La definición Unicode de letras incluye los caracteres latinos de la "a" a la "z" y de la "A" a la "Z".
  - El carácter de subrayado (_), arroba (@) o número (#).
- Ciertos símbolos al principio de un identificador tienen un significado especial en SQL Server. Un identificador que empieza con el signo de arroba indica un (#) parámetro o una variable local. Un identificador que empieza con el signo de número indica una tabla o procedimiento temporal. Un identificador que empieza con un signo de número doble (##) indica un objeto temporal global.
- Algunas funciones de Transact-SQL tienen nombres que empiezan con un doble signo de arroba (@@). Para evitar confusiones con estas funciones, se recomienda no utilizar nombres que empiecen con @@.
- No se permiten los caracteres especiales o los espacios incrustados.

Si queremos utilizar un nombre que no siga estas reglas, normalmente para poder incluir espacios en blanco, lo tenemos que escribir encerrado entre corchetes [ ] (también se pueden utilizar las comillas pero recomendamos utilizar los corchetes).

3.5 Tipos de datos

En SQL Server, cada columna, expresión, variable y parámetro está asociado a un tipo de datos.

Un tipo de datos, realmente define el conjunto de valores válidos para los campos definidos de ese tipo. Indica si el campo puede contener: datos numéricos, de caracteres, moneda, fecha y hora, etc.

SQL Server proporciona un conjunto de tipos de datos del sistema que define todos los tipos de datos que pueden utilizarse. También podemos definir nuestros propios tipos de datos en Transact-SQL o Microsoft .NET Framework.

Los tipos de datos de SQL Server 2005 se organizan en las siguientes categorías:

Numéricos exactos

bigint	decimal prederminado dec(18,0)
int	numeric
smallint	money
tinyint	smallmoney
bit	

Numéricos aproximados

float	real

Fecha y hora

datetime	smalldatetime AAAA-MM-DD hh:mm:ss
date a partir de SQL Server 2008	time a partir de SQL Server 2008
datetime2 a partir de SQLServer 2008	datetimeoffset a partir de SQL Server 2008

Cadenas de caracteres

char predeterminado char(1)	text desuso
varchar	

Cadenas de caracteres Unicode

nchar predeterminado nchar(1)	ntext desuso
nvarchar	

Cadenas binarias

binary predeterminado binary(1)	image desuso
varbinary	

Otros tipos de datos

cursor	timestamp (rowversion en 2008)
sql_variant	uniqueidentifier
table	xml

Todos los datos almacenados en Microsoft SQL Server 2005 deben ser compatibles con uno de estos tipos de datos básicos. El tipo de datos **cursor** y el tipo table no se pueden asignar a una columna de una tabla. Sólo se puede utilizar con variables y parámetros de procedimientos almacenados.

Para más información sobre tipos de datos puedes visitar la página:
http://technet.microsoft.com/es-es/library/ms187752(SQL.90).aspx Versión SQL Server 2005
http://technet.microsoft.com/en-us/library/ms187752(SQL.90).aspx Data types 2005
http://technet.microsoft.com/en-us/library/ms187752(v=sql.110).aspx 2012
http://sqlhints.com/2014/01/26/variables-and-data-types-in-sql-server/

3.6 Las constantes

Una constante *CONSTANT* es un valor específico o un símbolo que representa un valor de dato específico.

El formato de las constantes depende del tipo de datos del valor que representan.

Las **constantes de cadena de caracteres** van entre comillas simples e incluyen caracteres alfanuméricos (a-z, A-Z y 0-9) y caracteres especiales, como el signo de exclamación (!), la arroba (@) y el signo de número (#).
Si la opción QUOTED_IDENTIFIER se ha establecido en OFF para una conexión, las cadenas de caracteres también se pueden incluir entre comillas dobles, pero el proveedor de Microsoft SQL Native Client y el controlador ODBC utilizan automáticamente SET QUOTED_IDENTIFIER ON. Se recomienda el uso de comillas simples.

Si una cadena de caracteres entre comillas simples contiene una comilla, la comilla interna se representa con dos comillas simples. Esto no es necesario en las cadenas incluidas entre comillas dobles.

Por ejemplo:

'Juan García López' representa el valor *Juan García López*
'Escribir "Continuar"' representa el valor *Escribir 'Continuar'*

Las cadenas vacías se representan como dos comillas simples sin nada entre ellas. En el modo de compatibilidad 6.x, una cadena vacía se trata como un espacio.

Las constantes de cadena de caracteres admiten intercalaciones *COLLATION* mejoradas. Una intercalación define entre otras cosas el alfabeto que se va a utilizar, si se distingue entre mayúsculas y minúsculas o entre letras acentuadas o no.

(cláusula COLLATE ver
http://technet.microsoft.com/es-es/library/ms179886.aspx
http://technet.microsoft.com/en-us/library/ms179886.aspx).

Las **cadenas Unicode** tienen un formato similar al de las cadenas de caracteres, pero están precedidas por el identificador N (N es el idioma nacional en el estándar SQL-92). El prefijo N tiene que estar en mayúsculas. Por ejemplo, 'Miguel' es una constante de caracteres, mientras que N'Miguel' es una constante Unicode.

Las constantes Unicode se interpretan como datos Unicode. Los datos Unicode se almacenan con 2 bytes por carácter en lugar de 1 byte por carácter, como los datos de cadenas de caracteres.

Las constantes de cadena Unicode aceptan intercalaciones mejoradas.

Sin el prefijo **N**, la cadena se convierte a la página de códigos predeterminada de la bd.

Las constantes **binarias** tienen el prefijo 0x y son cadenas de números hexadecimales. No se incluyen entre comillas.

Por ejemplo:
0x1F
0xA345

Las constantes de tipo **bit** se representan con los números 0 ó 1, y no se incluyen entre comillas. Si se utiliza un número mayor que uno, se convierte en uno.

Las constantes de tipo **datetime** se representan mediante valores de fecha en formatos específicos de cadenas de caracteres, incluidos entre comillas simples.

Ejemplos de constantes **datetime**:

'Marzo 10, 1990'
'10 Marzo, 1990'
'900310'
'03/10/90'

Ejemplos de constantes de hora:
'14:30:24'
'04:24 PM'

Las constantes de tipo **integer** se representan mediante una cadena de números sin comillas y sin separadores decimales.

Las constantes de tipo **decimal** se representan mediante una cadena de números sin comillas y con el punto como separador decimal.

