

Tecnicatura Universitaria
en Programación

BASES DE DATOS I

Unidad Temática N°1:
Resumen de Datos

Teórico
1° Año – 2° Cuatrimestre



Índice

Consultas sumarias	2
Consultas agrupadas	5
Restricciones en consultas agrupadas.	6
Cláusula HAVING	7
Las restricciones en condiciones de búsqueda de grupos.	7
Combinación de resultados de consulta. UNION	9
Restricciones del UNION	10
UNION ALL.....	10
Introducción a las Vistas	11
Ventajas de las vistas	11
Creación de Vistas.....	12
Eliminar Vistas	14
Modificar Vistas	14
BIBLIOGRAFÍA	17

Consultas sumarias

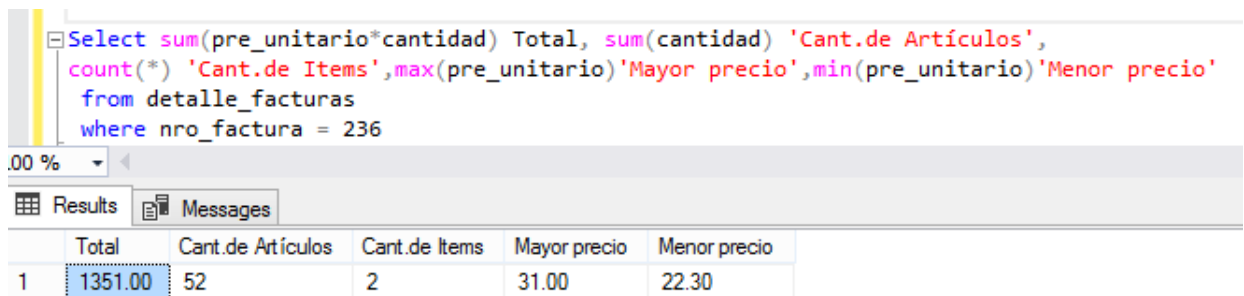
SQL permite sumarizar u obtener resumen de las tablas de la datos de la base de datos mediante un conjunto de funciones de columnas o funciones de agregado.

Estas funciones en SQL aceptan una columna entera de datos como argumentos y produce un único dato que sumariza o resume la columna.

Las funciones de columnas o funciones de agregado en SQL Server son:

- SUM(columna)** calcula el total de una columna
- AVG(columna)** calcula el valor promedio de una columna que surge de la suma de los valores de la columna dividido la cantidad de filas. Es equivalente a $\text{sum(columna)/count(*)}$
- MIN(columna)** encuentra el valor más pequeño en una columna
- MAX(columna)** encuentra el valor mayor en una columna
- COUNT(*)** cuenta las filas de resultados de la columna

Se quiere saber el total de la factura Nro. 236, la cantidad de artículos vendidos, cantidad de ventas (cantidad de registros en la tabla detalle_facturas, el precio máximo y mínimo vendido, en esa factura



The screenshot shows a SQL query in the Enterprise Manager query editor. The query is: `Select sum(pre_unitario*cantidad) Total, sum(cantidad) 'Cant.de Artículos', count(*) 'Cant.de Items', max(pre_unitario) 'Mayor precio', min(pre_unitario) 'Menor precio' from detalle_facturas where nro_factura = 236`. Below the query, the 'Results' tab is active, displaying a table with 5 columns: Total, Cant.de Artículos, Cant.de Items, Mayor precio, and Menor precio. The first row of data shows: 1, 1351.00, 52, 2, 31.00, 22.30.

