UNIDAD DE TRABAJO 2: BASES DE DATOS RELACIONALES

TEMA 6: CONSULTAS (parte 2)

6.11. Introducción

En este tema vamos a tratar el concepto de fila o "variables de tupla". Al comenzar a utilizar múltiples tablas en la cláusula FROM de las sentencias SELECT, es una buena idea para empezar a utilizar las variables de tupla (también conocido como alias de tabla) de una manera coherente.

Veremos las "JOINS", el "GROUP BY", las funciones de grupo más usadas, el "HAVING" y terminaremos viendo tres operadores importantes en oracle: UNION, MINUS e INTERSECT.

6.12. Variables de tupla

Hasta ahora, nosotros formulábamos nuestras sentencias SQL de la siguiente forma:

```
select ename, init, job
from employees
where deptno = 20;
```

En realidad, esta afirmación es bastante incompleta. Ahora seremos un poco más precisos, debido a que los comandos SQL son cada vez un poco más complicados. Para ser completa y exacta deberíamos haber puesto algo así:

```
select e.ename, e.init, e.job
from employees e
where e.deptno = 20;
```

En este ejemplo, e es una variable tupla. "Tupla" es sólo un nombre para el término "fila" derivado de la teoría relacional. **En Oracle, las variables de tupla se conocen como alias de tabla.** Esa variable de tupla se declara en la cláusula FROM, inmediatamente después del nombre de la tabla, separada por espacios en blanco únicamente.

Una variable tupla siempre se extiende sobre una tabla o una expresión de tabla. En el ejemplo anterior **"e"** es una variable que representa una fila de la tabla empleados en cualquier momento. En el contexto de una fila específica, puede hacer referencia a los valores de una columna específica, como se muestra en la cláusula SELECT y WHERE del ejemplo. La variable tupla precede al nombre de la columna, separados por un punto.

6.13. JOINS

Se pueden especificar múltiples tablas en el FROM de una consulta. Vamos a probar qué ocurre con el siguiente ejemplo: nuestra intención es descubrir que empleados pertenecen a los departamentos:

```
select deptno, location, ename, init
from employees, departments;
```

¿Qué ocurre?, nos da un error:

```
ERROR at line 1:
ORA-00918: column ambiguously defined
```

197

El motor de Oracle no puede saber a qué columna "deptno" nos estamos refiriendo. Ambas tablas tienen una columna DEPTNO.

Producto Cartesiano

La siguiente select hace un segundo intento para encontrar los empleados que pertenecen a los departamentos. Si observamos el resultado, lo que obtenemos (al no introducir restricciones en la cláusula WHERE) es el producto cartesiano de ambas tablas: fijaros que nos salen 56 filas, 14 empleados y 4 departamentos.

```
select d.deptno, d.location, e.ename, e.init
from employees e, departments d;
```

Combinaciones de Igualdad (equijoins)

```
select d.deptno, d.location, e.ename, e.init
from employees e, departments d
where e.deptno = d.deptno
order by d.deptno, e.ename;
```

Esta es la select correcta, donde en el WHERE estamos especificando esa condición: una fila de una tabla y otra fila de la otra tabla siempre que la clave principal de una de las tablas coincida con la clave ajena en la otra tabla (relación).

Convenciones de Diseño

Las sentencias SQL deben ser correctas, por supuesto. Pero en el momento en que se hacen más largas y complicadas, se hace cada vez más importante adoptar un estilo de diseño determinado. Espacios en blanco adicionales (espacios, tabuladores y líneas nuevas) no tienen ningún significado en el lenguaje SQL, pero sin duda mejoran la legibilidad del código:

```
select d.deptno
, d.location
, e.ename
, e.init
from employees e
, departments d
where e.deptno = d.deptno
order by d.deptno
, e.ename;
```

Esta disposición ha demostrado ser muy útil en la práctica. Fíjate en la colocación de las comas al comienzo de la siguiente línea. Esto hace que agregar y quitar líneas sea más fácil. Cualquier otra norma está bien, también. Esto es sobre todo una cuestión de gusto. Sólo asegúrate de adoptar un estilo y utilizarlo de manera consistente.