Ejemplos:
1894.1204
2.0

Las constantes de tipo **float** y **real** se representan en notación científica.

Ejemplos:
101.5E5
0.5E-2

Las constantes de tipo **money** se representan como el tipo decimal y un símbolo de moneda opcional como prefijo.

Por ejemplo:
€12
€542023.14

Las constantes de tipo **uniqueidentifier** son una cadena que representa un GUID (Global Unique Identifier). Son números que se generan de forma automática y cuya probabilidad que se repitan es muy pequeña. Se pueden especificar en formato de cadena de caracteres o binario.

Ejemplos:
'6F9619FF-8B86-D011-B42D-00C04FC964FF'
0xff19966f868b11d0b42d00c04fc964ff

Para indicar **valores negativos y positivos** añadimos el prefijo **+** o **-** según sea el valor positivo o negativo. Sin prefijo se entiende que el valor es positivo.
More information:
http://technet.microsoft.com/en-us/library/ms179899(SQL.90).aspx Constants

3.7 Las expresiones

Una expresión es una combinación de símbolos y operadores que el Motor de base de datos *DATABASE ENGINE* de SQL Server evalúa para obtener un único valor. Una expresión simple puede ser una sola constante, variable, columna o función escalar. Los operadores se pueden usar para combinar dos o más expresiones simples y formar una expresión compleja.

Dos expresiones **pueden combinarse** mediante un operador si ambas tienen tipos de datos admitidos por el operador y se cumple al menos una de estas condiciones:

- Las expresiones tienen el mismo tipo de datos.
- El tipo de datos de menor prioridad se puede convertir implícitamente al tipo de datos de mayor prioridad.
- La función CAST puede convertir explícitamente el tipo de datos con menor prioridad al tipo de datos con mayor prioridad o a un tipo de datos intermedio que pueda convertirse implícitamente al tipo de datos con la mayor prioridad.

Operadores numéricos:
+ (suma), - (resta), * (multiplicación), / (división), % (módulo, resto de una división).

Operadores bit a bit: realizan manipulaciones de bits entre dos expresiones de cualquiera de los tipos de datos de la categoría del tipo de datos entero.
& (AND bit a bit), | (OR bit a bit), ^ (OR exclusivo bit a bit).

Operadores de comparación:
= (Igual a), > (Mayor que), < (Menor que), >= (Mayor o igual que), <= (Menor o igual que), <> (Distinto de), != (No es igual a), !< (No menor que), !> (No mayor que)

Operadores lógicos:
Aquí sólo los nombraremos ya que en el tema de consultas simples los veremos en detalle.
ALL, AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR, SOME.

Operadores de cadenas:
+ (concatenación)

Resultados de la expresión

Si se combinan dos expresiones mediante operadores de comparación o lógicos, el tipo de datos resultante es booleano y el valor es uno de los siguientes: TRUE, FALSE o UNKNOWN.
Cuando dos expresiones se combinan mediante operadores aritméticos, bit a bit o de cadena, el operador determina el tipo de datos resultante.

Las expresiones complejas formadas por varios símbolos y operadores se evalúan como un resultado formado por un solo valor. El tipo de datos, intercalación, precisión y valor de la expresión resultante se determina al combinar las expresiones componentes de dos en dos, hasta que se alcanza un resultado final. La prioridad de los operadores de la expresión define la secuencia en que se combinan las expresiones.

Para más información:
http://technet.microsoft.com/es-es/library/ms190286.aspx SQL Server 2008
http://technet.microsoft.com/es-es/library/ms190286(SQL.90).aspx SQL Server 2005
http://technet.microsoft.com/en-us/library/ms190286(SQL.90).aspx Expressions 2005
http://technet.microsoft.com/en-us/library/ms190286(v=sql.110).aspx Expressions 2012

3.8 Funciones

SQL Server 2005 proporciona numerosas funciones integradas *BUILD-IN FUNCTIONS* y permite crear funciones definidas por el usuario *USER-DEFINED FUNCTIONS*.

Existen diferentes tipos de funciones

Funciones de conjuntos de filas *ROWSET FUNCTIONS*: devuelven un objeto que se puede utilizar, en instrucciones Transact-SQL, en lugar de una referencia a una tabla.

Funciones de agregado *AGGREGATE FUNCTIONS* (también llamadas funciones de columna): Operan sobre una colección de valores y devuelven un solo valor de resumen.

Funciones de categoría *RANKING FUNCTIONS*: Devuelven un valor de categoría para cada fila de un conjunto de filas, por ejemplo devuelve el número de la fila, el ranking de la fila en una determinada ordenación, etc.

Funciones escalares *SCALAR FUNCTIONS*: Operan sobre un valor y después devuelven otro valor. Son las funciones que estamos acostumbrados a utilizar.

Las funciones escalares se clasifican según el tipo de datos de sus operandos:

Funciones escalares	Descripción
Funciones de configuración	Devuelven información acerca de la configuración actual.
Funciones de cursor	Devuelven información acerca de los cursores.
Funciones de fecha y hora	Llevan a cabo operaciones sobre valores de entrada de fecha y hora.
Funciones matemáticas	Realizan cálculos basados en valores de entrada y devuelven valores numéricos.
Funciones de metadatos	Devuelven información acerca de la base de datos y los objetos de la base de datos.
Funciones de seguridad	Devuelven información acerca de usuarios y funciones.
Funciones de cadena	Realizan operaciones en el valor de entrada de una cadena (char o varchar) y devuelven una cadena o un valor numérico.
Funciones del sistema	Realizan operaciones y devuelven información acerca de valores, objetos y configuraciones de una instancia de SQL Server.
Funciones estadísticas del sistema	Devuelven información estadística acerca del sistema.
Funciones de texto e imagen	Realizan operaciones sobre los valores de entrada o columnas de texto o imagen, y devuelven información acerca del valor.

Para obtener más información sobre funciones:
http://technet.microsoft.com/es-es/library/ms174318.aspx Para SQL Server 2008.
http://technet.microsoft.com/es-es/library/ms174318(SQL.90).aspx Para SQL Server 2005.
http://technet.microsoft.com/en-us/library/ms174318(SQL.90).aspx Functions in 2005
http://technet.microsoft.com/en-us/library/ms174318(v=sql.110).aspx Functions in 2012
https://msdn.microsoft.com/es-es/library/ms174318.aspx Functions in 2014

3.9 Otros elementos del lenguaje

Como hemos comentado, SQL incluye características propias de cualquier lenguaje de programación, las comentaremos por encima en este punto y volveremos sobre ellas en los temas de programación.