	Total	Cant.de Artículos	Cant.de Items	Mayor precio	Menor precio
1	1351.00	52	2	31.00	22.30

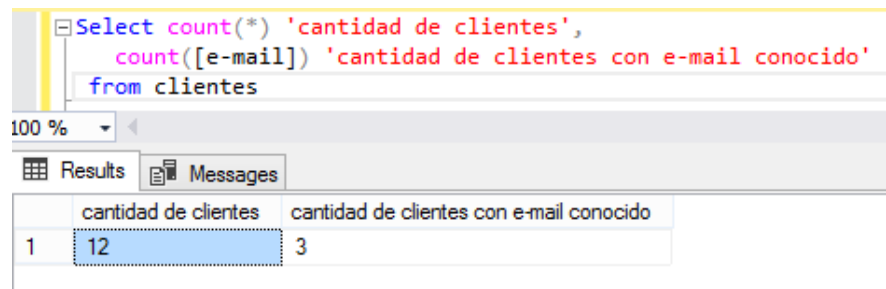
Contar registros

COUNT(columna)

Como ya se dijo antes **COUNT(*)** cuenta el número de registros que da por resultado una consulta, incluyendo las filas con columnas con valores nulos.

Si lo que se quiere es contar la cantidad de valores reales de una columna es decir, los valores no nulos de una columna se puede utilizar **COUNT(columna)**.

Por ejemplo:



```
Select count(*) 'cantidad de clientes',  
count([e-mail]) 'cantidad de clientes con e-mail conocido'  
from clientes
```

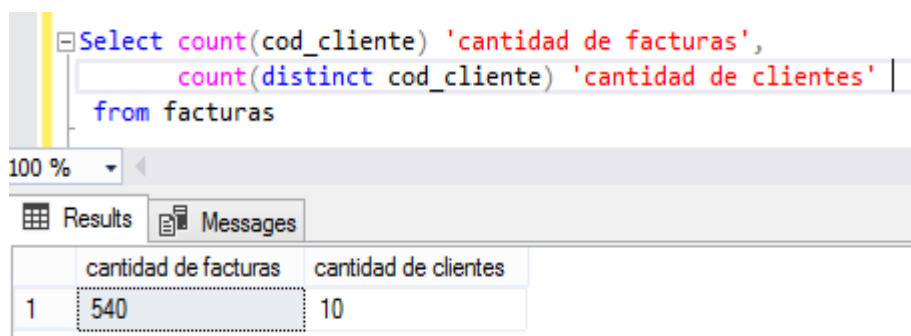
	cantidad de clientes	cantidad de clientes con e-mail conocido
1	12	3

La primera columna daría la cantidad de clientes que se tienen, es decir la cantidad de registros de la tabla clientes la segunda columna devolvería la cantidad de clientes con dirección de correo electrónico, es decir, la cantidad de registros cuyo campo e-mail no contiene valor *null*.

COUNT(distinct columna)

Todas las opciones de **COUNT** anteriores cuenta todos los registros, aunque estos tengan valor duplicado, si lo que se desea es que el valor devuelto no cuente valores duplicados se puede combinar esta función con **DISTINCT**. Este uso del count va a devolver la cantidad de valores distintos que tenga la columna en la consulta.

Por ejemplo:



```
Select count(cod_cliente) 'cantidad de facturas',  
count(distinct cod_cliente) 'cantidad de clientes' |  
from facturas
```

	cantidad de facturas	cantidad de clientes
1	540	10

La primer columna devolvería la cantidad total de registros de la tabla factura que tengan un cliente (no contaría registros donde el código de cliente sea null) y en este caso daría el mismo resultado que **COUNT(*)** o **COUNT(nro_factura)**, ya que en la tabla facturas de la base de datos librería no hay facturas con el cod_cliente nulo.

La segunda columna devolvería la cantidad de clientes (distintos) a los que alguna vez se le ha facturado independientemente de la cantidad de facturas que se le haya registrado a cada uno es decir no cuenta los valores duplicados de `cod_cliente`.

