

Bases de datos relacionales

DML

(Teórica)



Introducción

Bienvenidos al módulo de bases de datos relacionales, SQL DML. En este módulo trabajaremos en conceptualizar las bases de datos relacionales. Nos focalizaremos en las sentencias básicas para la manipulación de datos en SQL.

Una vez finalizado este módulo serás capaz de:

- Conocer la forma genérica de una consulta SQL.
- Trabajar con operaciones de junta como: Inner Join - Left/Right Join y Outer Join.
- Trabajar con operadores como: Union, Union All, Intersect, Except.
- Realizar consultas Anidadas con: In, Not In, Exists, Not Exists.
- Trabajar con Group By, Having y Order By.
- Utilizar funciones agregadas como: Count, Avg, Sum, Min, Max.



Tema 1. SQL DML

Objetivo

Bienvenidos al módulo de bases de datos relacionales, SQL DML. En este módulo trabajaremos con las sentencias básicas del subset DML del lenguaje SQL.

Una vez finalizado este módulo serás capaz de:

- Realizar consultas básicas en SQL con los siguientes operadores: In, Not In, Exists, Not Exists. Group By, Having, Order By. Funciones agregadas: Count, Avg, Sum, Min, Max.
- Realizar juntas de diferentes tipos como Inner Join, Left Join y Outer Join.
- Crear vistas

SCRIPT DE CREACIÓN y DINÁMICA

Para trabajar con este módulo, utilizaremos el siguiente script de creación de una base de datos con 7 tablas. Para comenzar debemos abrir SQL Server Management Studio, abrir una ventana de consulta con el botón “New Query”, copiar y pegar en la nueva query el siguiente texto:

-- SELECCIONAR DESDE ESTE PUNTO

-- Creación de la base de datos

create database SQL_DML;

GO

--creación de las tablas

use SQL_DML;

GO

create table Almacen (Nro int primary key, Responsable varchar(50))

create table Articulo (CodArt int primary key, Descripcion varchar(50),
Precio decimal(12, 3))

create table Material (CodMat int primary key, Descripcion
varchar(50))

create table Proveedor (CodProv int primary key, Nombre varchar(50),
Domicilio varchar(50), Ciudad varchar(50), fecha_alta date)

create table Tiene (NroAlmacen int foreign key references
almacen(Nro), CodArt int foreign key references articulo(codArt))

create table Compuesto_Por (CodArt int foreign key references
articulo(codArt), CodMat int foreign key references material(codMat))

create table Provisto_Por (CodMat int foreign key references
material(codMat), CodProv int foreign key references
proveedor(codProv))

GO

-- carga de datos

insert into Almacen values (1, 'Juan Perez'), (2, 'Jose Basualdo'), (3, 'Pablo Mancineli'), (4, 'Rogelio Funes Mori'), (5, 'Jonathan Maidana'), (6, 'Anita Martinez'), (7, 'Patricio Gonzalez'), (8, 'Diego Salaberry')

insert into Artículo values (1, 'Artículo Uno', 60), (2, 'Artículo Dos', 60), (3, 'Artículo Tres', 180), (4, 'Artículo Cuatro', 250), (5, 'Artículo Cinco', 350), (6, 'Artículo Seis', 450), (7, 'Artículo Siete', 650), (8, 'Artículo Ocho', 850), (9, 'Artículo Nueve', 1250)

insert into Material values (1, 'Material A'), (2, 'Material B'), (3, 'Material C'), (4, 'Material D'), (5, 'Material E'), (6, 'Material F'), (7, 'Material G'), (8, 'Material H'), (9, 'Material I'), (10, 'Material J'), (11, 'Material K'), (12, 'Material L'), (13, 'Material M'), (14, 'Material P')

insert into Proveedor values

(1, 'Proveedor Uno', 'Carlos Calvo 1212', 'CABA','01-01-1990'),

(2, 'Proveedor Dos', 'San Martín 121', 'Pergamino','01-01-1990'),

(3, 'Proveedor Tres', 'San Pedrito 1244', 'CABA','01-01-1992'),

(4, 'Proveedor Cuatro', 'Av. Boedo 3232', 'CABA','01-01-1993'),

(5, 'Proveedor Cinco', '5 3232', 'La Plata','01-01-1994'),

(6, 'Proveedor Seis', 'Av 7 287', 'La Plata','01-01-1995'),

(7, 'Proveedor Siete', 'Italia 954', 'San Fernando','01-01-1996'),

(8, 'Proveedor Ocho', 'María de Carmen 765', 'San Fernando del Valle de Catamarca','01-01-1997'),

(9, 'Proveedor Nueve', 'Av Corrientes 1200', 'Corrientes Capital','01-01-1998'),

(10, 'Proveedor Diez', 'Av Juan Manuel de Rosas', 'San Justo','01-01-1999'),

(11, 'Proveedor Once', 'Av Marite García 220', 'Ramos Mejía','01-01-1990'),

(12, 'Proveedor Doce', 'La Plata 343', 'Ciudadela','01-01-2007'),

(13, 'Proveedor Trece', 'Independencia 1343', 'Mar del Plata','01-01-2008'),

(14, 'Proveedor Catorce', 'Varela 343', 'San Justo','01-01-2009'),

(15, 'Proveedor Quince', 'Santander 943', 'Ituzaingó','01-01-2010')

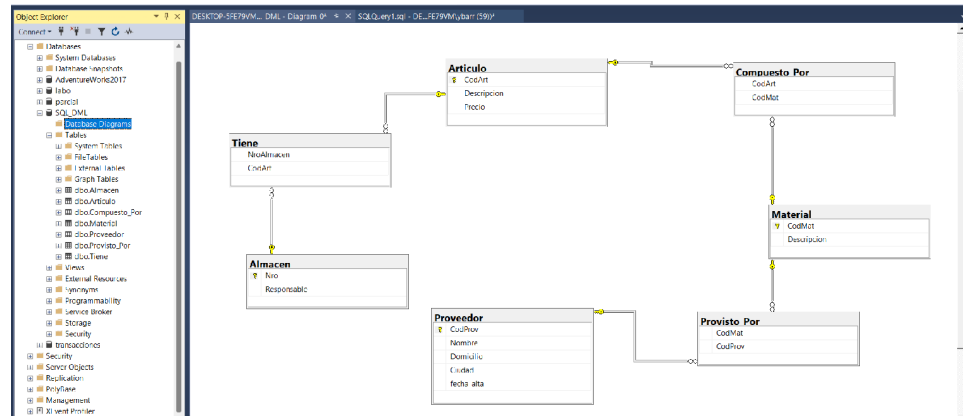
insert into Tiene values (1, 1), (1, 2), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 3), (4, 5), (4, 8), (5, 2), (5, 7), (5, 8), (5, 9), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8), (8, 9)

insert into Compuesto_Por values (1, 1), (1, 2), (1, 3), (2, 4), (2, 5), (3, 1), (3, 6), (4, 1), (4, 6), (4, 7), (4, 8), (9,12)

insert into Provisto_Por values (1, 1), (1, 3), (1, 6), (2, 2), (2, 3), (2, 4),(2, 6), (3, 2), (3, 3),(3, 6), (4, 3), (4, 4),(4, 6), (5, 1), (5, 3),(5, 6), (6, 3), (6, 4),(6, 6), (7, 3), (7, 5),(7, 6), (8, 3), (8, 5), (8, 6),(8, 10),(8, 15), (9, 6),(10, 6),(11, 6),(12, 6),(13, 6),(13, 10), (14, 10),(14, 6),(14, 15)