Estas son las principales:

- Variables. En Transact-SQL podemos definir variables *VARIABLES*, que serán de un tipo de datos determinado, como tipos de datos podemos utilizar los propios de la base de datos SQL-SERVER, pero también podemos utilizar tipos propios del lenguaje que no pueden ser utilizados en DDL.

- Instrucciones de Control de flujo. Las instrucciones de control de flujo *CONTROL-OF-FLOW* permiten definir bucles, saltos condicionales o no a una determinada etiqueta *LABEL*, ejecución condicional o no de determinadas líneas de código, etc. Por ejemplo BEGIN...END, RETURN, TRY... CATCH, IF… ELSE, WHILE, BREAK, CONTINUE, WAITFOR, GOTO y EXECUTE.

- Instrucciones para el control de errores. Son instrucciones y funciones que permiten llevar un control de los errores *ERROR-HANDLING/EXCEPTION-HANDLING* producidos en tiempo de ejecución. Por ejemplo TRY... CATCH, RAISERROR, ERROR_LINE(), ERROR_MESSAGE(), ERROR_NUMBER(), ERROR_PROCEDURE(), ERROR_SEVERITY(), ERROR_STATE().
- Mensajes. Tenemos también instrucciones para enviar mensajes *MESSAGE* al usuario como PRINT y SELECT.
- Comentarios. Con los símbolos /* y */ podemos encerrar varias líneas de comentarios *COMMENTS*. Con –- lo que escribamos después y hasta el final de la línea será considerado comentario y no se tendrá en cuenta en la ejecución.
- USE nombre_Basededatos Cambia el contexto de la base de datos al de la base de datos especificada, las instrucciones que se ejecuten a partir de ese momento se ejecutarán sobre la base de datos especificada.
- GO. No es una instrucción Transact-SQL, sino un comando reconocido por las utilidades **sqlcmd** y **osql**, así como por el Editor de código de SQL Server Management Studio. Es interpretado como una señal de que se debe enviar el lote actual de instrucciones Transact-SQL a una instancia de SQL Server para su ejecución. El lote actual de instrucciones está formado por todas las instrucciones específicadas desde el último comando GO o desde el comienzo de la sesión o script si se trata del primer comando GO. Por ejemplo si queremos crear una consulta para crear una base de datos y sus tablas, después del CREATE DATABASE…; tenemos que poner GO antes del primer CREATE TABLE para que el sistema efectúe la primera operación (la creación de la base de datos).

4 DML. Las consultas simples

Una vez visto las características generales del lenguaje Transact-SQL, estamos en disposición de abordar las instrucciones que lo forman. Empezaremos por el DML para aprender a manejar los datos almacenados en las tablas de la base de datos, luego estudiaremos el DDL para definir la estructura de la base de datos y finalmente aprenderemos a programar procedimientos en Transact-SQL.

Vamos a empezar pues por la instrucción que más se utiliza en SQL, la sentencia SELECT. La sentencia SELECT es, con diferencia, la más compleja y potente de las sentencias SQL, con ella podemos recuperar datos de una o más tablas, seleccionar ciertos registros e incluso obtener resúmenes de los datos almacenados en la base de datos. Es tan compleja que la estudiaremos a lo largo de varias unidades didácticas incorporando poco a poco nuevas funcionalidades.

El resultado de una SELECT es una tabla lógica (no se almacena físicamente en la base de datos sino que se encuentra en memoria) que alberga las filas resultantes de la ejecución de la sentencia.

La sintaxis completa es la siguiente:

```
SELECT sentencia::=[WITH <expresion_tabla_comun> [,...n]]
    <expresion_consulta>
    [ORDER BY {expression_columna|posicion_columna [ASC|DESC] }
  [ ,...n ]]
    [COMPUTE
   {{AVG|COUNT|MAX|MIN|SUM} (expression)}[ ,...n ] [BY expression[ ,...n ]]
    ]
    [<FOR clausula_for>]
    [OPTION (<query_hint>[ ,...n ])]

<expresion_consulta> ::=
    {<especificacion_consulta> | ( < expresion_consulta > ) }
    [ {UNION [ALL]|EXCEPT|INTERSECT}
        <especificacion_consulta> | (<expresion_consulta>) [...n ]
    ]

<especificacion_consulta> ::=
SELECT [ALL|DISTINCT]
    [TOP expresion [PERCENT] [WITH TIES] ]
    <lista_seleccion>
    [INTO nueva_tabla]
    [FROM { <origen> } [ ,...n ] ]
    [WHERE <condicion_busqueda> ]
    [GROUP BY [ ALL ] expresion_agrupacion [ ,...n ]
    [WITH { CUBE | ROLLUP } ]
    ]
    [HAVING < condicion_busqueda > ]
```

Debido a la complejidad de la sentencia (y en la sintaxis anterior no se han detallado algunos elementos), la iremos viendo poco a poco, empezaremos por ver consultas básicas para luego ir añadiendo más cláusulas.

Empezaremos por ver las consultas más simples, basadas en una sola tabla y nos limitaremos a la siguiente sintaxis:

```
SELECT [ALL|DISTINCT]
    [TOP expresion [PERCENT] [WITH TIES]]
    <lista_seleccion>
    FROM <origen>
    [WHERE <condicion_busqueda> ]
    [ORDER BY {{expression_columna|posicion_columna} [ASC|DESC]} [ ,...n ]]
```

5 Origen de datos FROM

De la sintaxis anterior, el elemento <origen> indica de dónde se va a extraer la información y se indica en la cláusula FROM, es la única cláusula obligatoria.

En este tema veremos un origen de datos basado en una sola tabla.

La sintaxis será la siguiente:

```
<origen>::=
nb_tabla | nb_vista [[ AS ] alias_tabla ]
```

nb_tabla representa un nombre de tabla

nb_vista un nombre de vista

Tanto para las tablas como para las vistas, podemos hacer referencia a tablas que están en otras bases de datos (siempre que tengamos los permisos adecuados), en este caso tenemos que cualificar el nombre de la tabla, indicando delante el nombre de la base de datos (Lógica) y el nombre del esquema al que pertenece la tabla dentro de la base de datos, por ejemplo:

MiBase.dbo.MiTabla se refiere a la tabla *MiTabla* que se encuentra en el esquema *dbo* de la base de datos *MiBase*.

Cuando no se definen esquemas, SQL-Server crea uno por defecto en cada base de datos denominado *dbo*.