Promedio

Se necesita conocer el promedio de facturación por factura el año pasado. Si utilizamos esto:

```
SELECT AVG (pre_unitario*cantidad)
FROM detalle_facturas d, facturas f
WHERE d.nro_factura=f.nro_factura
And year(fecha)=year(getdate())-1
```

La función de agregado **AVG(columna)** suma el valor de la columna y la divide por la cantidad de registros que contenga la consulta, en otras palabras lo que hace es la aplicación de esta operación: **SUM(columna)/COUNT(*)** con lo que no estaría dando el promedio por factura sino por detalle de factura para dar la solución a lo que se pide, habría que utilizar en la consulta anterior: **SUM(pre_unitario*cantidad)/COUNT(distinct nro_factura)**

Aquí se resuelve dando ambos resultados para que se entienda mejor:

```
select avg(pre_unitario*cantidad) 'promedio por detalle de factura',
       sum(pre_unitario*cantidad)/count(distinct d.nro_factura) 'promedio por factura'
from detalle_facturas d, facturas f
where d.nro_factura=f.nro_factura
and year(fecha)=year(getdate())-1
```

Results	
Messages	
promedio por detalle de factura	promedio por factura
538.307821	963.571000

La consulta está devolviendo dos promedios de las fechas que correspondan al año actual menos uno es decir el año anterior al actual.

La primer columna calcula el promedio de la cantidad vendida por el precio unitario de la venta de cada artículo dividiendo por la cantidad de registros que tienen todos los detalles de esas facturas.

La segunda columna, calcula la suma de la cantidad vendida por el precio unitario de la venta de cada artículo dividido por la cantidad de número de facturas distintos es decir uno por cada factura (ya que `nro_factura` el PK en facturas)

Consultas agrupadas

Cláusula GROUP BY

Las consultas sumarias son como totales finales de un informe; si lo que se necesita es sumarizar u obtener resúmenes de columnas de la consulta a un nivel de subtotal es decir por cada cambio en una columna en particular, se utiliza la cláusula **GROUP BY** de la sentencia **SELECT**.

Es decir, agrupar registros según los valores de una o más columnas y obtener totales o resúmenes de cada grupo.

Una consulta que incluya la cláusula **GROUP BY** se denomina consulta agrupada, ya que agrupa los datos de las tablas fuente y produce una única fila sumaria por cada grupo de filas.

Las columnas indicadas en la cláusula **GROUP BY** se denominan columnas de agrupación de la consulta, ya que ellas son las que determinan cómo se dividen las filas en grupo.

Si se incluyen funciones agregadas en la lista SELECT, GROUP BY calcula un valor de resumen para cada grupo. Estos se conocen como agregados vectoriales.

Los agregados AVG (DISTINCT columna), COUNT (DISTINCT columna) y SUM (DISTINCT columna) son conocidos como agregados **distinct**

Múltiples columnas de agrupación.

SQL puede agrupar resultados de consultas en base a contenidos de dos o más columnas. Por ejemplo, calcular el total facturado por cada vendedor y a cada cliente el año pasado ordenado por vendedor primero y luego por cliente:

```

select v.cod_vendedor,ape_vendedor+' '+nom_vendedor'Vendedor', ape_cliente+' '+nom_cliente'Cliente',sum(pre_unitario*cantidad)'Total'
from detalle_facturas d, facturas f,vendedores v, clientes c
where d.nro_factura=f.nro_factura and f.cod_cliente=c.cod_cliente
and f.cod_vendedor=v.cod_vendedor and year(fecha)=year(getdate())-1
group by v.cod_vendedor,ape_vendedor+' '+nom_vendedor,c.cod_cliente,ape_cliente+' '+nom_cliente
order by 2,3

```

	cod_vendedor	Vendedor	Cliente	Total
1	1	Camizo Martín	Abarca Héctor	817.00
2	1	Camizo Martín	Castillo Marta Analía	558.00
3	1	Camizo Martín	Luque Elvira Josefa	150.00
4	1	Camizo Martín	Morales Santiago	3390.00
5	1	Camizo Martín	Paez Roque	200.00
6	1	Camizo Martín	Perez Carlos Antonio	3406.00
7	1	Camizo Martín	Perez Rodolfo	551.50
8	1	Camizo Martín	Ruiz Marcos	800.00
9	2	Ledesma Mariela	Abarca Héctor	996.00
10	2	Ledesma Mariela	Castillo Marta Analía	120.00
11	2	Ledesma Mariela	Luque Elvira Josefa	4759.00
12	2	Ledesma Mariela	Morales Pilar	4665.00
13	2	Ledesma Mariela	Morales Santiago	2692.50
14	2	Ledesma Mariela	Paez Roque	2690.00
15	2	Ledesma Mariela	Perez Carlos Antonio	299.00
16	2	Ledesma Mariela	Perez Rodolfo	1672.00
17	2	Ledesma Mariela	Ruiz Marcos	2574.50
18	3	Lopez Alejandro	Abarca Héctor	1174.00