-- SELECCIONAR HASTA ESTE PUNTO

Una vez pegado el texto presionamos el botón “Execute” o F5. Se ejecutará el script y se creará la base de datos. El esquema de esta quedará según la imagen que se muestra a continuación:



Una vez verificada la correcta creación de la base de datos, nos focalizamos en algunas problemáticas que nos permitan aprender la utilización de algunas sentencias SQL para solucionar las mismas.

ESTRUCTURA BÁSICA DE UNA CONSULTA SQL

Una Query en SQL posee la siguiente estructura básica:

SELECT campo1, campo2,, campo_n

FROM tabla1

WHERE condición1 AND/OR condicion2

Ahora bien, analicemos que se realiza en cada una de las líneas:

Una cláusula **SELECT** se utiliza con el fin de especificar los nombres de los campos que contienen los datos que quiere usar en una consulta.

Una cláusula **FROM** se utiliza con el fin de especificar la Tabla a utilizar para poder listar los campos especificados en el **SELECT**.

En una query simple, son obligatorios el SELECT y el FROM, siendo el WHERE una cláusula opcional.

Entonces la cláusula WHERE se utiliza con el fin de filtrar los datos resultantes de la consulta, a través de una o varias condiciones establecidas en la misma. Veamos un ejemplo de una query simple con la base de datos creada:

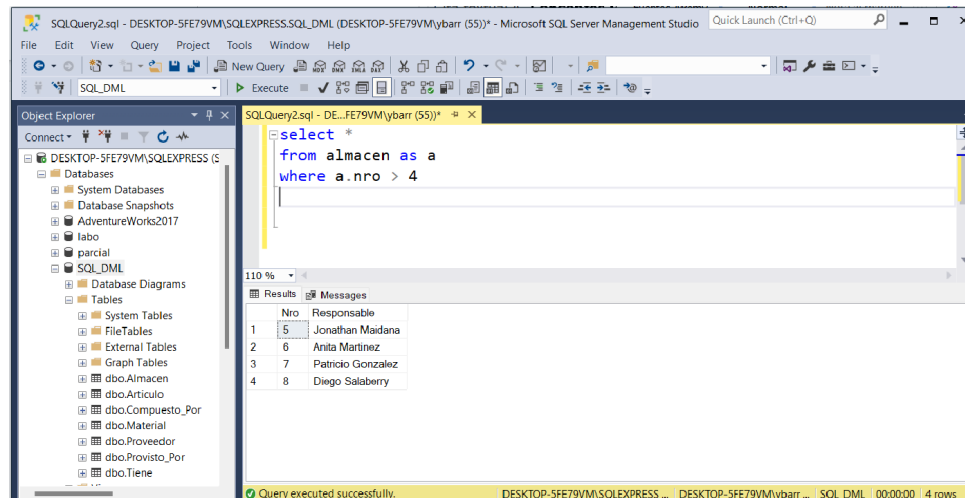
```
select *  
  
from almacen as a  
  
where a.nro > 4
```

La misma mostrará todos los campos de las tuplas que cumplan la condición cuyo número de almacén sean mayor a 4.

Ahora bien, analicemos un poco la misma:

- El * en select indica que deben listarse todos los campos de la tabla que aparece en el FROM.
- El “as a” en el FROM se conoce como ALIAS de la tabla almacén. Eso quiere decir que con solo mencionar el alias a, ya estaremos haciendo mención a la misma tabla.
- El “a.nro” en el WHERE, refiere a la tabla almacen (por el alias “a”) y luego, al campo de dicha tabla.

A continuación, se muestra pantalla de resultados:



Ahora que conocemos la estructura básica de las consultas SQL, avancemos sobre situaciones más complejas. A partir de este punto, la dinámica será la siguiente: se planteará un pedido sobre los datos de la base de datos y se indicará la consulta SQL que da solución a ese pedido, analizando en detalle cómo funciona esa consulta. Avancemos.

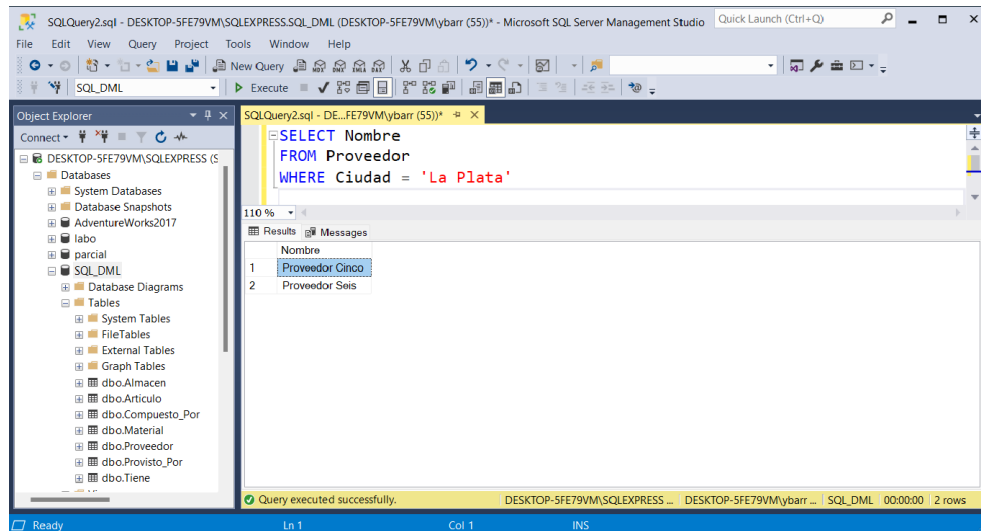
Cadena de texto en condición del WHERE

Se pide, listar los nombres de los proveedores de la ciudad de La Plata.