Opcionalmente podemos definir un **nombre de alias**.

Un nombre de alias (*alias_tabla*) es un nombre alternativo que se le da a la tabla dentro de la consulta.

Si se define un nombre de alias, dentro de la consulta, será el nombre a utilizar para referirnos a la tabla, el nombre original de la tabla ya no tendrá validez dentro de la consulta.

Se utilizan los nombres de alias para simplificar los nombres de tablas a veces largos y también cuando queremos combinar una tabla consigo misma; ya volveremos sobre los alias de tabla cuando veamos consultas multitabla.

La palabra AS no añade ninguna operatividad, está más por estética.

Podemos escribir:

```
SELECT ...
FROM tabla1       Sacamos los datos de la tabla tabla1
```

```
SELECT ...
FROM tabla1 t1  Sacamos los datos de la tabla tabla1 y le asignamos un alias
de tabla: t1
```

```
SELECT ...
FROM tabla1 AS t1  Es equivalente a la sentencia anterior.
```

Si la tabla o la vista están en otra base de datos del mismo equipo que está ejecutando la instancia de SQL Server, se utiliza el nombre cualificado con el formato nbBaseDatos*.nbEsquema.nbTabla*.

Si la tabla o la vista están fuera del servidor local en un servidor vinculado, se utiliza un nombre de cuatro partes con el formato *nbservidor*.*catalogo*.nbE*squema*.nbTabla.

6 La lista de selección

En la lista de selección <lista_seleccion> indicamos las **columnas** que queremos que **aparezcan en el resultado** de la consulta.

```
<lista_seleccion> ::=
 {  *
    | {nombre_tabla|nombre_vista|alias_tabla}.*
    | { [{nombre_tabla|nombre_vista|alias_tabla}.]
        {nb_columna|$IDENTITY|$ROWGUID}
        |<expresion>
      }[[AS] alias_columna]
    | alias_columna = <expresion>
 } [ ,...n ]
```

Separamos la definición de cada columna por una coma y las columnas del resultado aparecerán en el mismo orden que en la lista de selección.

Para cada columna del resultado su tipo de datos, tamaño, precisión y escala son los mismos que los de la expresión que da origen a esa columna.

Podemos definir las columnas del resultado de varias formas, mediante:

- una expresión simple:
  - una referencia a una función.
  - una variable local
  - una constante
  - una columna del origen de datos,

- una subconsulta escalar, que es otra instrucción SELECT que devuelve un único valor y se evalúa para cada fila del origen de datos (esto no lo veremos de momento),
- una expresión compleja generada al usar operadores en una o más expresiones simples.
- la palabra clave *.
- la asignación de variables con el formato *@variable_local* = expresión.
- la palabra clave $IDENTITY.
- la palabra clave $ROWGUID.

6.1 Columnas del origen de datos

Cuando queremos indicar en la lista de selección una columna del origen de datos, la especificamos mediante su nombre simple o nombre cualificado *QUALIFIED NAME*.

El nombre cualificado consiste en el nombre de la columna precedido del nombre de la tabla donde se encuentra la columna. Si en el origen de datos hemos utilizado una vista o un nombre de alias, deberemos utilizar ese nombre. Es obligatorio utilizar el nombre cualificado cuando el nombre de la columna aparece en más de una tabla del origen de datos.

Ejemplo de consulta simple:
Listar nombres, oficinas, y fechas de contrato de todos los empleados:

```
SELECT nombre, oficina, contrato
FROM empleados;
```

El resultado sería:

nombre	oficina	contrato .
Antonio Viguer	12	1986-10-20
Alvaro Jaumes	21	1986-12-10
Juan Rovira	12	1987-03-01
José González	12	1987-05-19
Vicente Pantalla	13	1988-02-12
Luis Antonio	11	1988-06-14
Jorge Gutiérrez	22	1988-11-14
Ana Bustamante	21	1989-10-12
María Sunta	11	1999-10-12
Juan Victor	NULL	1990-01-13

Listar una tarifa de productos:

```
SELECT idfab, idproducto, descripcion, productos.precio
FROM productos;
```

Hemos cualificado la columna precio aunque no es necesario en este caso.

El resultado sería:

Idfab	idproducto	descripcion	precio
aci	41001	arandela	0,58
aci	41002	bisagra	0,80
aci	41003	art t3	1,12
aci	41004	art t4	1,23
aci	4100x	junta	0,26
aci	4100y	extractor	28,88
aci	4100z	mont	26,25
bic	41003	manivela	6,52
bic	41089	rodamiento	2,25

6.2 Alias de columna

Por defecto, en el encabezado de cada columna del resultado, aparece el nombre de la columna origen, pero esto se puede cambiar definiendo un alias de columna, el alias de columna es un nombre alternativo que se le da a esa columna.

El alias de columna se indica mediante la cláusula AS. Se escribe el nuevo texto tal cual sin comillas siguiendo las reglas de los identificadores.

Ejemplo:

```
SELECT numclie,nombre AS nombrecliente
FROM clientes;
```

El resultado será :

Numclie	nombrecliente
2101	Luis García Antón
2102	Alvaro Rodríguez
2103	Jaime Llorens

en vez de:

numclie	nombre
2101	Luis García Antón
2102	Alvaro Rodríguez
2103	Jaime Llorens

La palabra AS es opcional.

```
SELECT numclie,nombre nombrecliente
FROM clientes;
```

Sería equivalente a la consulta anterior.

Si queremos incluir espacios en blanco en el nombre lo debemos encerrar entre corchetes.

```
SELECT numclie,nombre AS [nombre cliente]
FROM clientes;
```

Nota importante: Este nombre de alias se podrá utilizar en la lista de selección y en la cláusula ORDER BY pero no en la cláusula WHERE.

6.3 Columnas calculadas

Además de las columnas que provienen directamente de la tabla origen, una consulta SQL puede incluir columnas calculadas cuyos valores se evalúan a partir de una expresión.

La expresión puede contener cualquier operador válido (+, -, *, /, &…), cualquier función válida, nombres de columnas del origen de datos, nombres de parámetros o constantes y para combinar varias operaciones se pueden utilizar los paréntesis.