Esta consulta respondería a la pregunta: ¿Cuánto le vendió cada vendedor a cada cliente el año pasado?

En este ejemplo se debe observar detenidamente, qué es lo que se incluye como columnas de agrupación, es decir en el group by: las columnas correspondientes al vendedor y las que corresponden a clientes

Por otro lado, qué columnas son las que se utilizaron en la lista de selección de la misma consulta: columnas dentro de funciones de agregado y las columnas de agrupación.

Restricciones en consultas agrupadas.

Las consultas agrupadas están sujetas a algunas limitaciones bastante estrictas:

- Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula from de la consulta.
- No se pueden agrupar las filas basándose en el valor de una expresión calculada.
- Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas, es decir, pueden ser: una constante, una función de columna o de agregado (que producen un único valor sumalizando las filas del grupo), una columna de agrupación (que tienen el mismo valor) o una expresión que afecte a combinaciones de los anteriores.

Cláusula HAVING

Se utiliza para agregar condiciones de búsqueda para grupos es decir para las filas que resultan de la agrupación y cálculo de resultados de las funciones de agregado. Estas condiciones son las mismas explicadas

```
Select nro_factura, sum(pre_unitario*cantidad) Total
from detalle_facturas
group by nro_factura
having sum(pre_unitario*cantidad)>2500
```

para la cláusula **WHERE**

Results			Messages	
	nro_factura	Total		
1	45	2565.00		
2	167	2630.00		
3	223	3012.50		
4	287	2630.00		
5	311	2630.00		
6	403	3470.90		
7	406	3058.00		
8	414	2947.70		
9	420	4170.00		
10	422	3040.00		
11	461	2510.00		
12	465	3154.50		
13	466	2662.50		
14	470	2966.00		
15	475	2878.50		
16	513	2857.30		
17	519	2533.50		
18	522	3040.00		
19	532	3040.00		
20	537	2734.50		

El ejemplo muestra los montos totales por factura solo de aquellas cuyo monto total sea mayor a 2500.

Las restricciones en condiciones de búsqueda de grupos.

La cláusula **HAVING** se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales.

Esto significa que un elemento que aparezca dentro de la condición de búsqueda en el **HAVING** puede ser alguna de las mismas que las enumeradas en el **GROUP BY**.

Se sabe que, la cláusula **WHERE** se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales y la cláusula **HAVING** se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.


```
1 Select f.cod_vendedor Código, ape_vendedor Apellido, sum(pre_unitario*cantidad) Total
2 from detalle_facturas d, facturas f,vendedores v
3 where d.nro_factura = f.nro_factura and f.cod_vendedor=v.cod_vendedor
4 and year(fecha) = 2017
5 group by f.cod_vendedor, ape_vendedor
6 having sum(pre_unitario*cantidad)<17000
```

Results			Messages
Código	Apellido	Total	
1	Camizo	16380.30	
5	Monti	14732.00	

Por lo que, en el ejemplo anterior, la condición de búsqueda sobre la fecha de la factura como no está incluida en la agrupación del **GROUP BY** ni tampoco en una función de agregado deberá ir en el **WHERE** como también las condiciones para realizar la composición de tablas.

La condición referida al importe total de facturación como es una función de agregado no podrá ir en el **WHERE** sino en el **HAVING**.

Cláusula WHERE:

SQL elimina las filas que no cumplen las condiciones de la cláusula WHERE antes de realizar cualquier operación de agrupación.

