Administración Bases de Datos

Semana 5

Docente: Lic. Norberto A. Orlando

Lenguaje de definición de datos

Las instrucciones del lenguaje de definición de datos (DDL) definen estructuras de datos. Use estas instrucciones para crear, modificar o quitar estructuras de datos en una base de datos.

- ALTER
- CREATE
- DROP
- DISABLE TRIGGER
- ENABLE TRIGGER
- RENAME

Lenguaje de manipulación de datos

El lenguaje de manipulación de datos (DML) afecta a la información almacenada en la base de datos. Use estas instrucciones para insertar, actualizar y cambiar las filas de la base de datos.

- BULK INSERT
- DELETE
- INSERT
- UPDATE
- MERGE
- TRUNCATE TABLE

Seleccionar registros de una tabla: Cláusulas *SELECT*

La sentencia ***SELECT*** permite realizar operaciones de **selección, ordenación, agrupación y filtrado de registros**. Esta instrucción o sentencia utiliza diversas cláusulas:

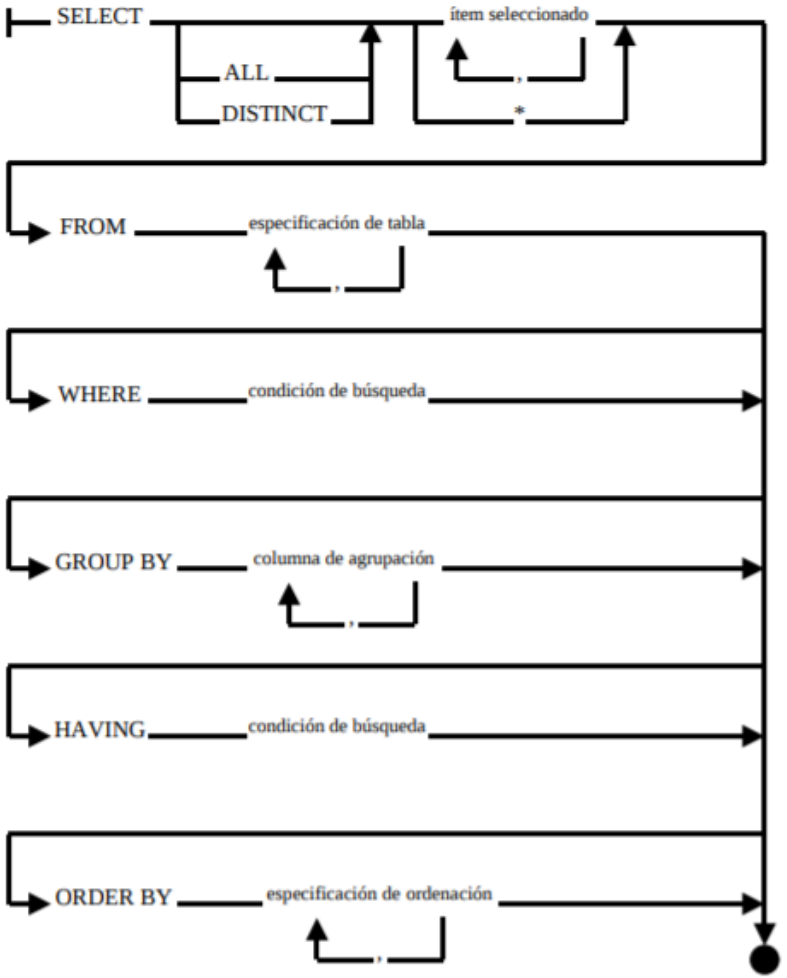
FROM	WHERE	GROUP BY	HAVING	ORDER BY
Especifica **la tabla** de la que se quieren obtener los registros	Especifica los **criterios o condiciones** que deben cumplir los registros a buscar dentro de la tabla	Permite **agrupar** los registros selecciona-dos en función de uno o más campos	Especifica las **condiciones** que deben cumplir los **grupos** generados	**Ordena** los registros seleccionados en **función de un campo**

SELECT

```
SELECT select_list
[ INTO new_table ]
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ]
```

```
SELECT *

FROM   nombre_tabla
```



Alias

Los **ALIAS** se usan para darle un nombre temporal y más amigable a las **tablas**, **columnas** y **funciones**. Los **alias** se definen durante una consulta y persisten **sólo** durante esa consulta.

Para definir un alias usamos las iniciales **AS** precediendo a la columna que estamos queriendo asignarle ese alias.

