

Capítulo 7

Consultas avanzadas con lenguaje SQL

Al finalizar el capítulo, el alumno podrá:

- Realizar consultas complejas aplicando los conceptos de agrupamiento, combinación y subconsulta.
- Construir vistas para simplificar el acceso a la información.

Temas:

1. Funciones de agrupamiento.
2. Consultas multitable.
3. Consultas anidadas.
4. Creando vistas

1. Funciones de agrupamiento

Funciones de agrupamiento GROUP BY

- Agrupar un conjunto de filas de acuerdo con los valores de una o más columnas o expresiones, devolviendo una fila por cada grupo.

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail AS sod
GROUP BY SalesOrderID
ORDER BY SalesOrderID;
```

```
SELECT DATEPART(yyyy,OrderDate) AS N'Year'
,SUM(TotalDue) AS N'Total Order Amount'
FROM Sales.SalesOrderHeader
GROUP BY DATEPART(yyyy,OrderDate)
ORDER BY DATEPART(yyyy,OrderDate);
```

Partition by CustID

CustID	OrderID	TotalDue
1	101	\$100
2	102	\$150
1	103	\$90
3	104	\$80
2	105	\$200
1	106	\$150

CustID	OrderID	TotalDue
1	101	\$100
1	103	\$90
1	106	\$150

CustID	OrderID	TotalDue
2	102	\$150
2	105	\$200

CustID	OrderID	TotalDue
3	104	\$80

B - 4
Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

1.1 Cláusulas GROUP BY y HAVING

La cláusula GROUP BY 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 en SQL Server 2016. Se devuelve una fila por cada grupo.

En algunos escenarios se establecen consultas complejas, que demandan de conceptos avanzados como consultas agrupadas, consultas multitablas o subconsultas.

1.1.1 La cláusula GROUP BY

La cláusula GROUP BY combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario, si se incluye una función SQL agregada, como por ejemplo **SUM** o **COUNT**, en la instrucción **SELECT**. Su sintaxis es:

```
SELECT  CAMPOS
FROM    TABLA
WHERE   CRITERIO
GROUP  BY CAMPOS DEL GRUPO
```

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Además, los valores NULL en los campos GROUP BY, se agrupan y no se omiten. No obstante, los valores NULL no se evalúan en ninguna de las funciones SQL agregadas.

Asimismo, se utiliza la cláusula WHERE para excluir aquellas filas que no se desea agrupar y la cláusula HAVING, para filtrar los registros, una vez agrupados.

A menos que contenga un dato Memo, un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no está incluido en la instrucción SELECT, siempre y cuando, esta instrucción incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de SELECT deben incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

```
SELECT CategoryID, SUM(UnitsInStock)
FROM Products
GROUP BY CategoryID
```

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

1.1.2 La cláusula HAVING

HAVING es similar a WHERE, pues determina qué registros se seleccionan. Una vez que los registros se han agrupado, HAVING determina cuáles de ellos se van a mostrar.

```
SELECT CategoryID, SUM(UnitsInStock)
FROM Products
GROUP BY CategoryID
HAVING SUM(UnitsInStock) > 500
```

Ejemplo 1:

Se tiene la siguiente tabla [Order Details]

OrderID	ProductID	Quantity	UnitPrice
10248	11	12	14.00
10248	42	10	9.80
10248	72	5	34.80
10249	14	9	42.40
10249	51	40	42.40

Se solicita mostrar el OrderID y el total por cada orden:

```
SELECT OrderID, SUM(UnitPrice * Quantity) as TotalAmount
FROM [Order Details]
GROUP BY OrderID
```

El resultado obtenido sería el siguiente.

OrderID	TotalAmount
10248	440.00
10249	1863.40

Ejemplo 2:

Haciendo uso del HAVING, se desea filtrar los datos agrupados obtenidos y mostrar solamente los que tengan un **TotalAmount** mayor de 15,000.00.

La solución sería la siguiente.