Cláusula HAVING:

SQL usa la cláusula HAVING para filtrar grupos en el conjunto de resultados.

Cláusula ORDER BY:

Utilizar la cláusula ORDER BY para ordenar el conjunto de resultados. La cláusula GROUP BY no ordena el conjunto de resultados.

Valores NULL:

Si una columna de agrupación contiene valores NULL, todos los valores NULL se consideran iguales y se recopilan en un solo grupo.

Combinación de resultados de consulta. UNION

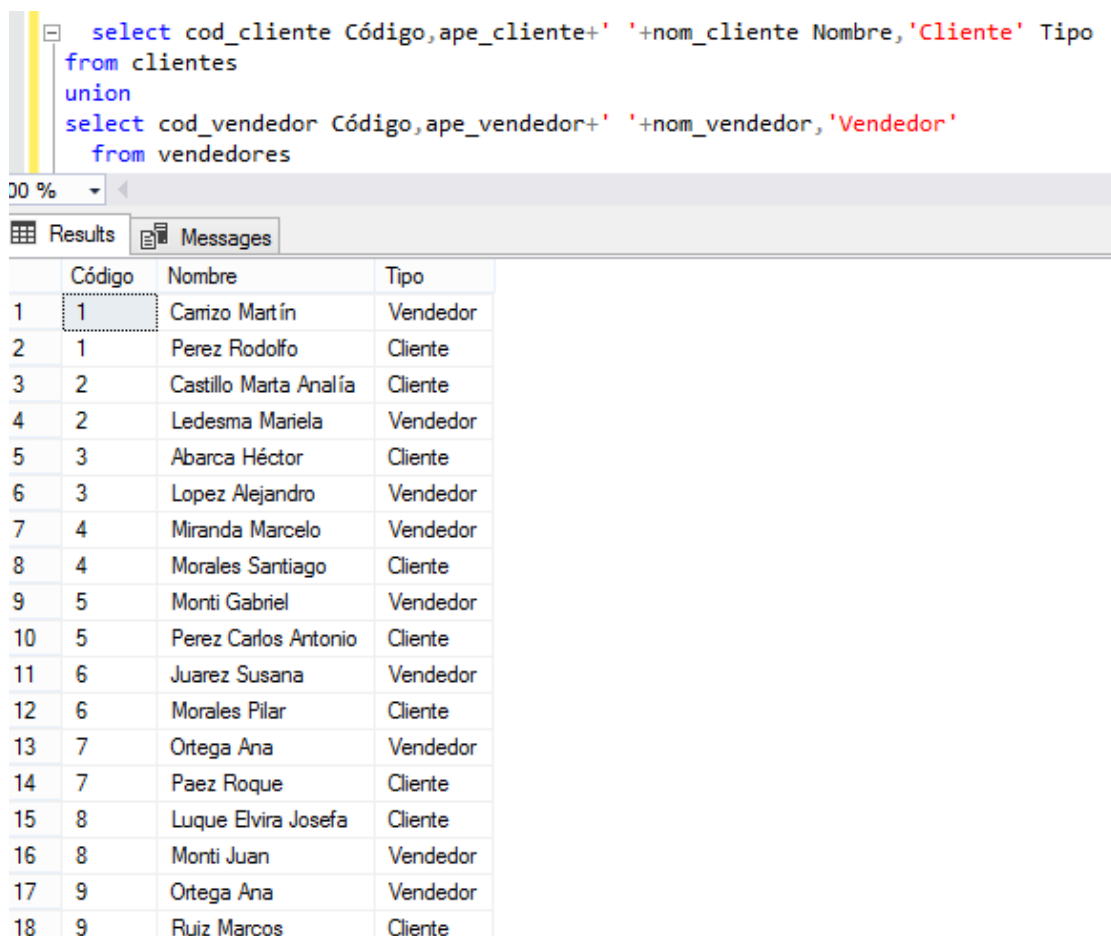
La característica **UNION** de la sentencia **SELECT** se utiliza cuando se necesita combinar los resultados de dos o más consultas en una única tabla de resultados. Por ejemplo, confeccionar un listado de los clientes y los vendedores indicando a qué grupo pertenece cada uno. Para el listado de clientes se utilizará la siguiente consulta:

```
SELECT cod_cliente as Código, ape_cliente + ' ' + nom_cliente as Nombre
FROM clientes
```

