Subconsultas

La consulta interna se evaluará primero, y la consulta externa recibe los valores de la consulta interna.

NOTA
Una consulta interna también puede estar anidada en una sentencia INSERT, UPDATE o DELETE, que se tratará más adelante en este libro.

Hay dos tipos de subconsultas:

- Autocontenidas: la consulta interna se evalúa lógicamente una sola vez.

Una subconsulta autocontenida puede utilizarse con los siguientes operadores:

- **Operadores de comparación**

  - Operador **IN**
  - Operador **ANY** o **ALL**

- Correlación: su valor depende de una variable de la consulta externa. Por lo tanto, la consulta interna de una subconsulta correlacionada se evalúa lógicamente cada vez que el sistema recupera una nueva fila de la consulta externa.

Consulta autocontenida

=

La subconsulta autocontenida que se utiliza con el operador =. Se puede utilizar cualquier operador de comparación, siempre que la consulta interna devuelva exactamente una fila.

Obtenga los nombres y apellidos de los empleados que trabajan en el departamento Research:

```
USE sample
SELECT emp_fname, emp_lname
FROM employee
WHERE dept_no =
      (SELECT dept_no FROM department WHERE dept_name = 'Research')
```

Obtenga los números de los empleados y las fechas de ingreso de todos los empleados con fechas de ingreso iguales a la fecha más antigua.

```
USE sample
SELECT emp_no,enter_date
FROM works_on
WHERE enter_date =
      (SELECT MIN(enter_date) FROM works_on)
```

El siguiente ejemplo muestra cómo se puede manejar el caso en el que el resultado de una consulta interna contiene un conjunto de valores.

Se desea traer aquellos nombres de productos (Production.Product) en donde el campo "Name" de la tabla (Production.ProductModel) sea "Long-sleeve logo jersey".

```
USE AdventureWorks2019
SELECT Name
FROM Production.Product
WHERE Product.ProductModelID =
      (SELECT ProductModelID FROM Production.ProductModel
           WHERE ProductModel.Name = 'Long-sleeve logo jersey')
```

IN

Obtener los detalles completos de todos los empleados cuyo departamento se encuentra en Dallas

```
USE sample
SELECT *
FROM employee
WHERE dept_no IN
      (SELECT dept_no FROM department WHERE location = 'Dallas')
```

Consulta con múltiples niveles de anidamiento

Obtener los apellidos de todos los empleados que trabajan en el proyecto Apollo

```
USE sample
SELECT emp_fname,emp_lname
FROM employee
WHERE emp_no IN
      (SELECT emp_no FROM works_on WHERE project_no IN
            (SELECT project_no FROM project WHERE project_name =
'Apollo'))
```

ANY y ALL

Los operadores ANY y ALL se utilizan siempre en combinación con uno de los operadores de comparación.

column_operator operador [ANY|ALL] consulta_interna

El operador ANY se evalúa como TRUE si el resultado de la consulta interna correspondiente contiene al menos una fila que satisfaga la comparación. La palabra clave SOME es el sinónimo de ANY.

Obtenga los números de los empleados, los números de los proyectos y los nombres de los puestos de trabajo de los empleados que no han pasado más tiempo en uno de los proyectos.

```
USE sample
SELECT DISTINCT emp_no,project_no,job
FROM works_on
WHERE enter_date > ANY
      (SELECT enter_date FROM works_on)
```

NOTA

¡No utilice los operadores ANY y ALL! Toda consulta que utilice ANY o ALL puede formularse mejor con la función EXISTS.

Consulta correlacionada

Obtener los apellidos de todos los empleados que trabajan en el proyecto p3

```
USE sample
SELECT emp_lname
FROM employee
WHERE 'p3' IN
      (SELECT project_no FROM works_on WHERE works_on.emp_no =
                                             employee.emp_no)
```

Obtenemos el mismo resultado con la siguiente consulta

```
USE sample
SELECT emp_lname
FROM employee
WHERE emp_no IN
      (SELECT emp_no FROM works_on WHERE project_no = 'p3')
```

EXISTS

La función EXISTS toma una consulta interna como argumento y devuelve TRUE si la consulta interna devuelve una o más filas, y devuelve FALSE si devuelve cero filas.