La consulta que resuelve el pedido es la siguiente:

```
SELECT Nombre
FROM Proveedor
WHERE Ciudad = 'La Plata'
```

Como vemos, podemos utilizar una cadena de caracteres para comprar contra un campo en el WHERE. El resultado mostrado es el siguiente:



Uso del Operador LIKE

LIKE es un tipo de operador lógico que se usa para poder determinar si una cadena de caracteres específica coincide con un patrón específico. Se utiliza normalmente en una sentencia Where para buscar un patrón específico de una columna.

Este operador puede ser de utilidad en los casos donde necesitamos realizar un apareamiento de patrones en vez de iguales o no iguales. El SQL Like se utiliza cuando se desea devolver una o varias filas, si una cadena de caracteres específica coincide con un patrón específico.

Con LIKE podemos usar el comodín “%” el cual indica que puede haber 0 o varios caracteres en su posición. También puede usarse el comodín “_” el cual indica que puede haber un carácter cualquiera donde el mismo esté ubicado. Entenderemos mejor su funcionamiento con el siguiente ejemplo:

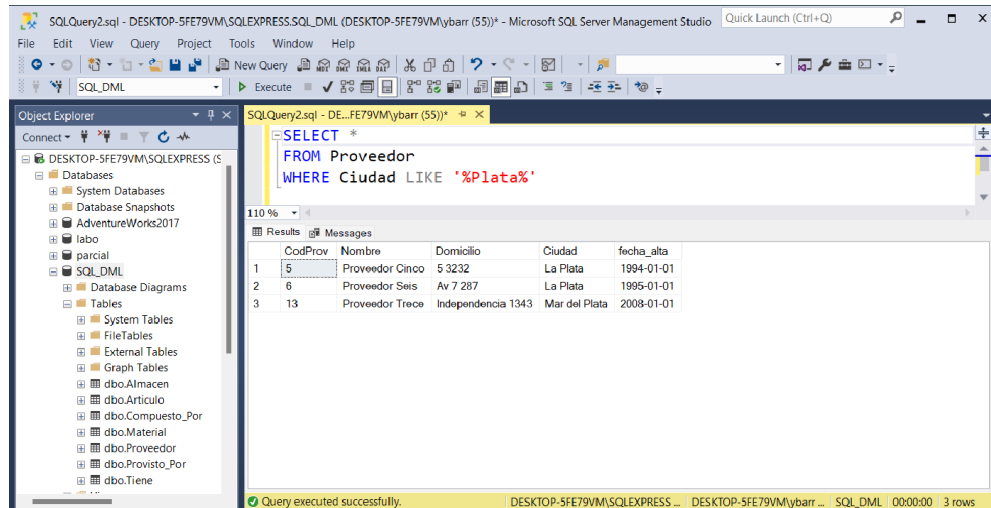
Se pide, Listar los proveedores cuya localidad contenga la cadena de texto “Plata”

La consulta que resuelve el pedido es la siguiente:

```
SELECT *
```

```
FROM Proveedor
WHERE Ciudad LIKE '%Plata%'
```

El resultado de su ejecución es el siguiente:



Como Podemos ver en las tuplas resultantes, se muestran los proveedores que contengan la cadena de caracteres “Plata” en el nombre de la ciudad, ya que los comodines % al inicio y al final de la cadena “Plata” hacen que no importe como inicie o como termine el nombre de la ciudad, se tomarán en cuenta aquellas tuplas que contengan la cadena “Plata”

CONSULTAS ANIDADAS: NOT IN / IN

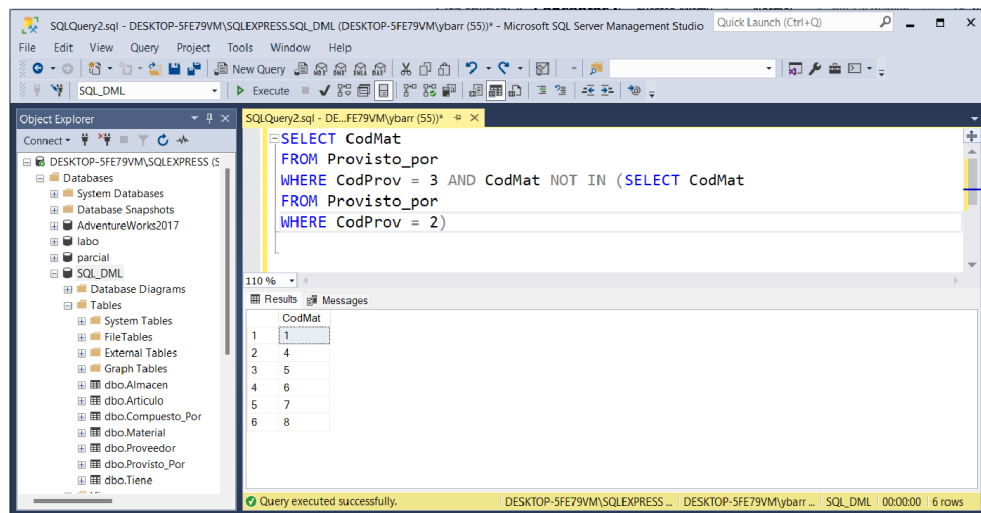
NOT IN es utilizado cuando se necesita determinar los valores que no están en una lista. En cambio IN, se utiliza cuando se necesita determinar los valores que si están en una lista. Veamos un ejemplo de cómo utilizar NOT IN:

Se pide, listar los códigos de los materiales que provea el proveedor 3 y no los provea el proveedor 2.

La consulta que resuelve el pedido es la siguiente:

```
SELECT CodMat
FROM Provisto_por
WHERE CodProv = 3 AND CodMat NOT IN (SELECT CodMat
FROM Provisto_por
WHERE CodProv = 2)
```

El resultado de su ejecución es el siguiente:



Como vemos, el operador NOT IN se utiliza en el WHERE y se compara el campo codMat contra una lista resultante que surge de una consulta anidada (La misma se encuentra entre paréntesis y es la que devuelve los códigos de materiales provistos por el proveedor 2). Algo importante a tener en cuenta es que NOT IN compara los mismos tipos de datos, entonces, dentro de la subconsulta no podría haber un *, ya que se listarían todos los campos y NOT IN no puede comparar datos incompatibles.

CONSULTAS ANIDADAS: NOT EXISTS

La cláusula NOT EXISTS / EXISTS se utiliza para filtrar los resultados de una consulta SQL. Esta cláusula se usa junto con una subconsulta, que es una consulta SQL que se ejecuta dentro de otra consulta SQL. La subconsulta se evalúa primero.

En el caso de EXISTS, si la subconsulta devuelve algún resultado, se ejecutará la consulta SQL principal. Si la subconsulta no devuelve ningún resultado, la cláusula EXISTS no se ejecutará y no se mostrarán los resultados de la consulta SQL principal.