Para la de vendedores, esta otra consulta:

```
SELECT cod_vendedor as Código, ape_vendedor + ' ' + nom_vendedor as Nombre
FROM vendedores
```

Esto estaría dando dos tablas de resultados. Para que ambas consultas aparezcan en una sola tabla de resultado se escribirá lo siguiente, teniendo en cuenta que solo se agregará alias a la primera consulta y se creará una columna extra con una constante que indique el origen del registro es decir si es un cliente o



```
select cod_cliente Código,ape_cliente+' '+nom_cliente Nombre,'Cliente' Tipo
from clientes
union
select cod_vendedor Código,ape_vendedor+' '+nom_vendedor,'Vendedor'
from vendedores
```

	Código	Nombre	Tipo
1	1	Camizo Martín	Vendedor
2	1	Perez Rodolfo	Cliente
3	2	Castillo Marta Analía	Cliente
4	2	Ledesma Mariela	Vendedor
5	3	Abarca Héctor	Cliente
6	3	Lopez Alejandro	Vendedor
7	4	Miranda Marcelo	Vendedor
8	4	Morales Santiago	Cliente
9	5	Monti Gabriel	Vendedor
10	5	Perez Carlos Antonio	Cliente
11	6	Juarez Susana	Vendedor
12	6	Morales Pilar	Cliente
13	7	Ortega Ana	Vendedor
14	7	Paez Roque	Cliente
15	8	Luque Elvira Josefa	Cliente
16	8	Monti Juan	Vendedor
17	9	Ortega Ana	Vendedor
18	9	Ruiz Marcos	Cliente

un vendedor; a esta nueva columna también se le agregará un alias.

Restricciones del UNION

Hay varias restricciones sobre las consultas para que puedan combinarse con una operación **UNION**:

- Las consultas a unir deben contener el mismo número de columnas.
- El tipo de datos de cada columna en la primera consulta debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda consulta. Del mismo modo si hubiera una tercer o cuarta consulta a unir con este predicado.
- Ninguna de las consultas puede estar ordenada con la cláusula ORDER BY. Se puede ordenar el conjunto combinado que es el resultado de la operación UNION agregando esta cláusula al final de la última consulta SELECT y se debe identificar las columnas por número o alias.

En el ejemplo, cuente, la cantidad de columnas que tiene cada una de las consultas: 3 columnas cada una. Ahora observe la primera columna es un integer en ambas consultas, la segunda es varchar y la tercera varchar. Y por último si quisiera ver el listado ordenado de alguna manera, por ejemplo, primero los clientes y luego los vendedores la cláusula order by será la última línea de todas las consultas.

```
SELECT cod_cliente Código, ape_cliente + ' ' + nom_cliente Nombre, 'Cliente' Tipo
FROM clientes
UNION
SELECT cod_vendedor Código, ape_vendedor + ' ' + nom_vendedor, 'Vendedor'
FROM vendedores
ORDER BY 3
```

Cada consulta select puede incluir todas las cláusulas ya vistas anteriormente (o las que se verán posteriormente) siempre y cuando se respete las restricciones enunciadas en los párrafos anteriores.

UNION ALL

La operación UNION podría producir resultados que contuvieran filas duplicadas, pero por omisión se eliminan. Si se desea mostrar las filas duplicadas en una operación UNION, se puede especificar la palabra clave ALL luego de la palabra UNION.