```
SELECT OrderID, SUM(UnitPrice * Quantity) as TotalAmount
FROM [Order Details]
GROUP BY OrderID
HAVING SUM(UnitPrice * Quantity) > 15000
```

El nuevo resultado obtenido sería el siguiente.

OrderID	TotalAmount
10865	17250.00
10981	15810.00
11030	16321.90

1.1.3 Las funciones de agregación

Sobre los diferentes grupos que se puedan formar, tras aplicar la cláusula GROUP BY, se pueden emplear funciones de agrupamiento. Estas funciones aplican conceptos aritméticos (sumatoria, promedio, máximo, mínimo o conteo) de los registros en cada grupo.

- **Función SUM**

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

```
SELECT SUM(EXPR) FROM TABLA
```

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

Por ejemplo:

```
SELECT SUM(UnitPrice * Quantity)
FROM [Order Details]
```

- **Función AVG**

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

```
SELECT AVG(EXPR) FROM TABLA
```

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

```
SELECT  AVG(UnitPrice)
FROM    Products
```

- **Funciones MIN y MAX**

Devuelven el mínimo o máximo de un conjunto de valores CONTENIDO en un campo específico de una consulta. Su sintaxis es.

```
SELECT  MIN(EXPR) FROM TABLA
SELECT  MAX(EXPR) FROM TABLA
```

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

Por ejemplo: se solicita el mínimo stock y luego, el máximo del stock de los productos.

```
SELECT  MIN(UnitsInStock)
FROM    Products

SELECT  MAX(UnitsInStock)
FROM    Products
```

- **Función COUNT**

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

```
SELECT  COUNT(EXPR) FROM TABLA
```

EXPR contiene el nombre del campo que desea contar. Sus operandos pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario, pero no otras de las funciones agregadas de SQL). Además, puede contar cualquier tipo de datos incluso texto.

Aunque **EXPR** puede realizar un cálculo sobre un campo, COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función COUNT no cuenta los registros que tienen campos NULL, a menos que EXPR sea el carácter comodín asterisco (*). Si se utiliza un asterisco, COUNT calcula el número total de registros, incluyendo aquellos que contienen campos NULL.

COUNT(*) es considerablemente más rápida que COUNT(Campo). No se debe poner el asterisco entre dobles comillas (*').

```
SELECT COUNT(*) as Total FROM Products
```

Asimismo, si EXPR identifica a múltiples campos, la función COUNT cuenta un registro, solo si al menos uno de los campos no es NULL. Si todos los campos especificados son NULL, no se cuenta el registro. Hay que separar los nombres de los campos con '+'. Hay que separar los nombres de los campos con '+'.

```
SELECT COUNT(ProductName + ' ' + QuantityPerUnit) FROM Products
```

1.1.4 Operadores CUBE o ROLLUP

Los operadores CUBE y ROLLUP son extensiones de la cláusula GROUP BY, permitiendo calcular subtotales de acuerdo con las agrupaciones especificadas en la consulta.

- **CUBE**

El operador CUBE genera un conjunto de resultados que es un cubo multidimensional. Este operador genera filas de agregado mediante la cláusula GROUP BY simple.

CUBE genera una agrupación para todas las permutaciones de expresiones de la <lista de elementos compuestos>. El número de agrupaciones generado es igual a 2^n , donde "n" es el número de expresiones de la <lista de elementos compuestos>.

Por ejemplo, vamos a listar la cantidad de órdenes por año y cliente cuyo nombre empiece con la letra 'D'.

```
SELECT YEAR(OrderDate) as n_year,
       CustomerID,
       COUNT(OrderID) As n_count
FROM   orders
WHERE  CustomerID LIKE 'D%'
GROUP BY CUBE(YEAR(OrderDate), CustomerID)
```

El resultado obtenido sería el siguiente:

n_year	CustomerID	n_count
1996	DRACD	2
1997	DRACD	1
1998	DRACD	3
NULL	DRACD	6
1996	DUMON	1
1997	DUMON	2
1998	DUMON	1
NULL	DUMON	4
NULL	NULL	10
1996	NULL	3
1997	NULL	3
1998	NULL	4

← Totales por CustomerID

← Totales General

← Totales por Año

• ROLLUP

El operador ROLLUP es útil para generar reportes que contienen subtotales y totales. Genera un conjunto de resultados que es similar al conjunto de resultados del CUBE.

Al usar el operador ROLLUP, se pueden crear agrupamientos en el conjunto de resultados. Para las filas agrupadas, se usa un valor NULL para representar todos los valores de la columna (excepto la columna SUM).

Por ejemplo, se necesita listar los empleados y la cantidad de órdenes por año, de los empleados cuyo apellido (LastName) empiece por 'A' o 'B' o 'C'; mostrando los totales por empleado.