SQL

```sql
SELECT nombre_columna1 AS alias_nombre_columna1
FROM nombre_tabla;
```

Alias para una **COLUMNA**

```sql
SELECT razon_social_cliente AS nombre
FROM cliente
WHERE nombre LIKE 'a%';
```

Seleccionamos la **columna** *razon_social_cliente* y le asignamos el **ALIAS** nombre.

Alias para una **TABLA**

```
SELECT nombre, apellido, edad
FROM alumnos_comision_inicial AS alumnos;
```

Hacemos la consulta sobre la tabla *alumnos_comision_inicial* y le **asignamos** el **ALIAS** alumnos.

Para asignar un alias con espacio es necesario escribirlo entre comillas simples:
FROM alumnos_comision_inicial **AS** 'Los alumnos';

De este modo, podemos darle alias a las **columnas** y **tablas** que vamos trayendo y hacer más legible la manipulación de datos, teniendo siempre presente que los alias **no modifican** los nombres originales en la base de datos.

SELECT

CÓMO USARLO

Toda consulta a la base de datos va a empezar con la palabra **SELECT**. Su funcionalidad es la de realizar consultas sobre **una** o **varias columnas** de una tabla.

Para especificar sobre qué tabla queremos realizar esa consulta usamos la palabra **FROM** seguida del nombre de la tabla.

SQL

```sql
SELECT nombre_columna, nombre_columna, ...
FROM nombre_tabla;
```

EJEMPLO

id	titulo	rating	fecha_estreno	pais
1001	Pulp Fiction	9.8	1995-02-16	Estados Unidos
1002	Kill Bill Vol. 1	9.5	2003-11-27	Estados Unidos

Para conocer los títulos y ratings de las películas guardadas en la tabla **peliculas,** podríamos hacerlo ejecutando la siguiente consulta:

SQL

```sql
SELECT titulo, rating
FROM peliculas;
```

Mostrar todo el contenido de una tabla

En el ejemplo de la derecha, se **selecciona de la tabla *Articulos* todos los registros contenidos en la tabla y se muestran todas las columnas**.

El **asterisco (*)** ubicado a continuación de la sentencia *SELECT* especifica que, en el resultado de la consulta, se deben **mostrar todas las columnas (campos) contenidos en la tabla**.

```
SELECT * FROM Articulos;
```

> **Nota:** En este ejemplo, sólo se hace uso de la cláusula ***FROM***. No se utilizan todas las otras cláusulas, ya que el objetivo final era obtener un **listado de todos los registros** contenidos en la tabla.

Generar columnas en una consulta

En el siguiente ejemplo, se selecciona de la tabla *Articulos* los valores de todas las columnas y se **agrega una nueva columna** con el nombre ***Precio con Aumento***, **al incrementar en un 25% el valor de la columna *Precio***:

```
SELECT *, Precio * 1.25 as 'Precio con Aumento' FROM Articulos;
```

Operadores

Operadores de comparación

Los operadores de comparación comprueban si dos expresiones son iguales o distintas. Se pueden usar en todas las expresiones, excepto en las de los tipos de datos *text, ntext, image.*

En la siguiente tabla se presentan los operadores de comparación Transact-SQL.

Operador	Significado
=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	No es igual a
!=	No es igual a (no es del estándar ISO)
!<	No es menor que (no es del estándar ISO)
!>	No es mayor que (no es del estándar ISO)

En el siguiente ejemplo, se selecciona de la tabla *Articulos* la columna ***Nombre*** y muestra todos aquellos registros cuyo **valor en la columna *codigo* sea igual a 1**:

```
SELECT Nombre FROM Articulos WHERE codigo = 1;
```

Y en el siguiente ejemplo, se selecciona de la tabla *Articulos* las columnas ***Nombre*** y ***Precio*** y muestra **todos aquellos registros cuyo precio sea superior a 150:**

```
SELECT Nombre, Precio FROM Articulos WHERE Precio > 150;
```

Operadores de comparación

El ejemplo siguiente hace uso de los operadores mayor, menor y distinto.

Sintaxis

```
SELECT Name, ListPrice, MakeFlag
FROM   Production.Product
WHERE  ListPrice > 0 AND ListPrice < 40 AND MakeFlag <> 0;
----------------------------------------
Name             ListPrice  MakeFlag
LL Headset          34,20
```

Operadores lógicos

Para crear expresiones lógicas disponemos de varios **operadores de comparación**.