Combinaciones de No Igualdad

Si utilizamos un operador de comparación que no sea un signo de igualdad en la cláusula WHERE, se llama una combinación de no igualdad. Para ver un ejemplo de una "no igualdad" veamos el siguiente ejemplo, que calcula el salario total anual para cada empleado.

```
select e.ename employee
, 12*e.msal+s.bonus total_salary
from employees e
, salgrades s
where e.msal between s.lowerlimit
and s.upperlimit;
```

Joins de tres o más tablas

Vamos a tratar de mejorar la consulta del listado anterior. En una tercera columna, también queremos ver el nombre del departamento en el que el empleado trabaja. Los nombres de los departamentos se almacenan en la tabla de departamentos; ¿cómo lo harías?

```
select e.ename employee
, 12*e.msal+s.bonus total_salary
, d.dname department
from employees e
, salgrades s
, departments d
where e.msal between s.lowerlimit
and s.upperlimit
and e.deptno = d.deptno;
```

El principio básico es simple. Ahora tenemos tres variables tupla (e, s, d). Por lo tanto, tenemos que tener (al menos) dos condiciones en la cláusula WHERE para obtener las combinaciones de filas correctas en el resultado de la consulta.

Self-Joins (Joins sobre una misma tabla)

El "autojoins" contendrá tablas a las que se hace referencia más de una vez en la cláusula FROM. Obligatoriamente utilizaremos las variables de tupla, ya que usando el nombre de la tabla puede producir ambigüedad.

Ejemplo: empleados nacidos después de enero de 1 de 1965 con el nombre de sus directivos.

```
select e.ename as employee
, m.ename as manager
from employees m
, employees e
where e.mgr = m.empno
and e.bdate > date '1965-01-01';
```

6.14. LA CLAÚSULA JOIN

Los ejemplos de combinación anteriores utilizan el operador producto cartesiano (la coma en la cláusula FROM) como punto de partida y luego filtran las filas en la cláusula WHERE. No hay absolutamente nada de malo en este enfoque, y la sintaxis es totalmente compatible con la norma ANSI / ISO SQL estándar, pero la norma ANSI / ISO SQL estándar también es compatible con la sintaxis alternativa para especificar las joins.

En primer lugar, vamos a ver de nuevo la instrucción anterior:

```
select e.ename as employee
, m.ename as manager
from employees m
, employees e
where e.mgr = m.empno
and e.bdate > date '1965-01-01';
```

Se puede entender que la cláusula WHERE de la consulta contiene dos tipos de condiciones diferentes: la línea 5 contiene la condición de combinación para asegurarse de que se combinan las filas de la derecha, y la línea 6 es un "verdadero" (no-unión) para filtrar los empleados con base en las fechas de nacimiento.
El siguiente listado muestra una consulta equivalente, produciendo los mismos resultados, utilizando una sintaxis de combinación diferente. Fijaros en las palabras clave JOIN y ON. También hay que tener en cuenta que esta sintaxis de combinación no utiliza comas en la cláusula FROM.

```
select e.ename as employee
, m.ename as manager
from employees m
JOIN
employees e
ON e.mgr = m.empno
where e.bdate > date '1965-01-01'
order by employee;
```

Esta sintaxis es más elegante que la sintaxis anterior, debido a que la unión está totalmente especificada en la cláusula FROM y la cláusula WHERE contiene sólo el filtrado.

Natural Joins

También puede utilizar el operador de unión natural en la cláusula FROM. El siguiente ejemplo muestra una unión entre EMPLEADOS y su HISTORIA.

```
select ename, beginyear, msal, deptno
from employees
natural join  history;
```

Pregunta: ¿cómo es posible que esta select produzca 15 filas en el resultado, en lugar de 14?

Para entender lo que está sucediendo, hay que saber lo que hace el operador de unión natural.
Este operador determina las columnas que las dos tablas (empleados e Historia) tienen en común. Y sin hacer un "=" en el where, son los resultados que va a sacar. En este caso, se trata de las tres columnas: EMPNO, MSAL Y DEPTNO.

1. Se unen las dos tablas (utilizando una combinación de igualdad) en todas las columnas que tienen en común.
2. Suprime las columnas duplicadas resultantes de la operación de unión en el paso anterior.
3. Por último, evalúa las cláusulas de consulta restantes. La única cláusula que queda es la cláusula SELECT. El resultado final muestra las cuatro columnas deseadas.

Al parecer, todos los empleados aparecen sólo una vez en el resultado, a excepción de Ward. Esto significa que este empleado ha sido contratado por el mismo departamento (30) por el mismo sueldo (1250) durante dos períodos diferentes de su carrera. Se trata de una pura coincidencia. Si la consulta hubiera devuelto 14 filas en lugar de 15, no nos habríamos percatado de que habría un error. A veces, consultas equivocadas pueden dar resultados "correctos" por accidente.