```
SELECT E.LastName + ' ' + E.FirstName as Empleado,
       YEAR(O.OrderDate) As Año,
       COUNT(O.Orderid) As 'Cantidad de Órdenes'
FROM   Orders As O
INNER JOIN Employees As E ON O.EmployeeID = E.EmployeeID
WHERE  E.LastName LIKE '[ABC]%'
GROUP BY E.LastName+ ' ' + E.FirstName, YEAR(O.OrderDate)
WITH ROLLUP
```

El resultado obtenido sería el siguiente:

Empleado	Año	Cantidad Ordenes
Buchanan Steven	1996	11
Buchanan Steven	1997	18
Buchanan Steven	1998	13
Buchanan Steven	NULL	42
Callahan Laura	1996	19
Callahan Laura	1997	54
Callahan Laura	1998	31
Callahan Laura	NULL	104
NULL	NULL	146

Note los totales incluidos por cada empleado

1.1.4 Operación UNIÓN

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 UNION. La operación **UNIÓN** es distinta de la utilización de combinaciones de columnas de dos tablas.

Las reglas básicas para combinar los conjuntos de resultados de dos consultas con **UNIÓN** son:

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

Por ejemplo: Se solicita el unir la ciudad y el país de las tablas clientes (**Customers**) y proveedores (**Suppliers**).


```
SELECT City, Country FROM customers
UNION
SELECT City, Country FROM suppliers
ORDER BY City
```


2. Consultas multitable

**Consultas multitable
INNER JOIN**

- Devuelven todos los pares de las filas coincidentes.
- Rechaza las filas no coincidentes de las dos tablas.

```
-- By default, SQL Server performs an INNER JOIN if only the JOIN
-- keyword is specified.
SELECT p.Name, sod.SalesOrderID
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY p.Name ;
```

6 - 11Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

2.1 Consultas multitable

Hasta el momento las consultas realizadas solo han involucrado una tabla, pero gracias al concepto de llave foránea, se pueden relacionar 2 o más tablas, en base a los campos en común que estas poseen y realizar combinaciones entre ellas.

2.1.1 El operador INNER JOIN

Una combinación interna es aquella en la que los valores de las columnas que se están combinando se comparan mediante un operador de comparación.

En el estándar SQL-92, las combinaciones internas se pueden especificar en las cláusulas FROM o WHERE. Este es el único tipo de combinación que SQL-92 admite en la cláusula WHERE.

Las combinaciones internas especificadas en la cláusula WHERE se conocen como combinaciones internas al estilo antiguo.

Esta consulta de Transact-SQL es un ejemplo de una combinación interna.

```
SELECT *
FROM Products P
INNER JOIN Suppliers S ON P.SupplierID = S.SupplierID
WHERE P.CategoryID = 1
```

Esta combinación interna se conoce como una combinación equivalente, pues devuelve todas las columnas de ambas tablas y solo devuelve las filas en las que haya un valor igual en la columna de la combinación.

Es equivalente a la siguiente consulta.

```
SELECT *
FROM Products P,
Suppliers S
WHERE P.SupplierID = S.SupplierID
AND P.CategoryID = 1
```

Comprobando los resultados, se observa que la columna SupplierID aparece dos veces. Puesto que no tiene sentido repetir la misma información, se puede eliminar una de estas dos columnas idénticas, si se cambia la lista de selección.

El resultado se llama combinación natural. La consulta anterior de Transact-SQL se puede volver a formular para que forme una combinación natural.

Por ejemplo:

```
SELECT P.ProductID, P.SupplierID, P.ProductName, P.QuantityPerUnit,
P.UnitPrice, P.UnitsInStock, S.CompanyName, S.ContactName,
S.ContactTitle
FROM Products P,
Suppliers S
WHERE P.SupplierID = S.SupplierID
AND P.CategoryID = 1
```

2.1.2 El operador LEFT OUTER JOIN

Las combinaciones internas solo devuelven filas cuando hay una fila de ambas tablas, como mínimo, que coincide con la condición de la combinación. Estas combinaciones eliminan las filas que no coinciden con alguna fila de la otra tabla. Sin embargo, las combinaciones externas devuelven todas las filas de una de las tablas o vistas mencionadas en la cláusula FROM, como mínimo, siempre que tales filas cumplan con alguna de las condiciones de búsqueda de WHERE o HAVING.

Además, todas las filas se recuperarán de la tabla izquierda a la que se haya hecho referencia con una combinación externa izquierda (LEFT OUTER JOIN) y de la tabla derecha a la que se haya hecho referencia con una combinación externa derecha (RIGHT OUTER JOIN).

Para mostrar mejor el trabajo del operador LEFT OUTER JOIN se ha creado el escenario siguiente: usted tiene un estilo de vida saludable, tiene una lista de 10 verduras que le gustaría comer y quiere seguir que verduras come en cada día de la semana; por lo que necesita saber que verduras comió y no comió en la semana, y que días de la semana no comió ningún vegetal.

Vamos a crear 3 tablas basado en el escenario descrito anteriormente, días de la semana, vegetales y comer.

```
CREATE TABLE wkday
( id tinyint NOT NULL CONSTRAINT pk_wkday PRIMARY KEY (id),
name varchar(20) NOT NULL,
)

insert into wkday (id, name)
values (1, 'Monday'), (2, 'Tuesday'), (3, 'Wednesday'),
(4, 'Thursday'), (5, 'Friday'), (6, 'Saturday'), (7, 'Sunday')

CREATE TABLE veggie
```

```
( id      tinyint NOT NULL CONSTRAINT pk_veggie PRIMARY KEY (id),
  name    varchar(30) NOT NULL
)

insert into veggie (id, name)
values (1, 'Potato'), (2, 'Carrot'), (3, 'Broccoli'), (4, 'Cauliflower'),
       (5, 'Celery'), (6, 'Spinach'), (7, 'Tomato'), (8, 'Pumpkins'),
       (9, 'Eggplant'), (10, 'Cucumbers')

CREATE TABLE eating
( id          int NOT NULL CONSTRAINT PK_eating PRIMARY KEY (id),
  wkday_id    tinyint NOT NULL
    CONSTRAINT FK_eating_wkday FOREIGN KEY (wkday_id)
    REFERENCES wkday (id),
  veggie_id   tinyint NOT NULL
    CONSTRAINT FK_eating_veggie FOREIGN KEY (veggie_id)
    REFERENCES veggie (id)
)

insert into eating (id, wkday_id, veggie_id)
values (1,1,3), (2,1,2), (3,2,4), (4,4,6), (5,4,9),
       (6,4,3), (7,5,1), (8,6,10), (9,6,2)
```

Suponiendo el caso que de la base de datos que hemos creado, se requiere listar todas las filas de la tabla **wkday**, tanto si hay una coincidencia en la columna **ID** de la tabla **eating** como si no la hay.

La solución sería la siguiente.

```
SELECT wd.name as wkday_name,
       wd.id as id_in_wkday_table,
       e.wkday_id as wkday_id_in_eating_table,
       e.veggie_id as veggie_id
FROM   wkday wd
LEFT   OUTER JOIN eating e ON e.wkday_id = wd.id
```

2.1.3 El operador RIGHT OUTER JOIN

Este operador es similar al anterior. En realidad, todo depende donde se coloquen las tablas a combinar. Por ejemplo, para incluir a todos los vendedores en los resultados, independientemente de si están asignados a un territorio de ventas, utilice la combinación externa **RIGHT OUTER JOIN**.

La solución sería la que se muestra.

