UNIDAD DE TRABAJO 2: BASES DE DATOS RELACIONALES

TEMA 6: CONSULTAS (parte 3)

6.21. INTRODUCCIÓN

En este tercer tema sobre consultas veremos:

- Revisaremos las subconsultas, comenzando con una introducción a los tres operadores: **ANY, ALL y EXISTS**. Estos operadores te permiten crear una relación especial entre las consultas principales y las subconsultas, en lugar de utilizar el operador **IN** o los operadores estándar de comparación.
- También veremos las subconsultas correlacionadas, que son subconsultas que en alguna de las cláusulas hacen referencia a expresiones de columnas de la consulta principal.
- Veremos subconsultas en componentes de la consulta que no sean la cláusula WHERE.
- Veremos la cláusula WITH, que permite definir uno o más subconsultas en el inicio de los comandos SQL, y luego hacer referencia a ellos por su nombre en el resto de su comando SQL.
- Seguiremos con las consultas jerárquicas. Las tablas relacionales son estructuras esencialmente planas, pero pueden representar estructuras de datos jerárquicos, por ejemplo, mediante el uso de claves foráneas hacen referencia a la clave principal de la misma tabla. La columna de MGR de la tabla de los empleados es un ejemplo clásico de una relación jerárquica. Oracle SQL admite una sintaxis explícita para simplificar la recuperación de estructuras de datos jerárquicos.

6.22. SUSCONSULTAS… CONTINUACIÓN

Ya hemos visto varios ejemplos de subconsultas, usando el operador IN o estándar operadores de comparación lógicos.
Como repaso, vamos a empezar con dos ejemplos subconsulta estándar.

Mostrar los cursos para la categoría 'BLD':

```
select r.attendee, r.course, r.begindate
from registrations r
where r.course in (select c.code
                   from courses c
                   where c.category='BLD');
```

Mostrar información de aquellos empleados que sean mayores que el empleado número 7566.

```
select e.empno, e.ename, e.init, e.bdate
from employees e
where e.bdate > (select x.bdate
from employees x
where x.empno = 7566);
```

Sabemos que esta subconsulta debe devolver únicamente una fila, ya que si no, nos daría error.

Los operadores ANY y ALL

SQL nos permite combinar los operadores estándar de comparación (<,>, =, etc) con subconsultas que devuelven cualquier número de filas. Podemos hacerlo mediante la especificación de ANY o ALL entre el operador de comparación y la subconsulta.

Vamos a ver un ejemplo de ANY mostrando todos los empleados con un sueldo mensual que es más alto **que al menos un** 'manager':

```
select e.empno, e.ename, e.job, e.msal
from employees e
where e.msal > ANY (select x.msal
                    from employees x
                    where x.job = 'MANAGER');
```

Veamos ahora un ejemplo con el ALL: mostrar todos los empleados con un sueldo mensual que es más alto **que todos los** 'manager':

```
select e.empno, e.ename, e.job, e.msal
from employees e
where e.msal > ALL (select x.msal
                    from employees x
                    where x.job = 'MANAGER');
```

Definiendo ANY y ALL

- CUALQUIER ... significa que el resultado es cierto para al menos un valor devuelto por la subconsulta.
- TODO ... significa que el resultado es cierto para todos los valores devueltos por la subconsulta.

Reescribiendo subconsultas con ANY y ALL

En muchos casos podemos reescribir consultas de forma que no necesitemos usar ANY y ALL.

Ejemplo: reescribir la consulta que mostraba todos los empleados con un sueldo mensual que es más alto **que todos los** 'manager' sin usar el ALL

```
select e.ename, e.job, e.msal
        from employees e
        where e.msal > (select max(x.msal)
                        from employees x
                        where x.job = 'MANAGER');
```

Fijaros que las siguientes construcciones son lógicamente equivalentes:

X = ANY(suconsulta) <=> X IN (suconsulta)
X <> ALL(suconsulta) <=> X NOT IN (suconsulta)

Ahora observad estos dos casos:

X = ALL(subconsulta)
X <> ANY(subconsulta)

Si la subconsulta devuelve dos o más valores diferentes, la primera expresión es siempre FALSO, porque X nunca puede ser igual a dos valores diferentes al mismo tiempo. Asimismo, si la subconsulta devuelve dos o más valores diferentes, la segunda expresión es siempre cierto, ya que cualquier X será diferente de al menos uno de estos dos valores de la subconsulta.