Este ejemplo muestra que se debe tener mucho cuidado cuando se utiliza el operador de unión natural. Probablemente una unión natural puede empezar a producir resultados extraños e indeseables si se agregan nuevas columnas a las tablas, o cambiar el nombre de las columnas existentes, haciendo que, accidentalmente, coincidan nombres de columna.

Ejercicio: probad el natural join con la tabla de empleados y departamentos. ¿Por qué saca 8 filas?. Pensadlo y escribidlo.

¡OJO!: Sólo usar el natural join cuando tengamos claro los nombres de los campos de las tablas a unir para no provocar resultados inesperados.

Combinaciones de igualdad en columnas con el mismo nombre

SQL ofrece una forma alternativa para especificar combinaciones de igualdad, lo que nos permite especificar explícitamente las columnas en la combinación de igualdad.
Como vimos en este ejemplo:

```
select e.ename as employee
, m.ename as manager
from employees m
JOIN
employees e
ON e.mgr = m.empno
where e.bdate > date '1965-01-01'
order by employee;
```

podemos utilizar la cláusula ON para establecer las columnas que deseamos que sean iguales. También podemos utilizar la cláusula USING, especificando los nombres de columna en lugar de predicados completos.

```
select e.ename, e.bdate
, h.deptno, h.msal
from employees e
join
history h
using (empno)
where e.job = 'ADMIN';
```

6.15. COMBINACIONES EXTERNAS (OUTER JOINS)

En el punto (anterior) "Combinaciones de Igualdad (equijoins)" hemos hecho esta consulta:

```
select d.deptno, d.location
, e.ename, e.init
from employees e, departments d
where e.deptno = d.deptno
order by d.deptno, e.ename;
```

Con el siguiente resultado:

```
DEPTNO   LOCATION  ENAME   INIT
-------- -------- -------- -----
10       NEW YORK  CLARK   AB
10       NEW YORK  KING    CC
10       NEW YORK  MILLER  TJA
20       DALLAS    ADAMS   AA
20       DALLAS    FORD    MG
20       DALLAS    JONES   JM
20       DALLAS    SCOTT   SCJ
20       DALLAS    SMITH   N
30       CHICAGO   ALLEN   JAM
30       CHICAGO   BLAKE   R
30       CHICAGO   JONES   R
30       CHICAGO   MARTIN  P
30       CHICAGO   TURNER  JJ
30       CHICAGO   WARD    TF
```

Si nos fijamos, nos damos cuenta de que no aparece el departamento 40. ¿Por qué razón? el departamento no existe en la tabla de empleados, ya que no hay ningún empleado perteneciente a este departamento.
Si deseamos que aparezca el departamento 40, independientemente de que no haya empleados asignados a tal departamento, pues lo podemos hacer realizando una "outer join".
Dos formas: Según la tabla que pongamos primero será una "right outer join" o una "left outer join".

Si ponemos empleados primero cogeremos todas las de la derecha (la segunda) aunque no tengan correspondencia con la primera:

```
select deptno, d.location
, e.ename, e.init
from employees e
right outer join
departments d
using (deptno)
order by deptno, e.ename;
```

Ename	deptno
CLARK	10
KING	10
MILLER	10
ADAMS	20
FORD	20
JONES	20
SCOTT	20
SMITH	20
ALLEN	30
BLAKE	30
JONES	30
MARTIN	30
TURNER	30
WARD	30

DEPTNO	LOCATION
10	NEW YORK
20	DALLAS
30	CHICAGO
40	BOSTON

Haced lo mismo con una left outer join (sin cambiar orden de las tablas) ¿qué ocurre?

```
select deptno, d.location
, e.ename, e.init
from employees e
left outer join
departments d
using (deptno)
order by deptno, e.ename;
```

Para que tuviera sentido, si lo que queremos es sacar todas los departamentos, aunque no tengan correspondencia con ningún empleado y se nos pide hacerlo con una left outer join, tendríamos que cambiar el orden y poner, lógicamente, primero departamentos.

6.16. LOS COMPONENTES DEL GROUP BY

Hasta ahora, hemos considerado sólo consultas que muestran información acerca de filas individuales. Cada fila del resultado de la consulta ha tenido una correspondencia uno-a-uno con algunos atributos de una determinada fila en la base de datos. Sin embargo, en la vida real, a menudo queremos saber información sobre un conjunto de filas.
Por ejemplo, es posible que deseemos mostrar el número de empleados (el recuento) por departamento. Para este tipo de consulta, es necesaria la cláusula GROUP BY del comando SELECT.

```
select e.deptno as "department"
, count(e.empno) as "number of employees"
from employees e
group by e.deptno;
```

Este ejemplo muestra la función CONTAR: cuenta el número de empleados por departamento. COUNT es un ejemplo de una función de grupo.