Operador	Descripción
AND	Se deben cumplir **todas** las condiciones especificadas
OR	Se debe cumplir **al menos una** de las condiciones especificadas
NOT	**No debe cumplir** las condiciones especificadas

Estos operadores se aplican a cualquier tipo de columna (fechas, cadenas, números, etc.). Y devuelven valores lógicos, que son: **verdadero o falso** (**1 ó 0**).

- Si uno o los dos valores a comparar son ***NULL***, el resultado es *NULL*.
  (Excepto con el operador <=> que es usado para una comparación con *NULL* segura).
- El operador <=> funciona igual que el operador =. Salvo que, si en la comparación una o ambas de las expresiones es nula, el resultado no es *NULL*. Si se comparan dos expresiones nulas, el resultado es verdadero.

En el siguiente ejemplo, se selecciona de la tabla *Articulos* todos aquellos registros cuyo ***precio*** tenga un **valor mayor o igual a 500, o su *stock* sea mayor o igual a 100**:

```
SELECT * FROM Articulos WHERE precio >= 500 OR stock >= 100;
```

Y en el siguiente ejemplo, se selecciona de la tabla *Articulos* todos aquellos registros cuyo ***precio*** tenga un **valor menor a 20 y su *stock* sea mayor o igual a 100**:

```
SELECT * FROM Articulos WHERE Precio < 20 AND stock >= 100;
```

Operadores

Tabla de verdad

Condiciones		Operadores Lógicos	
Primera Condición	**Segunda Condición**	**AND**	**OR**
VERDADERA	VERDADERA	TRUE	TRUE
VERDADERA	FALSE	FALSE	TRUE
FALSE	VERDADERA	FALSE	TURE
FALSE	FALSE	FALSE	FALSE

Operadores

Tabla de verdad

```
SELECT
      ProductID, ListPrice, Color
FROM
      Production.Product
WHERE
      (ListPrice<60 AND Color='Yellow') OR (ListPrice>2318 AND Color='Silver');

---------------------------------
ProductID   ListPrice   Color
771          3399,99    Silver
772          3399,99    Silver
773          3399,99    Silver
774          3399,99    Silver
779          2319,99    Silver
780          2319,99    Silver
781          2319,99    Silver
881            53,99    Yellow
882            53,99    Yellow
883            53,99    Yellow
884            53,99    Yellow
```

Operadores *IS NULL / IS NOT NULL*

Los operadores *IS NULL* e *IS NOT NULL* sirven para **verificar** si una expresión determinada **es o no nula**.

En el siguiente ejemplo, se mostrará todos aquellos registros de la tabla ***clientes*** que ***no*** **tengan cargado ningún valor en el campo *comentarios***:

```
SELECT * FROM clientes WHERE comentarios IS NULL;
```

Y en el siguiente ejemplo, se mostrará todos aquellos registros de la tabla ***clientes*** **que tengan algún valor cargado en el campo *comentarios***:

```
SELECT * FROM clientes WHERE comentarios IS NOT NULL;
```

WHERE

La funcionalidad del **WHERE** es la de condicionar y filtrar las consultas **SELECT** que se realizan a una base de datos.

SQL

```sql
SELECT nombre_columna_1, nombre_columna_2, ...
FROM nombre_tabla
WHERE condicion;
```

Teniendo una tabla **usuarios**, podría consultar nombre y edad, filtrando con un **WHERE** sólo los usuarios **mayores de 17 años** de la siguiente manera:

SQL

```sql
SELECT nombre, edad
FROM usuarios
WHERE edad > 17;
```

Operadores *BETWEEN / NOT BETWEEN*

Entre los operadores de ***MySQL***, existe uno denominado ***BETWEEN* (entre)**, el cual se utiliza para comprobar si una expresión está comprendida en un determinado **rango de valores**. La sintaxis es:

- **BETWEEN** mínimo **AND** máximo

```
SELECT * FROM Articulos WHERE precio BETWEEN 100 AND 200;
```

- **NOT BETWEEN** mínimo **AND** máximo

```
SELECT * FROM Articulos WHERE precio NOT BETWEEN 100 AND 200;
```

Operadores *IN / NOT IN*

Los operadores ***IN*** y ***NOT IN*** sirven para averiguar si el valor de una expresión determinada se encuentra **dentro de un conjunto indicado**.