```
SELECT wd.name as wkday_name,
       wd.id as id_in_wkday_table,
       e.wkday_id as wkday_id_in_eating_table,
       e.veggie_id as veggie_id
FROM   eating e
RIGHT   OUTER JOIN wkday wd ON e.wkday_id = wd.id
```

2.1.4 El operador FULL OUTER JOIN

Una combinación externa completa devuelve todas las filas de las tablas de la izquierda y la derecha. Cada vez que un final no tenga coincidencia en la otra tabla, las columnas de la lista de selección de la otra tabla contendrán valores NULL. Cuando hay una coincidencia entre las tablas, la fila completa del conjunto de resultados contendrá los valores de datos de las tablas base.

Para retener información que no coincida al incluir las filas no coincidentes en los resultados de una combinación, utilice una combinación externa completa. SQL Server proporciona el operador de combinación externa completa FULL JOIN, que incluye todas las filas de ambas tablas, con independencia de que la otra tabla tenga o no un valor coincidente. En forma práctica, podemos decir que FULL JOIN es una combinación de LEFT OUTER JOIN y RIGHT OUTER JOIN

Por ejemplo, se necesita un listado de los proveedores que no tienen clientes en su país, y clientes que no tienen proveedores en su país y clientes y proveedores que son del mismo país.

```
SELECT  C.CompanyName as CompanyNameCustomers,
        C.Country as CountryCustomers,
        S.CompanyName as CompanyNameSuppliers,
        S.Country as CountrySuppliers
FROM    Customers C
FULL    OUTER JOIN Suppliers S ON C.Country = S.Country
ORDER  BY C.Country, S.Country
```

El resultado obtenido sería el siguiente:

CompanyName Customers	Country Customers	CompanyName Suppliers	Country Suppliers
NULL	NULL	Pavlova, Ltd.	Australia
NULL	NULL	G'day, Mate	Australia
NULL	NULL	Tokyo Traders	Japan
NULL	NULL	Mayumi's	Japan
NULL	NULL	Zaanse Snoepfabriek	Netherlands
NULL	NULL	Leka Trading	Singapore
Cactus Comidas para llevar	Argentina	NULL	NULL
Océano Atlántico Ltda.	Argentina	NULL	NULL
Rancho grande	Argentina	NULL	NULL
Piccolo und mehr	Austria	NULL	NULL
Ernst Handel	Austria	NULL	NULL
Suprêmes délices	Belgium	NULL	NULL
Maison Dewey	Belgium	NULL	NULL


3. Consultas anidadas

Consulta anidada

- Devuelve un valor único y está anidada en una instrucción:
 - SELECT
 - INSERT
 - UPDATE
 - DELETE
- La sub-consulta incluye siempre paréntesis.
- No puede contener un COMPUTE.
- Puede incluir una cláusula ORDER BY.

6 - 14

Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.



3.1 Definición

Una consulta anidada o subconsulta es una consulta SELECT que devuelve un valor único y se encuentra anidada en una instrucción SELECT, INSERT, UPDATE o DELETE, o dentro de otra subconsulta.

Una subconsulta se puede utilizar en cualquier parte en la que se permita una expresión. Por ejemplo:

```
SELECT CustomerID, CompanyName
FROM Customers
WHERE CustomerID IN (SELECT CustomerID
                     FROM Orders
                     WHERE OrderDate > '1998-05-01')
```

Se llama también subconsulta a una consulta o selección interna, mientras que la instrucción que contiene una subconsulta también es conocida como consulta o selección externa.

Muchas de las instrucciones Transact-SQL que incluyen subconsultas se pueden formular también como combinaciones. Otras preguntas se pueden formular solo con subconsultas.

En Transact-SQL, normalmente no hay diferencias de rendimiento entre una instrucción que incluya una subconsulta y una versión equivalente que no lo haga. Sin embargo, en algunos casos en los que se debe comprobar la existencia de algo, una combinación produce mejores resultados. De lo contrario, la consulta anidada debe ser procesada para cada resultado de la consulta externa con el fin

de asegurar la eliminación de los duplicados. En tales casos, la utilización de combinaciones produciría los mejores resultados.

Ejemplo de una subconsulta SELECT y una combinación SELECT que devuelve el mismo conjunto de resultados:

```
/* FORMA 1 */
SELECT CustomerID, CompanyName
  FROM Customers
 WHERE CustomerID IN (SELECT CustomerID
                      FROM Orders
                      WHERE OrderDate > '1998-05-01')