En cambio, en el caso de NOT EXISTS, Si la subconsulta no devuelve ningún resultado, la cláusula EXISTS si se ejecutará y se mostrarán los resultados de la consulta SQL principal. A diferencia de NOT IN, la consulta externa y la sobconsulta deben estar vinculadas.

Para probar el funcionamiento de NOT EXISTS, nos sirve el mismo requerimiento que el punto anterior. Veamos dar cumplimiento al pedido anterior pero utilizando NOT EXISTS:

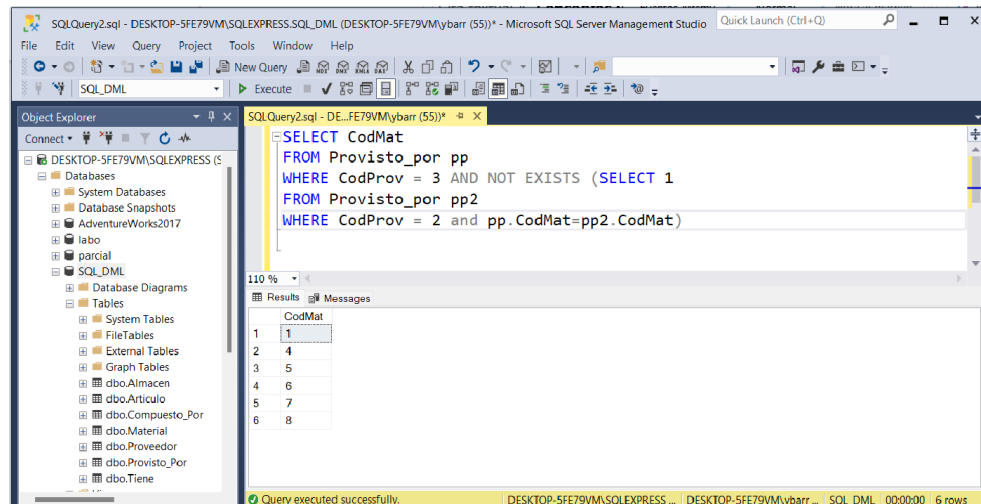
Se pide, listar los códigos de los materiales que provea el proveedor 3 y no los provea el proveedor 2.

La consulta que resuelve el pedido es la siguiente:

```
SELECT CodMat
FROM Provisto_por pp
WHERE CodProv = 3 AND NOT EXISTS (SELECT 1
FROM Provisto_por pp2
WHERE CodProv = 2 and pp.CodMat=pp2.CodMat)
```

Como vemos en este caso, la subconsulta está vinculada a la query externa por el campo codMat, el cual es el que debe listarse en el select externo.

El resultado de su ejecución es el siguiente:



Como vemos, el resultado es el mismo que si trabajamos con NOT IN.

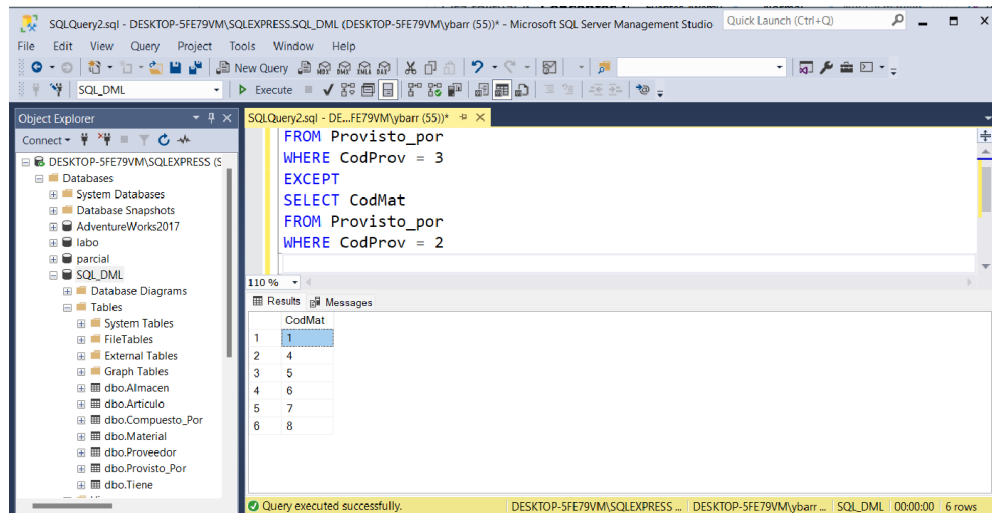
En este sentido, podemos resolver el mismo requerimiento usando el operador EXCEPT. El mismo permite trabajar con dos conjuntos de valor y devuelve aquellas tuplas que están en el primer conjunto, pero no en el segundo. El requerimiento, entonces es el mismo:

Se pide, listar los códigos de los materiales que provea el proveedor 3 y no los provea el proveedor 2.

La consulta que resuelve el pedido es la siguiente:

```
SELECT CodMat
FROM Provisto_por
WHERE CodProv = 3
EXCEPT
SELECT CodMat
FROM Provisto_por
WHERE CodProv = 2
```

Su ejecución nos devuelve lo siguiente:



Como vemos, el resultado es similar a NOT IN y NOT EXISTS. Como podemos ver, luego de analizar estos operadores, nos damos cuenta que son los operadores que debemos utilizar para realizar restas de conjuntos.

JUNTAS ENTRE TABLAS

INNER JOIN

El JOIN o INNER JOIN es una operación que hace coincidir los registros de una tabla con los registros de otra, a partir de uno o más campos, de manera que las columnas puedan ser colocadas lado a lado en el resultado de la consulta, simulando que vienen de una sola tabla.

La operación JOIN realiza una unión de tablas por medio de columnas en común.

Veamos un ejemplo:

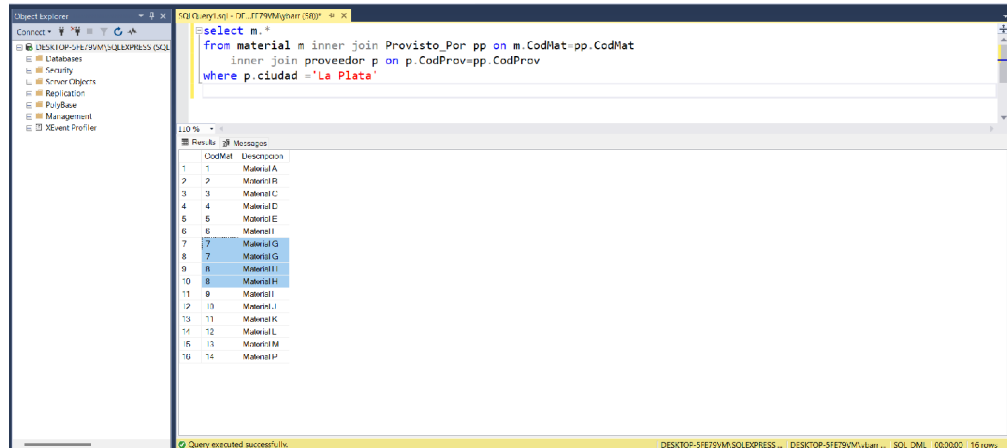
Se pide, Listar los materiales, código y descripción, provistos por proveedores de la ciudad de La Plata.