Ejemplos:
Listar la ciudad, región y el superávit de cada oficina. Consideraremos el superávit como el volumen de ventas que se encuentran por encima o por debajo del objetivo de la oficina.

```
SELECT ciudad, region, (ventas-objetivo) AS superavit
FROM oficinas;
```

El resultado será:

ciudad	region	superavit
Valencia	este	11800,00
Alicante	este	-6500,00
Castellon	este	1800,00
Badajoz	oeste	11100,00
A Coruña	oeste	-11400,00
Madrid	centro	NULL
Madrid	centro	-10000,00
Pamplona	norte	NULL
Valencia	este	-90000,00

De cada producto queremos saber el id de fabricante, id de producto, su descripción y el valor de sus existencias.

```
SELECT idfab,idproducto,descripcion,(existencias*precio) AS valoracion
FROM productos;
```

El resultado será:

idfab	idproducto	descripción	valoración
aci	41001	arandela	160,66
aci	41002	bisagra	133,60
aci	41003	art t3	231,84
aci	41004	art t4	170,97
aci	4100x	junta	9,62
aci	4100y	extractor	722,00
aci	4100z	mont	735,00
bic	41003	manivela	19,56
bic	41089	rodamiento	175,50

Listar el nombre, mes y año del contrato de cada vendedor.

```
SELECT nombre, MONTH(contrato) AS [Mes de contrato], YEAR(contrato) AS [Año
de contrato]
FROM  empleados;
```

El resultado será:

Nombre	Mes de contrato	Año de contrato
Antonio Viguer	10	1986
Alvaro Jaumes	12	1986
Juan Rovira	3	1987

Listar las ventas en cada oficina con el formato: 22 tiene ventas de 186,042.00 €

```
SELECT  oficina, 'tiene ventas de ' AS [ ], ventas
FROM oficinas;
```

El resultado será:

oficina		ventas
11	tiene ventas de	69300,00
12	tiene ventas de	73500,00
13	tiene ventas de	36800,00
21	tiene ventas de	83600,00
22	tiene ventas de	18600,00
23	tiene ventas de	NULL
24	tiene ventas de	15000,00
26	tiene ventas de	NULL
28	tiene ventas de	0,00

El incluir una constante como columna en la lista de selección puede parecer inútil (se repetirá el mismo valor en todas las filas) pero veremos más adelante que tiene utilidad en ciertos casos.

También podemos utilizar la sintaxis:

```
alias_columna = <expresion>
```

Ejemplo:

```
SELECT oficina, superavit = ventas-objetivo
```

Esto tiene el mismo efecto que

```
SELECT oficina, ventas-objetivo AS superavit
```

En cuanto a la utilización de funciones hay que recordar que una función representa un valor (el valor devuelto por la función), siempre se invoca por su nombre seguido de paréntesis dentro de los cuales indicamos los parámetros de entrada de la función (sobre qué valores se va a realizar el cálculo), en ocasiones la función no lleva parámetros en tal caso se escriben los paréntesis vacíos.

En un ejemplo anterior se ha utilizado la función MONTH(), devuelve el nº del mes de una determina fecha, la fecha la indicamos como parámetro. MONTH(contrato) representa el mes (en nº) de la fecha de contrato del empleado.

GETDATE() devuelve la fecha del sistema, en este caso no indicamos ningún parámetro.

Para más información sobre las funciones de Transact-SQL consulta el anexo sobre funciones. No se trata de conocer todas las funciones al pie de la letra sino saber que existen, saber buscarlas en la ayuda del lenguaje y luego aplicar su sintaxis.

Las funciones suelen clasificarse por tipo, funciones de fechas, de cadena, numéricas, etc. Cuando queremos obtener un resultado calculado a partir de la información almacenada en una columna, buscamos si existe una función que realice ese cálculo, el nombre de la función muchas veces nos indica lo que hace, una vez localizada la función es interpretar la sintaxis y aplicarla.

Las funciones pueden variar de un SGBD a otro, por ejemplo en Transact SQL para obtener la fecha del sistema utilizamos GETDATE(), en MySql la función es NOW(). Aunque tengamos variaciones de un sistema a otro, tenemos más o menos las mismas funciones, lo que puede cambiar es el nombre de la función y en algún caso su sintaxis, pero lo importante es saber utilizarla y cuándo.

6.4 Utilización del *

Si queremos visualizar todas las columnas del origen de datos, en lugar de indicar todas las columnas una a una se puede utilizar el carácter de sustitución *.

Mostrar todos los datos de la tabla oficinas.

```
SELECT *
FROM oficinas;
```

Obtener todos los datos y el superávit de cada oficina.

```
SELECT *, (ventas-objetivo) AS superavit
FROM oficinas;
```

También podemos cualificar el * con un nombre de tabla, de vista o un alias de tabla:

```
SELECT oficinas.*
FROM oficinas;
```

*oficinas.** se interpreta como: *todas las columnas de la tabla oficinas*.

Esta forma se utiliza normalmente cuando el origen está basado en varias tablas y queremos indicar todas las columnas no del origen completo sino de una tabla concreta.

6.5 La palabra clave $IDENTITY

La palabra clave $IDENTITY se interpreta como la columna de la tabla que tiene la propiedad IDENTITY (la columna de identidad que vimos en un tema anterior).

Por ejemplo, si en la columna codigo de la tabla usuarios (BD Biblio) se ha definido la propiedad IDENTITY,

```
SELECT $IDENTITY, nombre, apellidos
FROM usuarios;
```

Es equivalente a:

```
SELECT codigo, nombre, apellidos
FROM usuarios;
```

6.6 La palabra clave $ROWGUID.

La palabra clave $ROWGUID se interpreta como la columna de la tabla que tiene la propiedad ROWGUIDCOL y se puede utilizar de la misma forma que $IDENTITY.

https://msdn.microsoft.com/en-us/library/ms176104(v=sql.130).aspx SELECT Clause

7 Ordenación de las filas del resultado ORDER BY

Si queremos que las filas del resultado de la consulta aparezcan ordenadas, lo podemos indicar mediante la cláusula ORDER BY.

```
ORDER BY {{expression_columna|posicion_columna}[ASC|DESC]}[ ,...n ]
```

Podemos indicar una columna o varias separadas por una coma, la columna de ordenación se especifica mediante el nombre de columna en el origen de datos o su posición dentro de la lista de selección. Si utilizamos el nombre de columna, no hace falta que la columna aparezca en la lista de selección. Si utilizamos la posición es la posición de la columna dentro de la lista de selección empezando en 1.

Por defecto la filas se ordenarán en modo ascendente (ASC), de menor a mayor; si queremos invertir ese orden añadimos detrás de la columna la palabra DESC.