Subconsultas correlacionadas

Fijaros en el siguiente ejemplo e intentemos averiguar por qué a estas subconsultas se les conoce como "correlacionadas".

```
select e.ename, e.init, e.msal
from employees e
where e.msal > (select avg(x.msal)
                from employees x
                where x.deptno = e.deptno);
```

Efectivamente, la subconsulta hace referencia a la tabla de la consulta principal (e).

Oracle gestiona la consulta de la siguiente manera:

- La variable de tupla e va analizando la tabla EMPLEADOS, asumiendo así 14 valores diferentes.
- Para cada fila e, la subconsulta se ejecuta después de sustituir e.DEPTNO por el valor literal de departamento e fila.

Ojo!!!: Volver a ejecutar una subconsulta para cada fila de la consulta principal puede tener un impacto en el rendimiento significativo. El optimizador de Oracle tratará de producir un plan de ejecución eficiente, y hay algunos algoritmos de optimización inteligentes para subconsultas correlacionadas. Con muchas consultas, el rendimiento con conjuntos de datos pequeños (como el que te encuentras en una base de datos de desarrollo) es bastante bueno, pero no son eficientes en bases de datos en producción grandes.

El operador EXISTS

Muchas veces se combinan las consultas correlacionadas con el operador EXISTS. Veamos un ejemplo:

Mostrar las ofertas de cursos en las que no se haya registrado nadie

```
select o.*
from offerings o
where not exists
        (select r.*
        from registrations r
        where r.course = o.course
        and r.begindate = o.begindate);
```

El operador EXISTS no está interesado en las filas reales de la consulta, si las hubiera. Comprueba únicamente la existencia de resultados de la subconsulta. Si la subconsulta devuelve al menos una fila como resultado, el operador EXISTS se evalúa como verdadero. Si la subconsulta no devuelve ningún registro, entonces el resultado es falso.

Subconsultas tras un operador EXISTS

Se puede decir que el EXISTS y NOT EXISTS son una especie de operadores de chequeo de conjunto vacío. Esto implica que no importa las expresiones que se especifican en la cláusula SELECT de la subconsulta.
El ejemplo anterior también se podría haber escrito de la siguiente manera:

```
select o.*
from offerings o
where not exists
        (select 'x'
        from registrations r ...
```

EXISTS, IN, o JOIN?

Fijaros en la siguiente consulta: tiene por objeto facilitar los datos personales de todos los empleados que alguna vez enseñaron un curso de SQL Server.

```
select e.*
from employees e
where exists (select o.*
                from offerings o
                where o.course = 'SQL'
                and o.trainer = e.empno);
```

```
EMPNO ENAME  INIT    JOB MGR  BDATE     MSAL   COMM DEPTNO

7369  SMITH    N TRAINER 7902  17-DEC-1965  800           20
7902  FORD    MG TRAINER 7566  13-FEB-1959  3000         20
```

Este problema también se puede resolver con un operador IN:

```
select e.*
from employees e
where e.empno in    (select o.trainer
                     from offerings o
                     where o.course = 'SQL')
```

También puede utilizar una join para resolver el problema:

```
select DISTINCT e.*
from employees e
join
offerings o
on e.empno = o.trainer
where o.course = 'SQL'
```

Qué ocurre si se quita la opción DISTINCT en el este ejemplo?. El resultado de la consulta constará de tres filas, en lugar de dos.
Hasta ahora, hemos considerado sólo subconsultas en la cláusula WHERE. Sin embargo, se pueden utilizar subconsultas en otros componentes de los estados de SQL, como SELECT y FROM cláusulas.

Los NULL con los operadores EXISTS e IN causan problemas, a veces pueden producir resultados incorrectos.