La consulta que resuelve el pedido es la siguiente:

```
select m.*
```

```
from material m inner join Provisto_Por pp on
m.CodMat=pp.CodMat
    inner join proveedor p on p.CodProv=pp.CodProv
where p.ciudad ='La Plata'
```

El resultado de su ejecución es el siguiente:

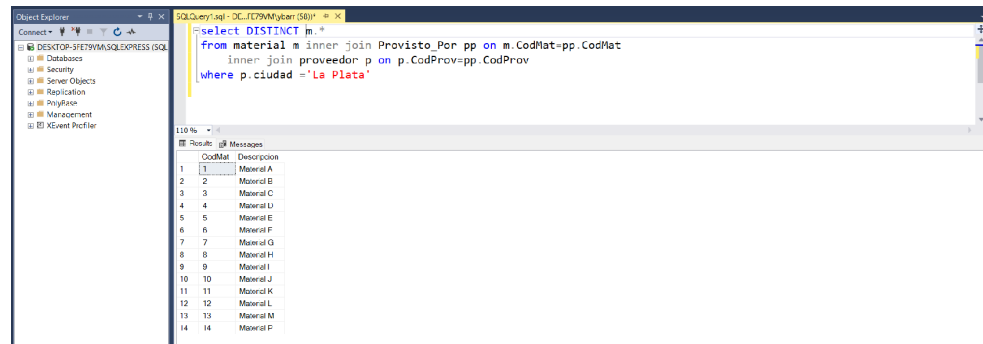


CodMat	Descripcion
1	Material A
2	Material B
3	Material C
4	Material D
5	Material E
6	Material F
7	Material G
8	Material H
9	Material I
10	Material J
11	Material K
12	Material L
13	Material M
14	Material N
15	Material O
16	Material P

Como vemos en el resultado, hay algunas tuplas que se repiten y que surgen de realizar la junta. Ahora bien, cómo podemos evitar estos resultados repetidos? Para ello utilizaremos en el select el operador DISTINCT. El mismo sirve para evitar que aparezcan tuplas duplicadas, entonces, modificamos la sentencia anterior, a la siguiente:

```
select DISTINCT m.*
from material m inner join Provisto_Por pp on
m.CodMat=pp.CodMat
    inner join proveedor p on p.CodProv=pp.CodProv
where p.ciudad ='La Plata'
```

Ahora, el resultado es el siguiente:



Como vemos, ya no aparecen tuplas duplicadas en el resultado.

LEFT JOIN

Otro tipo de JOIN muy utilizado es el LEFT JOIN que retorna todos los registros de la primera tabla más los registros de la segunda tabla cuyo valor para el campo de cruce coincida con el de la primera tabla.

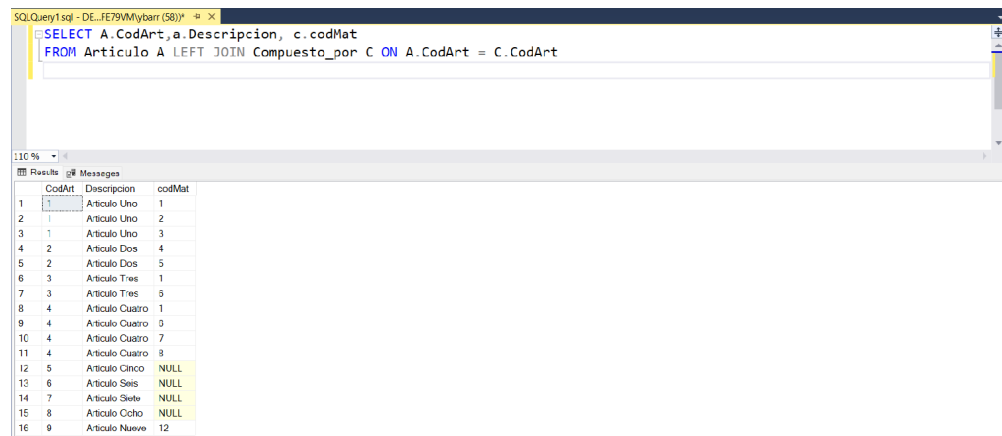
Veamos un ejemplo:

Listar todos los artículos y los materiales por los que están compuestos, informando "null" en el campo "material" para el caso de los artículos que no están compuestos por ningún material

La consulta que resuelve el pedido es la siguiente:

```
SELECT A.CodArt,a.Descripcion, c.codMat
FROM Articulo A LEFT JOIN Compuesto_por C ON
A.CodArt = C.CodArt
```

El resultado de su ejecución es el siguiente:



Como vemos en el resultado, LEFT JOIN mostrará las tuplas de la tabla a la izquierda (Articulo), incluso para aquellos casos en donde no puede realizar la junta por igualdad contra la tabla compuesto_por.

RIGHT JOIN

El RIGHT JOIN, a diferencia del LEFT JOIN, retorna todos los registros de la segunda tabla más los registros de la primera tabla cuyo valor para el campo de cruce coincida con el de la primera tabla.

FULL OUTER JOIN

FULL OUTER JOIN retorna todos los registros de la primera tabla y la segunda tabla incluso para aquellos casos en donde no coincida el atributo de junta.