El resultado de esta consulta es una tabla, al igual que cualquier resultado de una consulta. Sin embargo, no hay una asociación de uno a uno entre las filas de la tabla empleados y las tres filas del resultado. En su lugar, se juntan los datos de empleados por departamento.

Para explicar cómo trabaja el operador GROUP BY, y cómo el lenguaje SQL se encarga de esta unión, el siguiente listado muestra una representación imaginaria de un resultado intermedio.

	e.EMPNO	e.JOB	e.MGR	e.MSAL	e.COMM	e.DEPTNO
FILA 1	7782	MANAGER	7839	2450	NULL	10
	7839	DIRECTOR	NULL	5000	NULL	
	7934	ADMIN	7782	1300	NULL	
FILA 2	7369	TRAINER	7902	800	NULL	20
	7566	MANAGER	7839	2975	NULL	
	7788	TRAINER	7566	3000	NULL	
	7876	TRAINER	7788	1100	NULL	
	7902	TRAINER	7566	3000	NULL	
FILA 3	7499	SALESREP	7698	1600	300	30
	7521	SALESREP	7698	1250	500	
	7654	SALESREP	7698	1250	1400	
	7698	MANAGER	7839	2850	NULL	
	7844	SALESREP	7698	1500	0	
	7900	ADMIN	7698	800	NULL	

Este listado muestra una pseudo-tabla con tres filas y seis columnas. Para facilitar la lectura hemos obviado algunas columnas de la tabla Empleados. En la última columna, se ven los tres números diferentes de departamentos, y las otras cinco columnas muestran conjuntos de valores de atributos. Algunos de estos valores contienen nulos, como e.COMM para los departamentos de 10 y 20, por ejemplo.

Si volvemos sobre la anterior consulta:

```
select e.deptno as "department"
, count(e.empno) as "number of employees"
from employees e
group by e.deptno;
```

ahora nos queda claro lo que hace el count: devuelve el número de empleados de cada departamento.

Agrupando muchas columnas

Podemos agrupar varias columnas separadas por comas.

Vamos a ver lo que tiene la tabla registrations con un simple "select * from registrations", nos damos cuenta de que sale el curso y la fecha repetida por cada empleado que se haya apuntado a dicho curso. Con la siguiente sentencia, no sabemos el número de cada empleado, pero sí el total de empleados en los determinados cursos (produce una visión general del número de inscripciones por curso)

```
select r.course, r.begindate
, count(r.attendee) as attendees
from registrations r
group by r.course, r.begindate;
```

El group by y los valores nulos

Si una expresión de columna en la que se aplica la cláusula GROUP BY contiene valores nulos, estos valores nulos aparecen juntos al final en un grupo separado. Ver ejemplo:

```
select e.comm, count(e.empno)
from employees e
group by e.comm;
```

6.17. FUNCIONES DE GRUPO

En el apartado anterior hemos utilizado la función COUNT para contar el número de empleados por departamento y el número de inscripciones por curso. COUNT, como ya sabemos, es una función de grupo. Todas las funciones de grupo tienen dos características en común:

- Se pueden aplicar sólo a conjuntos de valores.
- Devuelven un solo valor "sumatorio" o "agregado", derivado de ese conjunto de valores.

Es por eso que las funciones de grupo normalmente se presentan en combinación con GROUP BY (y, opcionalmente, la cláusula HAVING)
Las funciones más importantes de grupo de Oracle se muestran en la siguiente tabla:

Función	Descripción	Aplicable a
COUNT()	Número de valores	Todos los tipos de datos
SUM()	Suma de todos los valores	Datos numéricos
MIN()	Valor mínimo	Todos los tipos de datos
MAX()	Valor máximo	Todos los tipos de datos
AVG()	La media	Datos numéricos
MEDIAN()	La mediana (el valor central en un conjunto de valores ordenados)	Numéricos o de fechas
STATS_MODE()	Modus: el valor más frecuente	Todos los tipos de datos

Las funciones MIN y MAX son aplicables a cualquier tipo de datos, incluyendo las fechas y cadenas alfanuméricas. MIN y MAX sólo necesitan un criterio de ordenación para el conjunto de valores. La función AVG sólo se puede aplicar a números, porque el promedio se define como la suma dividida por el número de valores, y la función SUMA sólo acepta datos numéricos.