Obtener los apellidos de todos los empleados que trabajan en el proyecto p1

```
USE sample
SELECT emp_lname
FROM employee
WHERE EXISTS
      (SELECT project_no FROM works_on
       WHERE works_on.emp_no = employee.emp_no
       AND project_no = 'p1')
```

Obtener los apellidos de todos los empleados que trabajan para departamentos que no están ubicados en Seattle.

```
USE sample
SELECT emp_lname
FROM employee
WHERE NOT EXISTS
      (SELECT dept_no FROM department
       WHERE department.dept_no = employee.dept_no
       AND location = 'Seattle')
```

Liste aquellos clientes que tengan al menos una compra realizada u Orden realizada. Se utilizan las tablas Sales.Customer y Sales.SalesOrderHeader.

```
USE AdventureWorks2019
SELECT C.CustomerID
FROM Sales.Customer C
WHERE EXISTS (SELECT CustomerID
              FROM Sales.SalesOrderHeader SO
              WHERE SO.CustomerID = C.CustomerID)
```

Lista de clientes que no hayan tenido una compra.

```
USE AdventureWorks2019
SELECT C.CustomerID
FROM Sales.Customer C
WHERE NOT EXISTS (SELECT CustomerID
                  FROM Sales.SalesOrderHeader SO
                  WHERE SO.CustomerID = C.CustomerID)
```

Expresiones de tabla

Hay dos tipos de subconsultas:

Las expresiones de tabla son subconsultas que se utilizan cuando se espera una tabla. Hay dos tipos de expresiones de tabla:

- Tablas derivadas
- Expresiones de tabla comunes

Tabla derivada

Una tabla derivada es una expresión de tabla que aparece en la cláusula FROM de una consulta. Puede aplicar tablas derivadas cuando el uso de alias de columna no es posible porque el traductor SQL procesa otra cláusula antes de conocer el nombre del alias.

Obtener todos los grupos de meses existentes de la columna enter_date de la tabla works_on

Forma incorrecta

El motivo del mensaje de error es que la cláusula GROUP BY se procesa antes que la lista SELECT correspondiente, y el nombre del alias enter_month no se conoce en el momento en que se procesa la agrupación.

```
USE sample
SELECT MONTH(enter_date) AS 'enter_month' FROM works_on
GROUP BY enter_month
```

Forma correcta

Utilizando una tabla derivada que contenga la consulta anterior (sin la cláusula GROUP BY), se puede resolver este problema, ya que la cláusula FROM se ejecuta antes que la cláusula GROUP BY.

```
USE sample
SELECT enter_month
FROM (SELECT MONTH(enter_date) AS 'enter_month' FROM works_on) AS M
GROUP BY enter_month
```

En general, es posible escribir una expresión de tabla en cualquier lugar de una sentencia SELECT donde pueda aparecer una tabla. (El resultado de una expresión de tabla es siempre una tabla o, en un caso especial, una expresión). El ejemplo muestra el uso de una expresión de tabla en una lista SELECT.

```
USE sample
SELECT w.job, (SELECT e.emp_lname
               FROM employee e
               WHERE e.emp_no = w.emp_no) AS name
FROM works_on w
WHERE w.job IN ('Manager', 'Analyst')
```

Se desea obtener las unidades de cantidad vendida (OrderQty) por cada producto. Tablas: Production.Product y Sales.SalesOrderDetail.

```
USE AdventureWorks2019
SELECT P.Name,SUM(OD.OrderQty) AS Cantidad
FROM Production.Product P
JOIN Sales.SalesOrderDetail OD
ON OD.ProductID = P.ProductID
GROUP BY P.Name
```

Lo mismo que lo anterior pero utilizando TABLA DERIVADA

```
USE AdventureWorks2019
SELECT P.Name, P.ProductID, (SELECT SUM(OrderQty)
                             FROM Sales.SalesOrderDetail OD
                             WHERE OD.ProductID = P.ProductID)
FROM Production.Product P
```

Ejemplos utilizando UNION y Tabla derivada

```
USE SQLCookbook
```

Dos formas de obtener el mismo resultado

```
SELECT DEPTNO
FROM emp
UNION
SELECT DEPTNO
FROM dept

SELECT DISTINCT DEPTNO
FROM (SELECT DEPTNO
        FROM emp
       UNION ALL
       SELECT DEPTNO
         FROM dept) U
```

Tabla común (CTE)

Una expresión de tabla común (CTE) es una expresión de tabla con nombre soportada por Transact-SQL. Hay dos tipos de consultas que utilizan CTEs:

- Consultas no recursivas
- Consultas recursivas

NOTA

Las expresiones de tabla comunes también son utilizadas por el operador APPLY, que permite invocar una función con valor de tabla para cada fila devuelta por una expresión de tabla externa de una consulta