/* FORMA 2 */
SELECT C.CustomerID, C.CompanyName
  FROM Customers C
 INNER JOIN Orders O ON C.CustomerID = O.CustomerID
 WHERE OrderDate > '1998-05-01'
```

Una subconsulta anidada en la instrucción externa SELECT tiene los siguientes componentes:

- Una consulta SELECT normal, que incluye los componentes normales de la lista de selección.
- Una cláusula normal FROM, que incluye uno o más nombres de tablas o vistas.
- Una cláusula opcional WHERE.
- Una cláusula opcional GROUP BY.
- Una cláusula opcional HAVING.

La consulta SELECT de una subconsulta se incluye siempre entre paréntesis. No puede incluir una cláusula COMPUTE y solo puede incluir una cláusula ORDER BY cuando se especifica también una cláusula TOP.

Una subconsulta puede anidarse dentro de la cláusula WHERE o HAVING de una instrucción externa SELECT, INSERT, UPDATE o DELETE, o dentro de otra subconsulta.

Se puede disponer de hasta 32 niveles de anidamiento, aunque el límite varía, dependiendo de la memoria disponible y de la complejidad del resto de las expresiones de la consulta. Una subconsulta puede aparecer en cualquier parte en la que se pueda usar una expresión, si devuelve un valor individual.

Si una tabla aparece solo en una subconsulta, las columnas de esa tabla no se pueden incluir en el resultado (la lista de selección de la consulta externa).

Las instrucciones que incluyen una subconsulta tienen uno de estos formatos:

- WHERE expresión [NOT] IN (subconsulta)
- WHERE expresión_de_comparacion [ANY | ALL] (subconsulta)
- WHERE [NOT] EXISTS (subconsulta)

En algunas instrucciones, la subconsulta se puede evaluar como si fuera una consulta independiente.

Conceptualmente, los resultados de la subconsulta se sustituyen en la consulta externa.

Hay dos tipos básicos de subconsultas:

- Las que operan en listas incorporadas con IN.

- Las que son pruebas de existencia que se introducen con EXISTS.

- **Predicado IN**

Se emplea para recuperar aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual.

Por ejemplo, la siguiente consulta busca los nombres de todos los productos que sus proveedores sean de 'UK'.

```
SELECT ProductName, QuantityPerUnit
FROM Products
WHERE SupplierID IN (SELECT SupplierID
                     FROM Suppliers
                     WHERE Country = 'UK')
```

- **Predicado EXISTS**

Es utilizado en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

Se puede utilizar, también, alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM, fuera de la subconsulta.

Por ejemplo, la siguiente consulta busca los nombres de todos los clientes que existan en las órdenes y que el país de envío sea España.

```
SELECT CompanyName
FROM Customers C
WHERE EXISTS (SELECT *
              FROM Orders O
              WHERE C.CustomerID = O.CustomerID
              AND O.ShipCountry = 'Spain')
```

3.2 Operador relacional PIVOT

Permite rotar datos de filas a columnas. La sentencia PIVOT tiene la siguiente sintaxis.