Su sintaxis es:

- `IN (<expr1>, <expr2>, <expr3>,...)`
- `NOT IN (<expr1>, <expr2>, <expr3>,...)`

- El operador ***IN*** devuelve un valor **verdadero** si el valor de la expresión es **igual a alguno de los valores** especificados en la lista.
- El operador ***NOT IN*** devuelve un valor **falso** en el **caso contrario**.

Ejemplo 1:

```
SELECT * FROM Articulos WHERE codigo IN (1,2,3);
```

Ejemplo 2:

```
SELECT * FROM Articulos WHERE nombre IN ('Pala', 'Maza');
```

Ejemplo 3:

```
SELECT * FROM Articulos WHERE nombre NOT IN ('Pala', 'Maza');
```

LIKE

Determina si una cadena de caracteres específica coincide con un patrón especificado.

Un patrón puede contener caracteres *normales* y caracteres *comodín.* Durante la operación de búsqueda de coincidencias de patrón, los caracteres normales 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.

El uso de caracteres comodín hace que el operador LIKE sea más flexible que los operadores de comparación de cadenas = y !=. Si alguno de los argumentos no es del tipo de datos de cadena de caracteres, Motor de base de datos de SQL Server lo convierte al tipo de datos de cadena de caracteres, si es posible.

LIKE

Comodín	Descripción	Ejemplo
%	Cualquier cadena de cero o más caracteres.	WHERE title LIKE '%computer%' busca todos los títulos de libros que contengan la palabra 'computer' en el título.
_	Cualquier carácter individual.	WHERE au_fname LIKE '_ean' busca todos los nombres de cuatro letras que terminen en ean (Dean, Sean, etc.)
[ ]	Cualquier carácter individual del intervalo ([a-f]) o del conjunto ([abcdef]) que se ha especificado.	WHERE au_lname LIKE '[C-P]arsen' busca apellidos de autores que terminen en arsen y empiecen por cualquier carácter individual entre C y P, como Carsen, Larsen, Karsen, etc. En las búsquedas de intervalos, los caracteres incluidos en el intervalo pueden variar, dependiendo de las reglas de ordenación de la intercalación.
[^]	Cualquier carácter individual que no se encuentre en el intervalo ([^a-f]) o el conjunto ([^abcdef]) que se ha especificado.	WHERE au_lname LIKE 'de[^l]%' busca todos los apellidos de autores que empiezan por de y en los que la siguiente letra no sea l.

ORDER BY

Ordena los datos devueltos por una consulta en SQL Server.

Use esta cláusula para ordenar el conjunto de resultados de una consulta por la lista de columnas especificada y, opcionalmente, limitar las filas devueltas a un intervalo especificado.

El orden en que se devuelven las filas en un conjunto de resultados no se puede garantizar, a menos que se especifique una cláusula ORDER BY.

Características

- Especifica el orden de los resultados de la sentencia SELECT
- Ordena los resultados de una consulta.
- Se puede realizar el ordenamiento por una o más columnas
- Esta cláusula es inválida para las vistas
- Se puede determinar que el orden sea Ascendente o Descendente

ORDER BY

Sintaxis
En el siguiente ejemplo se buscan las personas ordenadas por Nombre y luego por Apellido.

```
SELECT FirstName, LastName
FROM Person.Person
ORDER BY FirstName ASC, LastName DESC;
```

SELECT DISTINCT

Devuelve todos los estados posibles para la columna seleccionada del modelo.

Los valores devueltos varían dependiendo de si la columna especificada contiene valores discretos, valores numéricos de datos discretos o valores numéricos continuos.

Sintaxis

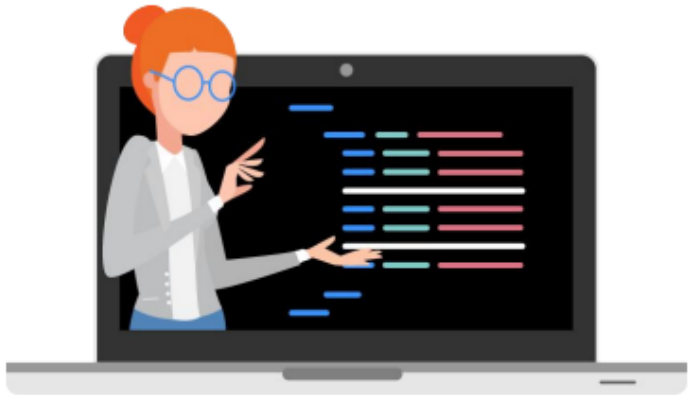
```
SELECT DISTINCT ProductID
FROM Production.Product;
```

Operador de conjunto UNION

Combina los resultados de dos o más consultas en un solo conjunto de resultados que incluye todas las filas que pertenecen a las consultas de la unión. La operación UNION es distinta de la utilización de combinaciones de columnas de dos tablas.