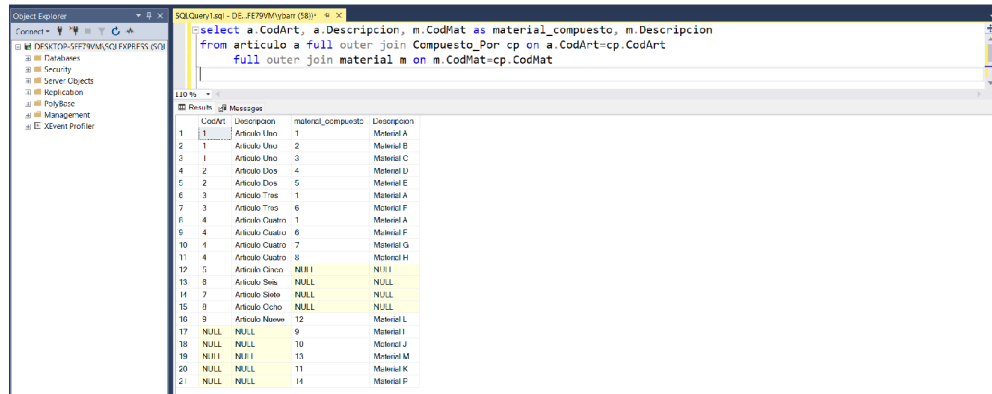
Veamos un ejemplo:

Se pide, Listar todos los artículos y materiales por los cuales están compuestos. Mostrar artículos sin materiales y Materiales que no componen ningún artículo

La consulta que resuelve el pedido es la siguiente:

```
select a.CodArt, a.Descripcion, m.CodMat as
material_compuesto, m.Descripcion
from articulo a full outer join Compuesto_Por cp on
a.CodArt=cp.CodArt
full outer join material m on m.CodMat=cp.CodMat
```

El resultado de su ejecución es el siguiente:



LocArt	Descripción	material_compuesto	Descripción
1	Artículo Uno	1	Material A
2	Artículo Uno	2	Material B
3	Artículo Uno	3	Material C
4	Artículo Uno	4	Material D
5	Artículo Dos	5	Material E
6	Artículo Tres	1	Material A
7	Artículo Tres	6	Material F
8	Artículo Cuatro	1	Material A
9	Artículo Cuatro	6	Material F
10	Artículo Cuatro	7	Material G
11	Artículo Cuatro	8	Material H
12	Artículo Cinco	NULL	NULL
13	Artículo Seis	NULL	NULL
14	Artículo Seis	NULL	NULL
15	Artículo Cinco	NULL	NULL
16	Artículo Nueve	12	Material L
17	NULL	9	Material I
18	NULL	10	Material J
19	NULL	13	Material M
20	NULL	11	Material K
21	NULL	14	Material P

1-5. Listar los números de almacenes que almacenan el artículo cuyo código es 1.

UNION

El operador de Unión combina los resultados de dos o más consultas dando lugar a la creación de un único conjunto de resultados que incluye todas las filas que pertenecen a todas las consultas en la Unión. En esta operación, combina dos consultas y elimina los duplicados.

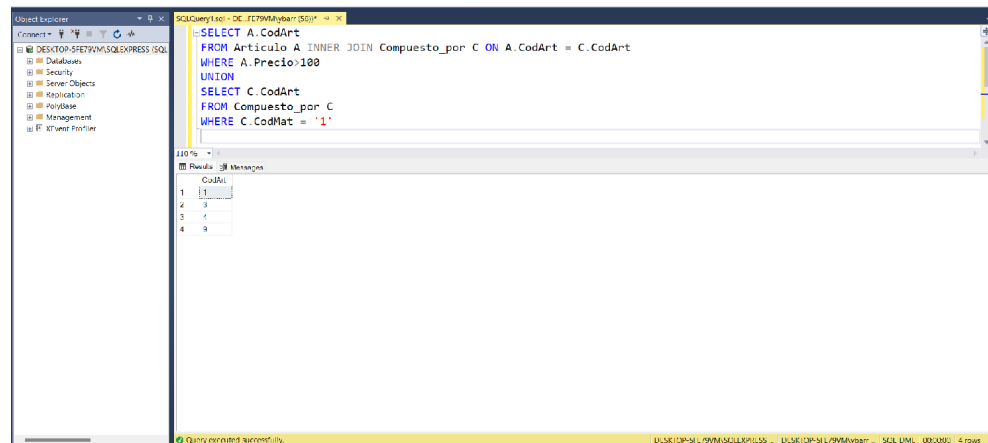
Veamos un ejemplo:

Se pide, Listar los artículos que cuesten más de \$100 o que estén compuestos por el material 1.

La consulta que resuelve el pedido es la siguiente:

```
SELECT A.CodArt
FROM Articulo A INNER JOIN Compuesto_por C ON
A.CodArt = C.CodArt
WHERE A.Precio>100
UNION
SELECT C.CodArt
FROM Compuesto_por C
WHERE C.CodMat = '1'
```

El resultado de su ejecución es el siguiente:



UNION ALL

El operador de Unión combina los resultados de dos o más consultas dando lugar a la creación de un único conjunto de resultados que incluye todas las filas que pertenecen a todas las consultas en la Unión. En esta operación, combina dos consultas y PERO NO elimina los duplicados.

Veamos el mismo ejemplo anterior pero con UNION ALL:

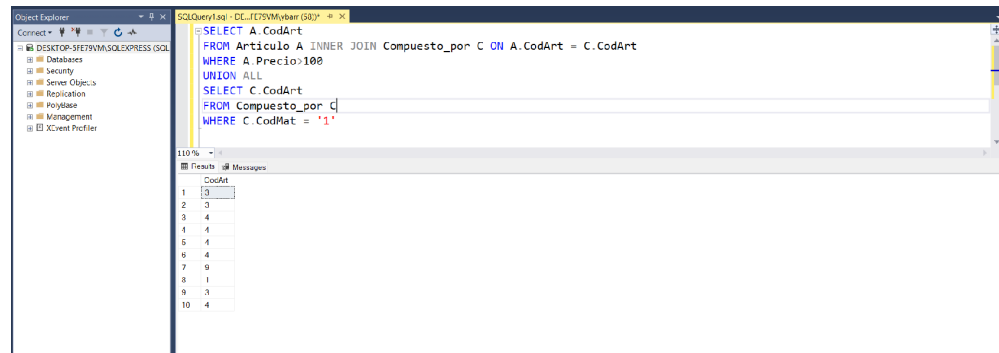
Se pide, Listar los artículos que cuesten más de \$100 o que estén compuestos por el material 1 (Sin eliminar duplicados).

La consulta que resuelve el pedido es la siguiente:

```

SELECT A.CodArt
FROM Articulo A INNER JOIN Compuesto_por C ON
A.CodArt = C.CodArt
WHERE A.Precio>100
UNION ALL
SELECT C.CodArt
FROM Compuesto_por C
WHERE C.CodMat = '1'
    
```

El resultado de su ejecución es el siguiente:



OPERADORES MIN, MAX, AVG

Estos operadores se usan en el select y tiene la siguiente sintaxis:

MIN (expr): En este caso se obtiene el valor mínimo de la expresión.

MAX (expr) : En este caso se obtiene el valor máximo de la expresión.

AVG (expr) : En este caso se obtiene el valor promedio de la expresión.

El marcador de posición de *expr* representa una expresión de cadena el campo que contiene los datos que desea evaluar o una expresión que realiza un cálculo con los datos de ese campo. Los operandos en *expr* pueden incluir el nombre de un campo de tabla, una constante o una función (que puede ser intrínseca o definida por el usuario, pero no una de las otras SQL de agregado). En general es utilizado con el nombre del campo.