```
SELECT <columna no dinamizada>,
       [primera columna dinamizada] AS <nombre de columna>,
       [segunda columna dinamizada] AS <nombre de columna>
       ...
       [última columna dinamizada] AS <nombre de columna>
FROM
  (<la consulta SELECT que genera los datos>)
  AS <alias de la consulta de origen>
PIVOT
(
  <función de agregación>(<columna que se agrega>)
FOR
  [<columna que contiene los valores que se convertirán en encabezados de columna>]
  IN ([primera columna dinamizada], [segunda columna dinamizada]
     ... [última columna dinamizada])
) AS <alias de la tabla dinamizada>
<cláusula ORDER BY opcional>;
```

El siguiente ejemplo se consulta la tabla Orders para determinar el número de pedidos de compra colocados por ciertos empleados.

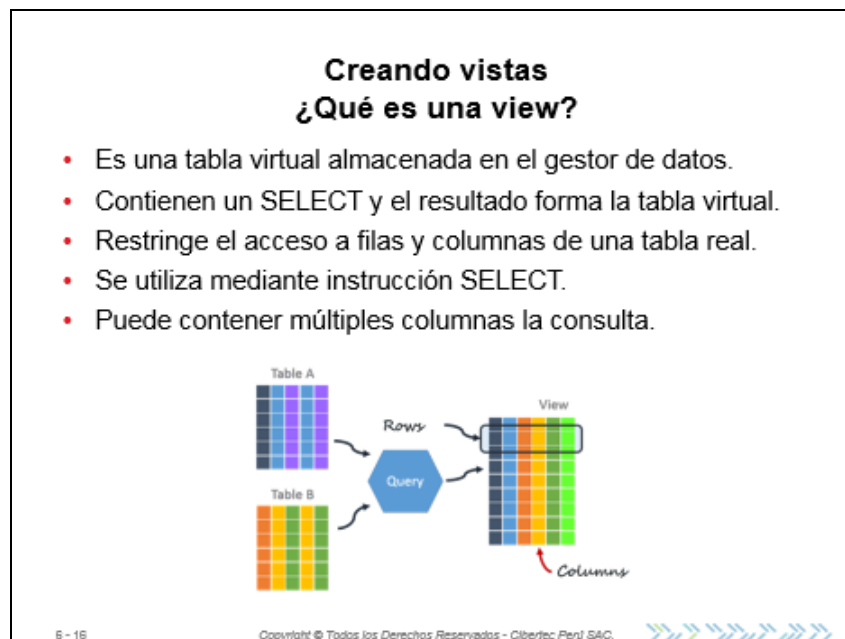
```
SELECT EmployeeID, [1996] as n_1996, [1997] as n_1997, [1998] as n_1998
FROM (SELECT EmployeeID, YEAR(OrderDate) as n_year, OrderID
      FROM Orders) O
PIVOT (COUNT (OrderID) FOR n_year IN ([1996], [1997], [1998])) AS pvt
ORDER BY pvt.EmployeeID
```

El resultado se muestra como sigue:

EmployeeID	n_1996	n_1997	n_1998
Andrew Fuller	16	41	39
Anne Dodsworth	5	19	19
Janet Leverling	18	71	38
Laura Callahan	19	54	31
Margaret Peacock	31	81	44
Michael Suyama	15	33	19
Nancy Davolio	26	55	42
Robert King	11	36	25
Steven Buchanan	11	18	13

Existe otra sentencia similar al PIVOT, llamada UNPIVOT que realiza la operación inversa a esta sentencia.

4. Creando vistas



4.1 Definición

Una vista se puede considerar una tabla virtual o una consulta almacenada. Los datos accesibles a través de una vista no están almacenados en un objeto distinto de la base de datos, pero sí una instrucción SELECT.

El resultado de la instrucción SELECT forma la tabla virtual que la vista devuelve. El usuario puede utilizar dicha tabla virtual haciendo referencia al nombre de la vista en instrucciones Transact-SQL, de la misma forma en que se hace referencia a las tablas.

a. Restringir el acceso del usuario a filas concretas de una tabla.

Por ejemplo, permitir que un empleado solo vea las filas que guardan su trabajo en una tabla de seguimiento de actividad laboral.

b. Restringir el acceso del usuario a columnas específicas.

Por ejemplo, permitir que los empleados que no trabajen en el departamento de nóminas vean las columnas de nombre, oficina, teléfono y departamento de la tabla de empleados, pero no permitir que vean las columnas con los datos de salario u otra información personal.

c. Combinar columnas de varias tablas de forma que parezcan una sola tabla.

Por ejemplo, una consulta muy extensa, que agrupa a varias tablas, se puede manejar como una sola tabla de manera práctica.

d. Agregar información en lugar de presentar los detalles.

Por ejemplo, presentar la suma de una columna o el valor máximo o mínimo de una columna.

4.2 Implementando vistas

Las vistas se crean definiendo la instrucción `SELECT` que recupera los datos presentados por la vista. Las tablas de datos a las que hace referencia la instrucción `SELECT` se conocen como las tablas base para la vista.

En este ejemplo, se crea una vista mediante una instrucción `SELECT` sencilla. Una vista sencilla resulta útil cuando se consulta con frecuencia una combinación de consultas. Los datos de esta vista provienen de las tablas `Employees` de la base de datos `NorthWind`. Los datos proporcionan el nombre e información sobre la fecha de contratación de los empleados. Esta vista puede crearse para la persona responsable del seguimiento de los aniversarios de trabajo, pero sin concederle acceso a todos los datos de la tabla.