Introducción a las Vistas

Una vista ofrece la posibilidad de almacenar una consulta predefinida como un objeto en una base de datos para usarse posteriormente. Las tablas consultadas en una vista se denominan tablas base. Algunos ejemplos habituales de vistas son los siguientes:

- Un subconjunto de las filas o columnas de una tabla base.
- Una unión de dos o más tablas base.
- Una combinación de dos o más tablas base.
- Un resumen estadístico de una tabla base.
- Un subconjunto de otra vista o alguna combinación de vistas y tablas base.

Ventajas de las vistas

Las vistas ofrecen diversas ventajas, como:

- Centrar el interés en los datos de los usuarios: Las vistas crean un entorno controlado que permite el acceso a datos específicos mientras se oculta el resto. Los usuarios pueden tratar la presentación de los datos en una vista de forma similar a como lo hacen en una tabla.
- Enmascarar la complejidad de la base de datos: Las vistas ocultan al usuario la complejidad del diseño de la base de datos. De este modo, los programadores pueden cambiar el diseño sin afectar a la interacción entre el usuario y la base de datos.
- Simplificar la administración de los permisos de usuario: En lugar de conceder a los usuarios permisos para consultar columnas específicas de las tablas base, los propietarios de las bases de datos pueden conceder permisos para que el usuario sólo pueda consultar los datos a través de vistas.
- Mejorar el rendimiento: Las vistas le permiten almacenar los resultados de consultas complejas. Otras consultas pueden utilizar estos resultados resumidos.
- Organizar los datos para exportarse a otras aplicaciones.

Creación de Vistas

Puede crear vistas con el Asistente para creación de vistas, con el Administrador corporativo de SQL Server o con Transact-SQL. Las vistas sólo se pueden crear en la base de datos actual. La sintaxis básica (parcial) para crear una vista es la siguiente:

```
create view NOMBREVISTA as  
    SENTENCIASSELECT  
    from TABLA;
```

En el siguiente ejemplo se crea la vista "vis_clientes", que es resultado de una combinación en la cual se muestran 4 campos:

```
create view vis_clientes  
as  
    select  (ape_cliente+' '+nom_cliente) as nombre, barrio,  
           calle, altura  
    from    clientes c  
           join barrios b  
           on c.cod_barrio=b.cod_barrio
```

La vista después de creada se trata como una tabla, es por ello que para ver el resultado de la misma se utiliza una sentencia select:

```
select * from vis_clientes;
```

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos columnas en la vista con el mismo nombre o alias).

Se crea la vista "vis_clientes_activos" que consulta los clientes que registran compras en los últimos dos años (este año y el anterior):

```
create view vis_clientes_activos (nombre,barrio)  
as  
    select  distinct ape_cliente+' '+nom_cliente, barrio  
    from    clientes c
```

```
join barrios b
on c.cod_barrio=b.cod_barrio
join facturas f
on c.cod_cliente=f.cod_cliente
Where year (fecha) >=year (getdate ()) -1
```

La diferencia es que se colocan entre paréntesis los encabezados de las columnas que aparecerán en la vista. Los nombres que se colocan entre paréntesis deben ser tantos como los campos o expresiones que se definen en la vista.

Restricciones

Existen algunas restricciones para el uso de "create view", a saber:

- no se pueden crear vistas temporales ni crear vistas sobre tablas temporales.
- no se pueden asociar reglas ni valores por defecto a las vistas.
- no puede combinarse con otras instrucciones en un mismo lote.
- Las vistas no pueden hacer referencia a más de 1.024 columnas.

Vistas encriptadas

Se puede ver el texto que define una vista ejecutando el procedimiento almacenado del sistema "**sp_helptext**" seguido del nombre de la vista:

SP_HELPTEXT nombrevista;

Para ocultar el texto que define una vista se emplea "with encryption". Si se crea la vista anterior con su definición oculta no se podrá ver su texto al usar "sp_help":

```
create view vis_clientes_activos (nombre,barrio)
with encryption
as
select distinct ape_cliente+' '+nom_cliente, barrio
from clientes c
join barrios b
on c.cod_barrio=b.cod_barrio
join facturas f
```



```
on c.cod_cliente=f.cod_cliente
Where year (fecha)>=year (getdate()) -1
```