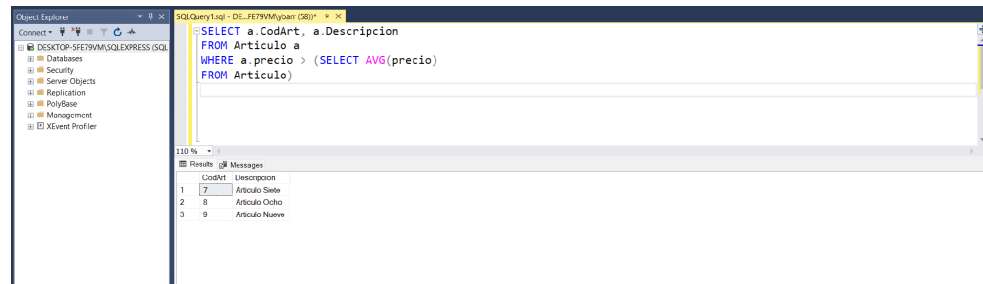
Veamos un ejemplo con Average:

Se solicita, listar los artículos cuyo precio superan la media.

La consulta que resuelve el pedido es la siguiente:

```
SELECT a.CodArt, a.Descripcion
FROM Articulo a
WHERE a.precio > (SELECT AVG(precio)
FROM Articulo)
```

El resultado de su ejecución es el siguiente:



GROUP BY

La sentencia GROUP BY utiliza una o varias columnas para con el fin de agrupar resultados. Divide los datos en grupos en función a los valores de la columna especificada, y devuelve una fila de resultados para cada grupo.

Es posible utilizar GROUP BY con más de un nombre de columna y los mismos deben separarse con comas). El operador GROUP BY se utiliza luego del FROM o WHERE en una consulta, y antes de HAVING y ORDER BY.

El operador GROUP BY acumula los resultados por grupo, pero no necesariamente ordena los grupos; para ordenarlos se necesita utilizar una sentencia ORDER BY. Veremos ORDER BY más adelante en este documento.

En general, GROUP BY produce una fila para cada distinto valor de la columna especificada en la cláusula GROUP BY. Cuando se mencionan varias columnas en la cláusula GROUP BY, se produce una fila nueva en el informe cada vez que cambia un valor en una de las columnas.

Todas las columnas del SELECT sin una función de agregación asociada deben aparecer en la cláusula GROUP BY. Pero las columnas que se colocan en el GROUP BY no necesariamente deben aparecer en el SELECT

En párrafos anteriores mencionamos a las funciones de agregación. Pero... cómo operan estas funciones en un group by?

Una función agregada es utilizada en el group by para trabajar sobre las filas resultantes del agrupamiento. Por ejemplo, podemos utilizar COUNT, COUNT DISTINCT, MAX, MIN, AVG, etc para operar sobre las tuplas resultantes del agrupamiento propuesto.

FILTRANDO RESULTADOS DEL GROUP BY

La palabra clave HAVING debe utilizarse con datos agrupados. Cuando la sentencia HAVING y la sentencia GROUP BY se utilizan juntas, la sentencia HAVING debe seguir a la sentencia GROUP BY.

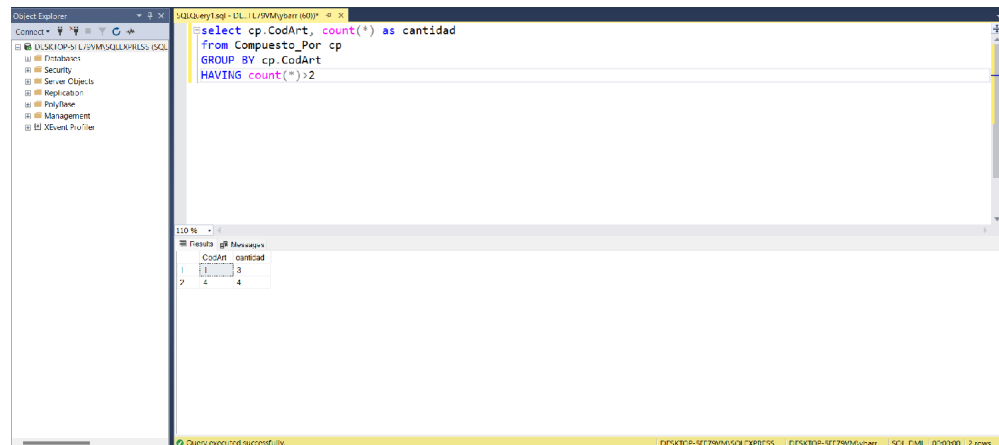
Veamos un ejemplo de group by con Having

Se pide, Listar los códigos de los artículos compuestos por más de 2 materiales.

La consulta que da respuesta a lo pedido es la siguiente:

```
select cp.CodArt, count(*) as cantidad
from Compuesto_Por cp
GROUP BY cp.CodArt
HAVING count(*)>2
```

El resultado de la ejecución es el siguiente:



Aquí vemos como se agrupó la tabla compuesto_por a través del campo código de artículo, para contar las tuplas existentes luego del agrupamiento para cada uno de los materiales (Esto indica la cantidad de materiales por los que está compuesto). Finalmente, utilizamos HAVING para filtrar aquellos casos donde el COUNT(*) es mayor a 2.

ORDER BY

La cláusula ORDER BY se utiliza para ordenar las tuplas resultantes de una consulta en orden ascendente o descendente. Por defecto, ORDER BY ordena los resultados de forma ascendente. Para solicitar un ordenamiento en formato descendente utilizaremos: ORDER BY nombre_campo DESC

Vemos como cambia el resultado de la consulta del punto anterior utilizando la siguiente query:

```
select cp.CodArt, count(*) as cantidad

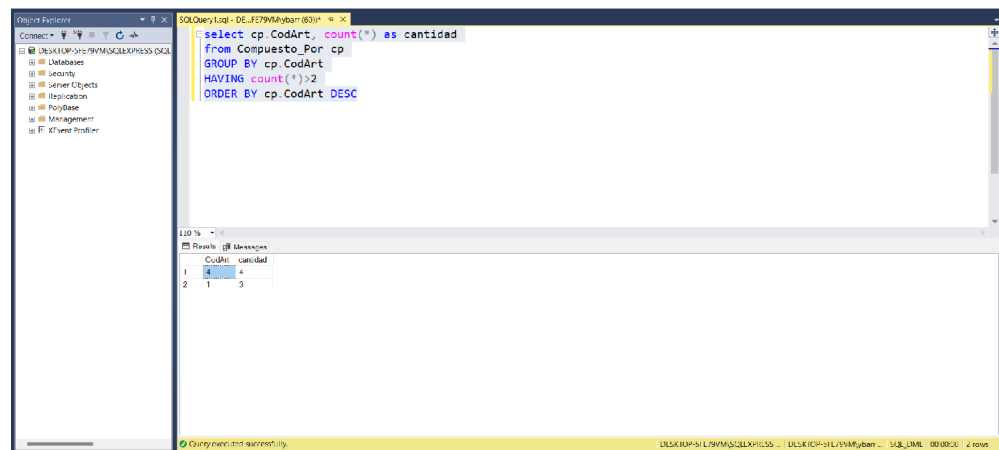
from Compuesto_Por cp
```


GROUP BY cp.CodArt

HAVING count(*)>2

ORDER BY cp.CodArt DESC

El resultado de la ejecución es el siguiente:



VISTAS

Podemos definir a una VISTA como una consulta almacenada.