Si la columna de ordenación es alfanumérica, las filas se ordenarán por orden alfabético.
Si la columna de ordenación es numérica, las filas se ordenarán de menor a mayor.
Si la columna de ordenación es de tipo fecha, las filas se ordenarán de más antigua a más reciente o futura.

Ejemplos:
Mostrar las ventas de cada oficina, ordenadas por orden alfabético de región y dentro de cada región por ciudad.

```
SELECT oficina, region, ciudad, ventas
FROM oficinas
ORDER BY region, ciudad;
```

Da como resultado:

Oficina	region	ciudad	ventas
24	centro	Aranjuez	15000,00
23	centro	Madrid	NULL
12	este	Alicante	73500,00
13	este	Castellón	36800,00
11	este	Valencia	69300,00
28	este	Valencia	0,00
26	norte	Pamplona	NULL
22	oeste	A Coruña	18600,00
21	oeste	Badajoz	83600,00

Listar las oficinas de manera que las oficinas de mayores ventas aparezcan en primer lugar.

```
SELECT ciudad, region, ventas
FROM oficinas
ORDER BY ventas DESC;
```

Ciudad	region	ventas
Badajoz	oeste	83600,00
Alicante	este	73500,00
Valencia	este	69300,00
Castellón	este	36800,00
A Coruña	oeste	18600,00
Aranjuez	centro	15000,00
Valencia	este	0,00
Pamplona	norte	NULL
Madrid	centro	NULL

Listar las oficinas clasificadas en orden descendente de rendimiento de ventas, de modo que las de mayor rendimiento aparezcan las primeras.

```
SELECT ciudad, region, ventas-objetivo
FROM oficinas
ORDER BY 3 DESC;
```

Lo mismo que el anterior pero agrupadas por región.

```
SELECT region, ciudad, (ventas-objetivo) AS superavit
FROM oficinas
ORDER BY region, superavit DESC;
```

Resultado:

Region	ciudad	superavit
centro	Aranjuez	-10000,00
centro	Madrid	NULL
este	Valencia	11800,00
este	Castellón	1800,00
este	Alicante	-6500,00
este	Valencia	-90000,00
norte	Pamplona	NULL
oeste	Badajoz	11100,00
oeste	A Coruña	-11400,00

En este caso hemos utilizado el alias de columna para hacer referencia a la columna calculada y también se puede observar que las filas aparecen ordenadas por región ascendente (no hemos incluido nada después del nombre de la columna) y dentro de cada región por superávit y descendente.

https://msdn.microsoft.com/en-us/library/ms188385(v=sql.130).aspx ORDER BY Clause

8 La cláusula TOP

```
    [TOP <expresión> [PERCENT] [WITH TIES]]
```

La cláusula TOP indica que en el resultado no deben aparecer todas las filas resultantes sino un cierto número de registros, las n primeras. Si la consulta incluye la cláusula ORDER BY, se realiza la ordenación antes de extraer los n primeros registros.

La expresión representa ese número n y debe devolver un número entero sin signo.

Por ejemplo tenemos la siguiente tabla:

	cod	ventas
1	1	100,00
2	2	200,00
3	3	200,00
4	4	10000,00
5	5	10000,00
6	6	10,00
7	7	250,00

```
SELECT * FROM productos:
```

Si ordenamos por ventas:

```
SELECT * FROM productos
ORDER BY ventas;
```

Obtenemos el siguiente resultado:

	cod	ventas
1	6	10,00
2	1	100,00
3	2	200,00
4	3	200,00
5	7	250,00
6	4	10000,00
7	5	10000,00

Si añadimos la cláusula TOP:

```
SELECT TOP 3 * FROM productos
ORDER BY ventas;
```

	cod	ventas
1	6	10,00
2	1	100,00
3	3	200,00

Obtenemos los 3 primeros registros:

Si existen más registros con las mismas ventas que el último valor de la lista, éstos no saldrán en el resultado de la consulta.

En el ejemplo el registro con cod = 2 no sale en el resultado y tiene las mismas ventas que cod = 3.

Si queremos que salgan añadimos la cláusula WITH TIES. La cláusula WITH TIES sólo se puede emplear si la SELECT incluye un ORDER BY, de lo contrario dará error.

Si añadimos la cláusula WITH TIES:

```
SELECT TOP 3 WITH TIES *
FROM productos
ORDER BY ventas
```

Obtenemos:

	cod	venta
1	6	10,00
2	1	100,0
3	2	200,0
4	3	200,0

Se incluyen en el resultado todos los registros que tienen ventas iguales al último registro.

Otro ejemplo:

```
SELECT TOP 10 oficina, ciudad, ventas
FROM oficinas
ORDER BY ventas;
```

Devuelve las 10 peores oficinas en cuanto a ventas: ordenamos las oficinas por ventas de menor a mayor y sacamos las 10 primeras.

Si incluimos la palabra PERCENT, entonces *n* no indica el número de registros a recuperar sino el porcentaje de registros a recuperar del total de filas recuperadas después de ejecutar la cláusula WHERE.

```
SELECT TOP 50 PERCENT  *
FROM productos
ORDER BY ventas
```

Devuelve:

	cod	ventas
1	6	10,00
2	1	100,00
3	2	200,00
4	3	200,00

Si el porcentaje no da exacto, siempre redondea al alza.

Esta cláusula puede variar de un sistema a otro aquí un ejemplo, la misma instrucción en tres sistemas diferentes:

SQL Server
```
SELECT TOP 50 *
FROM productos;
```

MySQL
```
SELECT *
FROM productos
LIMIT 50;
```

Oracle
```
SELECT *
FROM productos
WHERE rownum<=50;
```

https://msdn.microsoft.com/en-us/library/ms189463(v=sql.130).aspx TOP Clause

9 Eliminar filas duplicadas DISTINCT/ALL

SQL no elimina las filas duplicadas en el resultado de la consulta, si nosotros no queremos que se repitan las filas, tenemos la cláusula DISTINCT.

Al incluir la cláusula DISTINCT en la SELECT, se eliminará del resultado las **repeticiones** de **filas del resultado**. Si por el contrario queremos que aparezcan todas las filas seleccionadas podemos incluir la cláusula ALL o nada, ya que ALL es el valor por defecto.

Listar los nº de empleado de los directores de las oficinas:

```
SELECT  dir
FROM oficinas;
```

dir
106
104
105
108
108
108
108
NULL
NULL

Si un mismo empleado dirige varias oficinas (por ejemplo el 108), su código aparece repetido en el resultado.