A continuación, se muestran las reglas básicas para combinar los conjuntos de resultados de dos consultas con UNION:

- El número y el orden de las columnas debe ser el mismo en todas las consultas.
- Los tipos de datos deben ser compatibles.



Operador de conjunto UNION

UNION

Específica que se deben combinar varios conjuntos de resultados para ser devueltos como un solo conjunto de resultados.

Sintaxis

```
SELECT BusinessEntityID
FROM Sales.SalesPerson
UNION

SELECT BusinessEntityID
FROM HumanResources.Employee;
```

UNION ALL

Agrega todas las filas a los resultados. Incluye las filas duplicadas. Si no se especifica, las filas duplicadas se quitan.

Sintaxis

```
SELECT BusinessEntityID
FROM Sales.SalesPerson

UNION ALL
SELECT BusinessEntityID
FROM HumanResources.Employee;
```

Case

CASE

Evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles. La expresión CASE tiene dos formatos:

- La expresión CASE sencilla compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.
- La expresión CASE buscada evalúa un conjunto de expresiones booleanas para determinar el resultado.

Ambos formatos admiten un argumento ELSE opcional.

CASE se puede utilizar en cualquier instrucción o cláusula que permite una expresión válida. Por ejemplo, puede utilizar CASE en instrucciones como SELECT, UPDATE, DELETE y SET, y en cláusulas como “select_list”, IN, WHERE, ORDER BY y HAVING.

Case Sintaxis

```
SELECT   ProductLine
         ,Category = CASE ProductLine
                         WHEN 'R' THEN 'Road'
                         WHEN 'M' THEN 'Mountain'
                         WHEN 'T' THEN 'Touring'
                         ELSE 'Not for sale'
                     END
FROM Production.Product;
```

```
SELECT
       CASE WHEN EmailPromotion=0 then 'No tiene mail'
            WHEN EmailPromotion=1 then 'Tiene mail'
            WHEN EmailPromotion=2 then 'Tiene muchos mail'
            ELSE 'Desconocido'
       END Mail
FROM
       Person.Person;
```

```
SELECT
      LastName
       ,TerritoryName
       ,CountryRegionName
FROM
       Sales.vSalesPerson
WHERE
       TerritoryName IS NOT NULL
ORDER BY CASE CountryRegionName WHEN 'United States' THEN TerritoryName
         ELSE CountryRegionName END;
```

Funciones de Agregado

Una función de agregado realiza un cálculo sobre un conjunto de valores y devuelve un solo valor.

Las funciones de agregado ignoran los valores NULL.

Las funciones de agregado se suelen usar con la cláusula GROUP BY de la instrucción SELECT.

SQL proporciona las siguientes funciones de agregado:

AVG	Devuelve el promedio
MAX	Devuelve el valor máximo
MIN	Devuelve el valor mínimo
SUM	Devuelve la sumatoria
COUNT	Devuelve la cantidad. El resultado es de tipo int
COUNT_BIG	Devuelve la cantidad. El resultado es de tipo bigint
VAR	Devuelve la varianza estándar
VARP	Devuelve la varianza de la población
STDEVP	Devuelve la desviación estándar
STDEV	Devuelve la desviación de la población

Función COUNT

Esta función devuelve el número de elementos encontrados en un grupo. COUNT funciona como la función COUNT_BIG. Estas funciones difieren sólo en los tipos de datos de sus valores devueltos.

COUNT siempre devuelve un valor de tipo de datos int. COUNT_BIG siempre devuelve un valor de tipo de datos bigint.

Sintaxis

```
SELECT COUNT(*) AS Cantidad

FROM HumanResources.EmployeeDepartmentHistory;
-------------
Cantidad
296
```

Comportamiento con valores NULL de la función de agregado Count. Dada la siguiente tabla veamos los posibles resultados:

ID	MONTO
1	10
2	20
NULL	10
4	5

RESULTADO	
COUNT(*)	4
COUNT(1)	4
COUNT(MONTO)	4
COUNT (ID)	3

Importante: La función de agregado Count() ignora los valores NULL.