Eliminar Vistas

Para quitar una vista se emplea:

DROP VIEW nombrevista

Si se elimina una tabla a la que hace referencia una vista, la vista no se elimina, hay que eliminarla explícitamente. Por ejemplo, para eliminar la vista denominada "vis_clientes":

```
drop view vis_clientes;
```

Modificar Vistas

Para modificar una vista se puede hacer con "alter view". En el ejemplo siguiente se altera vis_clientes_activos para agregar la calle y altura:

```
alter view vis_clientes_activos (nombre,barrio,calle,altura)
as
select distinct ape_cliente+' '+nom_cliente, barrio,
               calle, altura
from   clientes c
       join barrios b
       on c.cod_barrio=b.cod_barrio
       join facturas f
       on c.cod_cliente=f.cod_cliente
Where   year (fecha)>=year (getdate()) -1
```

Si crea una vista con "select *" y luego agrega campos a la estructura de las tablas involucradas, los nuevos campos no aparecerán en la vista; esto es porque los campos se seleccionan al ejecutar "create view"; debe alterar la vista.

Modificar datos de una tabla a través de vistas

Si se modifican los datos de una vista, se modifica la tabla base. Se puede insertar, actualizar o eliminar datos de una tabla a través de una vista, teniendo en cuenta lo siguiente, las modificaciones que se realizan a las vistas:

- no pueden afectar a más de una tabla consultada.
- no se pueden cambiar los campos resultado de un cálculo.
- pueden generar errores si afectan a campos a las que la vista no hace referencia.

Tablas temporales

Las tablas temporales son visibles solamente en la sesión actual. Se eliminan automáticamente al terminar la sesión, la función o procedimiento almacenado en el cual fueron definidas. Aunque también se pueden eliminar con "drop table".

Pueden ser locales (son visibles sólo en la sesión actual) o globales (visibles por todas las sesiones).

Para crear tablas temporales locales se emplea la misma sintaxis que para crear cualquier tabla, excepto que se coloca un signo numeral (#) precediendo el nombre.

Sintaxis:

```
create table #NOMBRE(  
    CAMPO DEFINICION,  
    ...  
);
```

Para referenciarla en otras consultas, se debe incluir el numeral(#), que es parte del nombre. Por ejemplo:

```
insert into #personas default values;  
select *from #personas;
```

Una tabla temporal no puede tener una restricción "foreign key" ni ser indexada, tampoco puede ser referenciada por una vista.

Para crear tablas temporales globales se emplea la misma sintaxis que para crear cualquier tabla, excepto que se coloca un signo numeral doble (##) precediendo el nombre.

```
create table ##NOMBRE(  
    CAMPO DEFINICION,  
    ...  
);
```

El (o los) numerales son parte del nombre. Así que puede crearse una tabla permanente llamada "articulos", otra tabla temporal local llamada "#articulos" y una tercera tabla temporal global denominada "##articulos".

No podemos consultar la tabla "sysobjects" para ver las tablas temporales, debemos tipear:

```
select * from tempdb.sysobjects;
```

BIBLIOGRAFÍA

Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D. & Woody B (2019) Introducing Microsoft SQL Server 2019. Reliability, scalability, and security both on premises and in the cloud. Packt Publishing Ltd. Birmingham UK

Microsoft (2021) SQL Server technical documentation. Disponible en: <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15>

Opel, A. & Sheldon, R. (2010). Fundamentos de SQL. Madrid. Editorial Mc Graw Hill

Varga S., Cherry D., D'Antoni J. (2016). Introducing Microsoft SQL Server 2016 Mission-Critical Applications, Deeper Insights, Hyperscale Cloud. Washington. Microsoft Press



Atribución-No Comercial-Sin Derivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Referenciarlo de la siguiente manera:
Universidad Tecnológica Nacional Facultad Regional Córdoba (S/D). Material para la Tecnicatura Universitaria en Programación, modalidad virtual, Córdoba, Argentina.