Entonces, una vista es una tabla virtual cuyo contenido está definido por una consulta ya escrita. Al igual que una tabla, una vista es un objeto de nuestra base de datos y consta de un conjunto de columnas con un nombre

Una vista actúa como una abstracción de las tablas subyacentes a las que se hace referencia en ella. La consulta que define la vista puede provenir de una o de varias tablas (definidas por algún tipo de JOIN), o bien de otras vistas de la base de datos.

Las vistas suelen usarse para simplificar la complejidad de la base de datos para cada usuario. En este sentido, las vistas pueden emplearse como mecanismos de seguridad, ya que permiten a los

usuarios acceder a los datos por medio de la vista, pero no se les concede permisos a las tablas base de la vista.

Las vistas se crean con la siguiente sentencia:

```
CREATE VIEW nombre_vista AS ( consulta_SQL)
```

Veamos un ejemplo de una vista para que quede más claro su funcionamiento. Vamos utilizar la consulta del GROUP by para crear una vista, ya que no queremos estar recordando cómo agrupar y contar cada vez que necesitamos solucionar un pedido de ese estilo. Entonces, crearemos una vista para “Almacenar” esa consulta y utilizarla como una tabla. Para ello, ejecutamos lo siguiente:

```
CREATE VIEW v_articulos_con_mas_de_dos_materiales as

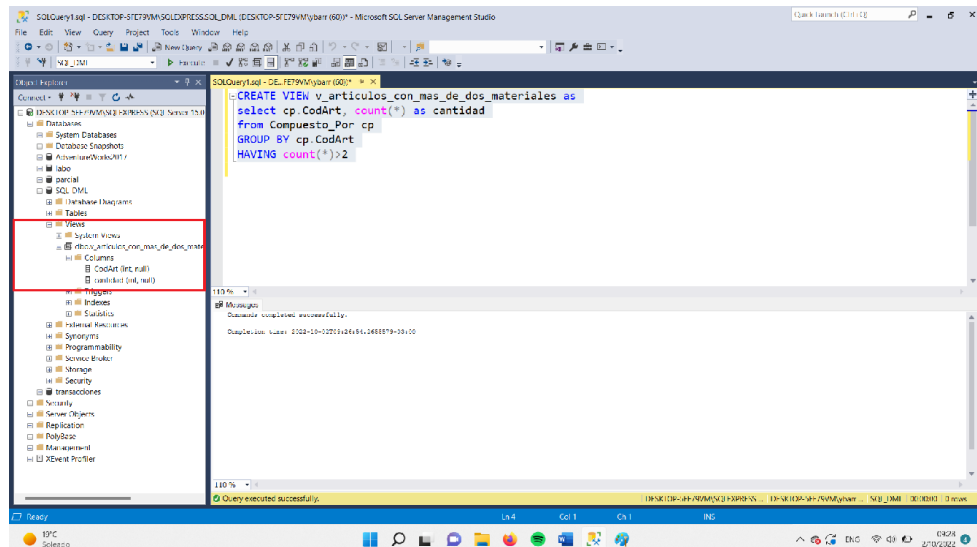
select cp.CodArt, count(*) as cantidad

from Compuesto_Por cp

GROUP BY cp.CodArt

HAVING count(*)>2
```

SQL Nos informa que la ejecución es correcta y no nos informa tuplas en el resultado. Ahora, veamos el explorador de objetos para verificar si se ha creado nuestra vista:

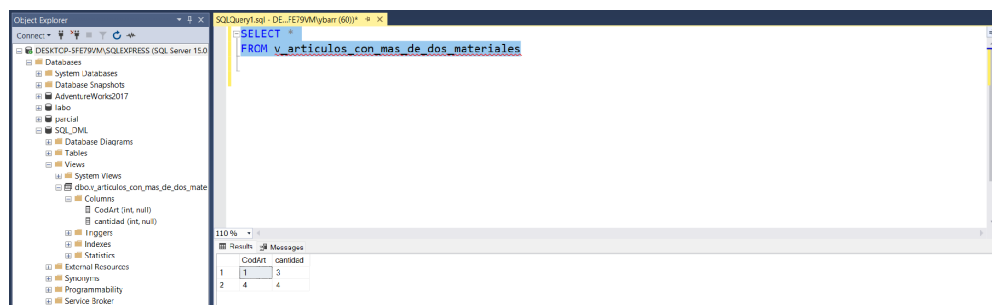


Aquí vemos como se ha creado nuestra vista y las columnas que la componen. Ahora, para utilizar la misma, simplemente la utilizamos en una consulta SQL desde el FROM de esta forma:

SELECT *

FROM v_articulos_con_mas_de_dos_materiales

Veamos el resultado de la ejecución:





Cierre

A lo largo de este módulo repasamos las sentencias básicas del subset DML (Data Manipulation Language) del lenguaje SQL. Resolvimos requerimientos utilizando operadores como: In, Not In, Exists, Not Exists. Group By, Having, Order By. Por otro lado, también utilizamos funciones agregadas: Count, Avg, Sum, Min, Max. También repasamos los conceptos de los distintos tipos de juntas como son Inner Join, Left Join y Outer Join. En este documento también vimos cómo crear y utilizar vistas en un motor de bases de datos relacionales.



Referencias

<https://www.sqlshack.com/es/introduccion-y-resumen-del-operador-logico-sql-like/>

<https://thedevelopmentstages.com/sql-exists-con-ejemplos/>

<https://www.ibm.com/docs/es/qmf/12.1.0?topic=queries-group-by>

<https://support.microsoft.com/es-es/office/cl%C3%A1usula-order-by-e8ea47f7-5388-460a-bec8-dcc81792d762>

<https://learn.microsoft.com/es-es/sql/relational-databases/views/views?view=sql-server-ver16>

tiiiiiit by 