Para evitarlo modificamos la consulta:

```
SELECT DISTINCT dir
FROM oficinas;
```

dir
NULL
104
105
106
108

Han desaparecido los valores duplicados.

Los que se eliminan son valores duplicados **de filas** del resultado, por ejemplo:

```
SELECT DISTINCT dir, region
FROM oficinas;
```

Dir	region
NULL	este
NULL	norte
104	este
105	este
106	este
108	centro
108	oeste

Ahora el 108 aparece dos veces porque las dos filas donde aparece no son iguales (por la región).

NOTA: La cláusula DISTINCT hace que la consulta tarde algo más en ejecutarse debido al proceso adicional de buscar y eliminar las repeticiones, por lo que se aconseja utilizarla únicamente cuando sea imprescindible.

10 Selección de filas del origen WHERE

La cláusula WHERE se emplea para especificar las filas que se desean recuperar del origen de datos.

```
WHERE <condicion_búsqueda>

<condicion_búsqueda> ::=
    { [NOT]<predicado>
     |(<condicion_búsqueda>)
    }
    [{AND|OR} [NOT] {<predicado>|(<condicion_búsqueda>)}]
[ ...n ]
```

En el resultado de la consulta sólo aparecerán las filas que cumplan que la condición de búsqueda especificada en el WHERE sea verdadera TRUE.

La condición de búsqueda puede ser una condición simple o una condición compuesta por varias condiciones (predicados) unidas por operadores AND y OR, no hay límite en cuanto al número de predicados que se pueden incluir. En las condiciones compuestas se pueden utilizar paréntesis para delimitar predicados y se aconseja su uso cuando se incluyen operadores AND y OR en la misma condición de búsqueda.

10.1 Predicados

En SQL tenemos 5 tipos de predicados *PREDICATES*, condiciones básicas de búsqueda:

- Comparación estándar
- Pertenencia a un intervalo (BETWEEN)
- Pertenencia a un conjunto (IN)
- Test de valor nulo (IS NULL).
- Coincidencia con patrón (LIKE)

Existen otros dos predicados (CONTAINS y FREETEXT) para búsquedas complejas en textos que no veremos.

10.1.1 Comparación estándar

Compara el valor de una expresión con el valor de otra. Para la comparación se pueden emplear = , <> , !=, < , <= , !<, > , >= ,!>

Sintaxis:

```
<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} <expresion>
```

<expresion> Puede ser:

- Un nombre de columna,
- una constante,
- una función (inclusive la función CASE),
- una variable,
- una subconsulta escalar o
- cualquier combinación de nombres de columna, constantes y funciones conectados mediante uno o varios operadores o una subconsulta.

Ejemplo:
Listar los "buenos" vendedores (los que han rebasado su cuota)

```
SELECT numemp, nombre, ventas, cuota
FROM empleados
WHERE ventas > cuota
```

Numemp	nombre	ventas	cuota
101	Antonio Viguer	30500,00	30000,00
102	Alvaro Jaumes	47400,00	35000,00
103	Juan Rovira	28600,00	27500,00
105	Vicente Pantalla	36800,00	35000,00
106	Luis Antonio	29900,00	27500,00
108	Ana Bustamante	36100,00	35000,00
109	María Sunta	39200,00	3000,00

Las columnas que aparecen en el WHERE no tienen por qué aparecer en la lista de selección, esta instrucción es igual de válida:

```
SELECT numemp, nombre
FROM empleados
WHERE ventas > cuota;
```

Hallar vendedores contratados antes de 1988.

```
SELECT numemp, nombre, contrato
FROM empleados
WHERE contrato < '01/01/1988';
```

Numemp	nombre	contrato
101	Antonio Viguer	1986-10-20
102	Alvaro Jaumes	1986-12-10
103	Juan Rovira	1987-03-01
104	José González	1987-05-19

También podemos utilizar funciones, ésta es equivalente a la anterior:

```
SELECT numemp, nombre
FROM empleados
WHERE YEAR(contrato) < 1988;
```

La función YEAR(fecha) devuelve el año de una fecha.

Hallar oficinas cuyas ventas estén por debajo del 80% de su objetivo:

```
 SELECT oficina
 FROM oficinas
 WHERE ventas < (.8 * objetivo);
```

Hallar las oficinas dirigidas por el empleado 108:

```
 SELECT oficina
 FROM oficinas
 WHERE dir = 108;
```

10.1.2 Pertenencia a un intervalo. BETWEEN

```
<expresion> [NOT] BETWEEN <expresion2> AND <expresion3>
```

Examina si el valor de la expresión de test está en el rango delimitado por los valores resultantes de expresion1 y expresion2, estos valores no tienen porqué estar ordenados en ANSI/ISO; expresion1 debe ser menor o igual a expresion2.

Hallar vendedores cuyas ventas estén entre 20.000 euros y 50.000.

```
SELECT numemp, nombre, ventas
FROM empleados
WHERE  ventas BETWEEN 20000 AND 100000;
```

Numemp	nombre	ventas
101	Antonio Viguer	30500,00
102	Alvaro Jaumes	47400,00
103	Juan Rovira	28600,00
105	Vicente Pantalla	36800,00
106	Luis Antonio	29900,00
108	Ana Bustamante	36100,00
109	María Sunta	39200,00

La instrucción anterior es equivalente a:

```
SELECT numemp, nombre, ventas
FROM empleados
WHERE  ventas >= 20000 AND ventas <=100000;
```

Parece que con BETWEEN se lee mejor.

Observa que no hemos utilizado separadores de millares (100.000), se habría interpretado por una coma decimal.

10.1.3 Test de pertenencia a conjunto IN

```
<expresion> IN ( <exp_valor> [ ,...n ] )
```

Examina si el valor de la expresion es uno de los valores incluidos en la lista de valores indicados entre paréntesis. Se pueden expresar los valores mediante cualquier expresión, la única condición es que todas las exp_valor devuelvan el mismo tipo de datos.

Ejemplo:
Obtener los empleados que trabajan en las oficinas 11, 20 o 22:

```
SELECT oficina, numemp, nombre
FROM empleados
WHERE oficina IN (11,20,22);
```

Oficina	numemp	nombre .
11	106	Luis Antonio
22	107	Jorge Gutiérrez
11	109	María Sunta

10.1.4 Test de valor nulo IS NULL

```
<expression> IS [NOT] NULL
```

Una condición de búsqueda puede ser TRUE, FALSE o NULL/UNKNOW, este último caso se produce cuando algún campo que interviene en la condición tiene valor NULL.