Tenemos que tener muy claros los siguientes conceptos:

- NULL no es un dato, sino algo desconocido
- NULL = NULL, NULL != NULL or NULL IN (NULL) siempre se evalúan como desconocido, es decir, ni verdadero ni falso.
- No es posible unir dos filas con NULL en la "join" columna

Vamos a ver un ejemplo de los problemas con los NULL. Se van a ver dos formas diferentes de mostrar los "managers", uno con IN y otro con EXISTS, a primera vista los dos enfoques funcionan y no parece haber ninguna diferencia entre ellos:

```
select ename
from employees
where empno in (select mgr from employees);
```

```
ENAME
--------
JONES
BLAKE
CLARK
SCOTT
KING
FORD
```

```
select e1.ename
from employees e1
where exists (select e2.mgr
              from employees e2
              where e1.empno = e2.mgr);
```

```
ENAME
--------
JONES
BLAKE
CLARK
SCOTT
KING
FORD
```

Como podemos ver el uso de IN o EXISTS es equivalente en términos de resultados, a pesar de las operaciones reales son diferentes.
EXISTS ejecuta la subconsulta para cada empno y devuelve TRUE si encuentra una coincidecia con empno. Sin embargo, las dos consultas devuelven los mismos

resultados sólo porque no están involucrados valores NULL. Si hay un valor NULL en empno, la subconsulta EXISTS no devolvería un registro para ese número de empleado, debido a que un valor NULL empno no se uniría con el valor NULL mgr (NULL = NULL no se evalúa como TRUE).

EXISTS responde a la pregunta, "¿es este el valor de la columna de la tabla especificada?" Como valores NULL no se puede equiparar, cuando se unen un mgr NULL a un empno NULL no devuelve TRUE.
Mientras que el IN responde a esta pregunta: "¿no existe en ninguna parte de la lista este valor?" Si un valor de la lista coincide con el valor externo, entonces la expresión se evalúa como TRUE.

Los NULLS con el NOT EXISTS y el NOT IN

Intuitivamente, el NOT EXISTS y el NOT IN deben devolver las filas de la tabla que no se devuelven por el EXISTS y el IN respectivamente. Esto es cierto para el NOT EXISTS, pero cuando se encuentran valores NULL, NOT IN no devolverá las filas que no se devuelvan por IN.
En el punto anterior, sacamos los empleados que eran "managers". Ahora queremos sacar los empleados que no son "managers".

```
select e1.ename
from employees e1
where not exists     (select e2.mgr
                      from employees e2
                      where e1.empno = e2.mgr);
```

```
ENAME
--------
SMITH
ALLEN
WARD
MARTIN
TURNER
ADAMS
JONES
MILLER
```

Hay 14 empleados, 6 que son "managers" y 8 que no lo son. Utilizando EXISTS y NOT EXISTS, todos los empleados están en la lista, independientemente de la presencia de un estado mgr NULL para una de las filas (empleado KING).

Ahora observemos el resultado utilizando el NOT IN.

```
select ename
from employees
where empno not in (select mgr from employees);
```

Ninguna fila seleccionada!!!. Aparentemente tenemos todos "managers" y ningún otro tipo de trabajador!!. ¿Por qué ocurre esto?, la razón radica en la pregunta, no en la respuesta.
Mientras los valores de la lista no coincidan con el valor externo, entonces la expresión se evalúa como TRUE. Por ejemplo, NOT IN 1234 (1234, NULL) es equivalente a 1234! = 1234 Y empno! = NULL. Cada comprobación de igualdad puede ser evaluada por separado y el resultado sería Verdadero y Desconocido. Verdadero y Desconocido es DESCONOCIDO.

6.23. SUBCONSULTAS EN LA CLÁUSULA SELECT

El estándar ANSI / ISO SQL estándar ofrece un la posibilidad de utilizar una subconsulta en la cláusula SELECT.

Seleccionar el número de empleados de cada departamento:

```
select d.deptno, d.dname, d.location,
        (select count(*)
        from employees e
        where e.deptno = d.deptno) as emp_count
from departments d;
```

No es sólo una solución correcta, también es una solución muy elegante. Es elegante, porque la tabla "padre" de esta consulta (véase la cláusula FROM) es la tabla DEPARTMENTS. Después de todo, estamos en busca de información acerca de los departamentos, por lo que la tabla DEPARTMENTS es la tabla más apropiada y obvia para comenzar nuestra búsqueda del resultado. Los tres primeros atributos (DEPTNO, DNAME y LOCATION) son los atributos "regulares" que se pueden encontrar en las columnas correspondientes de la tabla DEPARTAMENTOS, sin embargo, el cuarto atributo (el número de empleados) no se almacena como un valor de columna en la base de datos.