```
CREATE OR ALTER VIEW vw_HireDate
AS
    SELECT TitleOfCourtesy, LastName, FirstName, Title, HireDate
    FROM Employees
```

Una vez creada, se puede hacer referencia a **hiredate_view** en las instrucciones, de la misma forma, que se hace referencia a una tabla.

```
/* PRUEBA*/
SELECT * FROM vw_HireDate
```

Se podría modificar una vista con el comando **ALTER VIEW [NombreVista]** o eliminar la vista con el comando **DROP VIEW [Nombre Vista]**.

Ventajas de las vistas:

- **Seguridad:** cada usuario puede acceder a la base de datos, a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver, restringiendo el acceso a datos almacenados.
- **Simplicidad de consulta:** una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que las consultas multitablas se formulen como consultas de una sola tabla con respecto a la vista.
- **Simplicidad estructurada:** las vistas pueden dar una visión personalizada de la estructura de la base de datos, presentando un conjunto de tablas virtuales que tienen sentido para ese usuario.
- **Aislamiento frente al cambio:** una vez vista puede presentar una imagen consistente inalterada de la estructura de la base datos, incluso si las tablas fuente subyacentes se dividen.
- **Integridad de datos:** si los datos se acceden y se introducen a través de una vista, el DBMS puede comprobar automáticamente los datos para asegurarse que satisfacen restricciones de integridad específicas.

Desventajas de las vistas:

- **Rendimiento:** crean la apariencia de una tabla, pero el DBMS debe traducir las consultas con respecto a la vista de consultas y a las tablas fuente, subyacentes. Si la vista se define mediante una consulta multitabla compleja, entonces una consulta sencilla con respecto a la vista, se convierte en una composición complicada y puede tardar mucho tiempo en completarse.
- **Restricciones de actualización:** cuando un usuario trata de actualizar filas de una vista, el DBMS debe traducir la petición de una actualización sobre las filas de las tablas fuente, subyacentes. Esto es posible para vistas sencillas, pero para vistas más complejas no pueden ser actualizadas; son solo de lectura.

4.3 Parámetros básicos

Dependiendo de las circunstancias, es posible encriptar una vista, es decir, que una vez creada, no sea posible ver la sentencia SELECT que la generó. Esto se suele hacer por medida de seguridad y para evitar que se altere el contenido de alguna vista importante. Para lograr tal objetivo, se debe incluir el parámetro WITH ENCRYPTION.

En el siguiente ejemplo, se selecciona una lista completa de las órdenes por cliente.

```
CREATE OR ALTER vw_Order_Customers WITH ENCRYPTION
AS
SELECT O.OrderID, C.CompanyName,
       SUM(OD.Quantity * OD.UnitPrice) as Total
FROM Orders O
INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
INNER JOIN Customers C ON C.CustomerID = O.CustomerID
GROUP BY O.OrderID, C.CompanyName
```

Ahora, se sabe que es poco probable actualizar datos de las tablas subyacentes desde una vista, pero en algunos casos, se puede dar tal situación. Si es una vista basada en una consulta condicionada, es posible hacer que la vista valide las actualizaciones en la vista, de tal forma que cumplan con el filtro de la misma.

Para tal efecto, se emplea el parámetro WITH CHECK OPTION, como se indica en el siguiente ejemplo, donde se muestra una vista denominada LondonOnly que hace referencia a la tabla Employees y permite modificar datos aplicados únicamente a los empleados que viven en **London**.

```
CREATE OR ALTER VIEW vw_LondonOnly
AS
SELECT LastName, FirstName, Title, TitleOfCourtesy,
       Address, City, Region, PostalCode, Country
FROM Employees
WHERE City = 'London'
WITH CHECK OPTION
```