A veces es útil comprobar explícitamente los valores NULL en una condición de búsqueda ya que estas filas puede que queramos darles un tratamiento especial, para ello tenemos el predicado IS NULL.

Este test produce un valor TRUE o FALSE, por lo que se podrá combinar con otras condiciones. El valor NULL no es en sí un valor por eso no lo podemos utilizar en una igualdad.

```
SELECT numemp,nombre
FROM empleados
WHERE oficina = NULL;
```

Esta instrucción no da error pero no obtiene lo que en principio parece que quiere obtener. No obtenemos los empleados cuya oficina sea un valor nulo (es decir los empleados que no tienen oficina), no obtenemos nada, en cambio los obtendremos utilizando el test de valor nulo:

```
SELECT numemp,nombre, oficina
FROM empleados
WHERE oficina IS NULL;
```

Resultado:

Numemp	nombre	oficina
110	Juan Victor	NULL

Juan Victor es el único empleado que no tiene oficina asignada.

Listar los vendedores asignados a alguna oficina.

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina IS NOT NULL;
```

Numemp	nombre	oficina
101	Antonio Viguer	12
102	Alvaro Jaumes	21
103	Juan Rovira	12
104	José González	12
105	Vicente Pantalla	13
106	Luis Antonio	11
107	Jorge Gutiérrez	22
108	Ana Bustamante	21
109	María Sunta	11

10.1.5 Test de correspondencia con patrón LIKE

Se utiliza cuando queremos comparar el valor de una columna con un patrón en el que se utilice caracteres comodines.

```
<expression>  [NOT] LIKE <patron>  [ESCAPE 'car_escape']
```

Con expresión indicamos el valor a comparar (normalmente será el nombre de una columna) y patrón es la cadena que se busca.
El patrón es de tipo texto y tiene que escribirse entre comillas simples.
Dentro del patrón podemos utilizar los siguientes comodines:

% representa cualquier cadena de cero o más caracteres.

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE 'An%';
```

Numemp	nombre
101	Antonio Viguer
108	Ana Bustamante

Obtiene todos los nombres que empiecen por An

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '%z';
```

Numemp	nombre
104	José González
107	Jorge Gutiérrez

Obtiene los nombres que acaban en z.

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '%on%';
```

Numemp	nombre
101	Antonio Viguer
104	José González
106	Luis Antonio

Obtiene los nombres que contienen on.

_ representa cualquier carácter (sólo uno).

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '__a%';
```

Numemp	nombre
103	Juan Rovira
108	Ana Bustamante
110	Juan Victor

Obtiene los nombres cuya tercera letra sea una a (en el patrón tenemos dos caracteres subrayado).

[ ] sirve para indicar un carácter cualquiera perteneciente al conjunto indicando.
El conjunto se indica enumerando los caracteres o indicando un intervalo.

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '[a-d]%';
```

Obtiene los nombres que empiezan por cualquier letra de la *a* a la *d*

Es equivalente a escribir:

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '[abcd]%';
```

[^] significa cualquier carácter individual que no se encuentre en el conjunto.

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '[^abcd]%';
```

Y

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '[^a-d]%';
```

Obtienen los nombres que no empiecen por *a*, *b*, *c* ni *d*.

Es importante tener en cuenta que dentro del patrón el espacio en blanco es considerado como un carácter más, si colocamos dos espacios en el patrón, se buscarán dos espacios en el campo.

Si queremos incluir en el patrón uno de los caracteres comodines y que no sea interpretado como un comodín, sino como un carácter normal, lo tenemos que encerrar entre corchetes o utilizar un carácter de escape.

[ESCAPE 'car_escape']

La cláusula ESCAPE es opcional y permite definir un carácter de escape.

Un carácter de escape es un carácter que se coloca delante de un carácter comodín para indicar que el comodín no debe interpretarse como tal, sino como un carácter normal.

Por ejemplo queremos buscar los nombres compuestos que incluyen un subrayado. En este caso tenemos que poner el carácter _ como un carácter normal no como un comodín, así que lo escribiremos así:

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '%[_]%';
```

O bien,

```
SELECT numemp,nombre
FROM empleados
WHERE nombre LIKE '%!_%' ESCAPE '!';
```

10.2 Condiciones de búsqueda compuestas

En una cláusula WHERE podemos incluir una condición de búsqueda simple (formada por un solo predicado) o compuesta (formada por la combinación de predicados unidos por los operadores lógicos NOT, AND, OR).

Cuando la condición incluye varios operadores lógicos, el orden de prioridad de estos operadores es:

1. NOT (el más alto),
2. seguido de AND y OR (estos dos al mismo nivel).

Como siempre, se pueden utilizar paréntesis para alterar esta prioridad en una condición de búsqueda.

El orden de evaluación de los operadores lógicos puede variar dependiendo de las opciones elegidas por el optimizador de consultas.

Los operadores lógicos pueden devolver tres valores distintos: TRUE, FALSE, NULL (UNKNOWN).

Tablas de verdad de los operadores:

AND Combina dos condiciones y se evalúa como TRUE cuando ambas condiciones son TRUE.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR Combina dos condiciones y se evalúa como TRUE cuando alguna de las condiciones es TRUE.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT Niega la expresión booleana que especifica el predicado

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Hallar los vendedores que están por debajo de su cuota y tienen ventas inferiores a 30.000.

```
SELECT nombre
FROM empleados
WHERE ventas < cuota AND ventas < 30000;
```

Hallar los vendedores que están debajo de su cuota, pero cuyas ventas no sean inferiores a 150.000.

```
SELECT nombre
FROM empleados
WHERE ventas < cuota AND ventas !< 150000;
```

Hallar las oficinas no dirigidas por el empleado 108

```
SELECT oficina
FROM oficinas
WHERE NOT dir = 108;
```

O

```
SELECT oficina
FROM oficinas
WHERE  dir <> 108;
```

Devuelven:

oficina
11
12
13

Las oficinas sin director no aparecen, porque cuando Dir es nulo la comparación = (y <>) no es true ni false sino null por lo que las filas no son seleccionadas. Para que aparezcan debes añadir otro predicado:

```
SELECT oficina, dir
FROM oficinas
WHERE   NOT dir = 108 or dir is null;
```

Oficina	dir
11	106
12	104
13	105
26	NULL
28	NULL

https://msdn.microsoft.com/en-us/library/ms188047(v=sql.130).aspx WHERE clause