Se pueden resolver la mayoría (si no todos) de los problemas de agregación utilizando una subconsulta correlacionada en la cláusula SELECT, sin utilizar GROUP BY en absoluto.

6.24. SUBCONSULTAS EN LA CLÁUSULA FROM

La cláusula FROM es uno de los lugares más obvios para permitir subconsultas en SQL. Veamos un ejemplo de una subconsulta en la cláusula FROM.

```
select e.ename, e.init, e.msal
from employees e
join
        (select x.deptno
        , avg(x.msal) avg_sal
        from employees x
        group by x.deptno ) g
using (deptno)
where e.msal > g.avg_sal;
```

La gran diferencia entre la tabla "real" y la subconsulta es que la primera tiene un nombre. Por lo tanto, si utilizamos subconsultas en la cláusula FROM, se debe definir una variable de tupla (o alias de la tabla, en la terminología de Oracle) sobre el resultado de la subconsulta. Como vemos en la consulta anterior, hemos definido g como variable de tupla. Esta variable tupla nos permite referirnos a las expresiones de las columnas de la subconsulta, como lo demuestra g.AVG_SAL en la última línea del ejemplo. **Un requisito es que la subconsulta debe ser independiente de la consulta externa, no se pueden correlacionar.**

6.25. LA CLÁUSULA WITH

La anterior consulta nos mostraba un ejemplo del uso de una subconsulta en una cláusula FROM. Podríamos haber escrito la misma consulta con una sintaxis un poco diferente, como se muestra a continuación. Esta construcción se llama "factoring subquery".

```
WITH g AS
        (select x.deptno
        , avg(x.msal) avg_sal
        from employees x
        group by x.deptno)
select e.ename, e.init, e.msal
from employees e
join g
using (deptno)
where e.msal > g.avg_sal;
```

El uso de la sintaxis de la cláusula WITH se hace aún más atractiva si se hace referencia varias veces a la misma subconsulta de la consulta principal. Se pueden

definir tantas subconsultas como quieras en una sola cláusula WITH, separadas por comas.

```
WITH v1 AS (select ... from ...)
, v2 AS (select ... from ...)
, v3 AS ...
select ...
from ...
```

Hay varias ventajas de utilizar subconsultas "factorizadas". En primer lugar, pueden facilitar el desarrollo mediante el aislamiento de cada consulta y en segundo lugar, que el código sea más claro.

```
select ...
        from (select ...
                from (select ...
                        from (select ... from ...) v3
                        ) v2
                ) v1
```

Cuando hay un problema con la consulta, puede ser difícil localizar el problema real. Mediante el uso de subconsulta factoriales, se puede crear la subconsulta como una consulta independiente, seleccionar * en ella para comprobar la integridad, y poder ver si funciona. Veamos en un ejemplo los pasos que seguiríamos:

1.
```
select x.deptno
, avg(x.msal) avg_sal
from employees x
group by x.deptno;
```

2.
```
WITH g AS
            (select x.deptno
            , avg(x.msal) avg_sal
            from employees x
            group by x.deptno)
select *
from g;
```

3.
```
WITH g AS
            (select x.deptno
            , avg(x.msal) avg_sal
            from employees x
            group by x.deptno)
select e.ename, e.init, e.msal
from employees e
join g
using (deptno)
where e.msal > g.avg_sal;
```

Si se definen varias subconsultas en la cláusula WITH, se puede hacer referencia a cualquier nombre de subconsulta que hemos definido anteriormente en la misma cláusula, es decir, la definición de la subconsulta puede hacer referencia a V2 V1 en su cláusula FROM, y la definición de V3 puede referirse tanto a V1 y V2:

```
WITH v1 AS (select ... from ...)
        , v2 AS (select ... from V1)
        , v3 AS (select ... from V2 join V1)
select ...
from ...
```