Función MAX

Devuelve el valor máximo de la expresión.

ID	MONTO
1	10
2	20
NULL	10
4	5

RESULTADO	
MAX(MONTO)	20

Sintaxis

```
SELECT MAX(ListPrice) AS Maximo

FROM Production.Product;
--------------
Maximo
3578,27
```

Función MIN

Devuelve el valor mínimo de la expresión

ID	MONTO
1	10
2	20
NULL	10
4	5

RESULTADO	
MIN(MONTO)	5

Sintaxis

```
SELECT MIN(ListPrice) AS Minimo

FROM Production.Product;
------------
Minimo
0,00
```

Función SUM

Devuelve la suma de todos los valores o sólo de los valores DISTINCT de la expresión. SUM sólo puede utilizarse con columnas numéricas. Se omiten los valores NULL.

ID	MONTO
1	10
2	20
NULL	10
4	5

RESULTADO	
SUM(MONTO)	45

Sintaxis

```
SELECT SUM(ListPrice) AS Total

FROM Production.Product;
--------------------
Total
221102,80
```

Función AVG

Devuelve el promedio de los valores de un grupo. Omite los valores NULL.

ID	MONTO
1	10
2	20
NULL	10
4	5

RESULTADO	
AVG(MONTO)	11.25

Sintaxis

```
SELECT AVG(ListPrice) AS Promedio

FROM Production.Product;
---------------
Promedio
438,696
```

Funciones de Agregado

Función AVG con campos NULL

Tabla1

Col1
10
5
0
3
0
6

Tabla2

Col1
10
5
NULL
3
NULL
6

SELECT AVG(col1) AS media FROM tabla1

En este caso los ceros entran en la media por lo que sale igual a 4
(10+5+0+3+0+6)/6 = 4

SELECT AVG(col1) AS media FROM tabla2

En este caso los ceros se han sustituido por valores nulos y no entran en el cálculo por lo que la media sale igual a 6
(10+5+3+6)/4 = 6

Agrupación y Funciones de Agregado

Agrupa un conjunto de filas seleccionado en un conjunto de filas de resumen de acuerdo con los valores de una o más columnas o expresiones.

Se devuelve una fila para cada grupo. Las funciones de agregado de la lista <select> de la cláusula SELECT proporcionan información de cada grupo en lugar de filas individuales.

Agrupación y Funciones de Agregado

Características

- Especifica los grupos en los que se deben colocar las filas de salida
- GROUP BY Establece la lista de columnas por las cuales se agrupará la información
- Esta cláusula organiza los resúmenes de datos agrupados.
- Si se incluyen funciones de agregado en la <lista de selección> de la cláusula SELECT, GROUP BY calcula un valor de resumen para cada grupo.
- Cuando se especifica GROUP BY, cada columna que esté en una expresión no agregada de la lista de selección se debe incluir en la lista de GROUP BY o la expresión GROUP BY debe coincidir exactamente con la expresión de la lista de selección.
- Si no se especifica la cláusula ORDER BY, los grupos devueltos con la cláusula GROUP BY no están en un orden determinado. Se recomienda utilizar siempre la cláusula ORDER BY para especificar un orden.

Agrupación y Funciones de Agregado

ID	MONTO
1	10
1	20
2	10
2	5

SUMATORIA AGRUPADO POR ID	
GRUPO 1	30
GRUPO 2	15

Sintaxis

```
SELECT ProductID, MAX(LineTotal) as Maximo
FROM Sales.SalesOrderDetail
WHERE ProductID > 995
GROUP BY ProductID;
----------------------
ProductID  Maximo
999         3990.094108
997         2915.946000
...
```

HAVING

Especifica una condición de búsqueda para un grupo. HAVING solo se puede utilizar con la instrucción SELECT.

Normalmente, HAVING se usa con una cláusula GROUP BY. Cuando no se usa GROUP BY, hay un solo grupo implícito agregado.

ID	MONTO
1	10
1	20
2	10
2	5

HAVING SUM(MONTO) >15	
GRUPO 1	30
GRUPO 2	15

Sintaxis

```
SELECT ProductID, MAX(LineTotal) as Maximo
FROM Sales.SalesOrderDetail
WHERE ProductID>995
GROUP BY ProductID
HAVING MAX(LineTotal)>3000;

----------------------
ProductID  Maximo
999         3990.094108
998         5925.040275
```