Consulta no recursiva

```
USE AdventureWorks2019

SELECT SalesOrderID
FROM Sales.SalesOrderHeader
WHERE TotalDue > (SELECT AVG(TotalDue)
                  FROM Sales.SalesOrderHeader
                  WHERE YEAR(OrderDate) = '2002')
   AND Freight > (SELECT AVG(TotalDue)
                  FROM Sales.SalesOrderHeader
                  WHERE YEAR(OrderDate) = '2002')/2.5
```

Utilizando WITH

La sintaxis de la cláusula WITH en las consultas no recursivas es

```
WITH cte_name (column_list) AS
(inner_query)
outer_query
```

```
USE AdventureWorks2019

WITH price_calc(year_2011) AS (SELECT AVG(TotalDue)
                               FROM Sales.SalesOrderHeader
                               WHERE YEAR(OrderDate) = '2011')

SELECT SalesOrderID
FROM Sales.SalesOrderHeader
WHERE TotalDue > (SELECT year_2011  FROM price_calc)
AND Freight > (SELECT year_2011  FROM price_calc)/2.5
```

Se crea una tabla virtual de nombre "price_calc" de una columna con nombre "year_2011".

La versión 9i de las bases de datos Oracle permite el uso de la claúsula WITH en SQL y PLSQL. Este comando permite reusar una consulta SELECT cuando esta hay que utilizarla más de una vez en una sentencia o consulta SQL compleja. Los resultados de la consulta definida en la claúsula WITH son almacenados en una tabla temporal pudiendo de esta forma mejorar el rendimiento de la sentencia principal.

Aunque no siempre conseguiremos mejorar el rendimiento utilizando la claúsula WITH, lo que sin duda facilitaremos es la lectura y el mantenimiento del código PL/SQL o SQL. Dentro de la claúsula WITH daremos un nombre a las consultas SELECT a reutilizar (WITH admite la definición de múltiples consultas con sólo separarlas por comas), dicho nombre será visible para todas las consultas definidas posteriormente dentro del mismo WITH. Obviamente, también será visible para la sentencia o consulta principal.

Uso de la claúsula WITH

En el siguiente ejemplo encontraremos todos las divisiones de una empresa cuyos empleados tienen un salario medio un 10 por ciento por encima del salario medio de la empresa:

```
WITH salario_division AS (SELECT division, AVG(salario) salario_medio
                          FROM   empleados
                          GROUP BY division)

SELECT division, salario_medio
FROM   salario_division
WHERE  salario_medio > (SELECT AVG(salario_medio) * 1.1
                        FROM salario_division)
ORDER BY salario_medio;
```

Como se puede observar en la consulta existen dos bloques, una consulta principal y una subconsulta. Ambas necesitan realizar operaciones agrupando datos. Reescribamos la sentencia sin utilizar la claúsula WITH:

```
SELECT division,
       AVG(salario) salario_medio
FROM   empleados
GROUP BY division
HAVING AVG(salario) > (
       SELECT AVG(salario) * 1.1
       FROM   empleados )
ORDER BY AVG(salario)
```

Comparando ambas sentencias podemos concluir que al utilizar la claúsula SQL WITH y almacenar en una tabla temporal la consulta que hace el GROUP BY, evitamos que se tengan que agrupar los datos más de una vez. Este hecho debe hacernos pensar que el rendimiento de la sentencia que utiliza la claúsula WITH debe ser mejor que el de la sentencia que no lo utiliza.

Características de la claúsula WITH

a) Sólo se puede usar en sentencias SELECT.

b) Cuando se define una consulta con el mismo nombre de una tabla existente en la base de datos Oracle, puesto que el analizador sintáctico o parser (ver fases en la ejecución de una sentencia SQL) de las sentencias SQL o PLSQL busca de dentro a fuera, el nombre dentro de la claúsula WITH tendrá prioridad frente al nombre de la tabla.

c) Puede contener más de una consulta. Cada consulta se separa mediante comas. Las consultas definidas después de otras consultas pueden utilizar las definiciones previas.

Subconsultas en joins

```
SELECT
      a.product_id ,
      a.product_name ,
      a.category,
      b.quantity
FROM product AS a
LEFT JOIN (SELECT product_id, SUM(quantity) AS quantity
           FROM sales
           GROUP BY product_id) AS b
ON a.product_id = b.product_id
ORDER BY b.quantity DESC
```