1. Seleccionar el departamento, el número de trabajos en cada departamento, la suma de las comisiones de los empleados de cada departamento, la media de los salarios y la mediana.

```
select e.deptno
        , count(e.job)
        , sum(e.comm)
        , avg(e.msal)
        , median(e.msal)
from employees e
group by e.deptno;
```

Funciones de grupo y valores duplicados

Si se aplica una función de grupo a un conjunto de valores de columna, el conjunto de valores puede contener valores duplicados. Por defecto, estos valores duplicados son tratados como valores individuales, lo que contribuye al resultado final de todas las funciones de grupo aplicadas al conjunto de valores.
Por ejemplo, tenemos cinco empleados del departamento 20, pero sólo tenemos dos puestos de trabajo distintos en ese departamento:

```
select deptno,count(empno)
 from employees
 group by deptno;
```

No obstante, este listado muestra 5 como resultado de COUNT (e.JOB) para el departamento 20.

Si se quiere en las funciones de grupo ignorar los valores duplicados se debe especificar la palabra clave DISTINCT.
A pesar de que es sintácticamente correcta, la adición de DISTINCT no tiene sentido para las funciones MIN y MAX.

Observad este ejemplo:

```
select count(deptno), count(distinct deptno)
, avg(comm), avg(coalesce(comm,0))
from employees;
```

Fijaros que se pueden utilizar las funciones de grupo sin una cláusula GROUP BY. La ausencia de una cláusula GROUP BY en combinación con la presencia de funciones de grupo en la cláusula SELECT siempre se traduce en un resultado de una sola fila. En otras palabras, la tabla completa se reúne en una sola fila.

Funciones de grupo y valores nulos

El estándar ANSI / ISO SQL decidió ignorar los valores nulos por completo. Sólo hay una excepción a esta regla: el COUNT (*).
Observemos la siguiente tabla para que veamos las consecuencias de esta premisa:

Valor X	SUM(X)	MIN(X)	AVG(X)	MAX(X)
{1,2,3,NULL}	6	1	2	3
{1,2,3,0}	6	0	1.5	3
{1,2,3,2}	8	1	2	3

2. Cuántos cursos se han programado diferentes para cada entrenador y el número total de cursos programados.

```
select trainer
, count(distinct course)
, count(*)
from offerings
group by trainer;
```

Resultado: aparentemente tenemos 3 cursos sin profesor asignado ¿no?

Agrupando los resultados de una join

3. Mostrar: curso, "trainer" y la media de calificaciones para cada "trainer" en todos los cursos que imparten (tablas: offerings y registrations: debéis estudiar los campos, lo que se os pide, hacer una valoración antes de hacer la select).

```
select r.course, avg(r.evaluation), o.trainer
from offerings o, registrations r
where o.course = r.course
and o.begindate = r.begindate
group by r.course, o.trainer;
```

La función COUNT(*)

Como se mencionó anteriormente, las funciones de grupo operan en un conjunto de valores, con una importante excepción. Además de los nombres de columna, se puede especificar el asterisco (*) como un argumento para la función COUNT. Esto amplía el alcance de la función COUNT: de una columna específica pasa a contar filas completas. **COUNT (*) devuelve el número de filas de todo el grupo.**

4. Contar el número de empleados por departamento

```
select e.deptno, count(*)
from employees e
group by e.deptno;
```

5. Obviamente, el departamento 40 no se encuentra en este resultado. Si queremos cambiar la consulta a fin de mostrar la 40 ¿qué haríamos?

```
select deptno, count(*)
from employees e
right outer join
departments d
using (deptno)
group by deptno;
```

6. ¿Qué pasa con esta consulta? Al parecer, de repente tenemos un empleado que trabaja para el departamento 40.

```
select deptno, count(e.empno)
from employees e
right outer join
departments d
using (deptno)
group by deptno;
```

Comparar los resultados del listado 5 y del 6. La única diferencia es el argumento de la función CONTAR. Al contar el e.EMPNO clave primaria, estaremos seguros de que todos los empleados "reales" serán contados, mientras que el valor nulo introducido por la combinación externa está bien que sea ignorado.
Necesitamos una combinación externa, y debemos asegurarnos de especificar el argumento correcto en la función CONTAR para obtener resultados correctos.

Cuidado: Debemos tener cuidado con la función COUNT, especialmente si hay valores nulos, ya que pueden causar problemas (las funciones de grupo los ignoran) y deseamos contar las ocurrencias de las filas.

Combinaciones válidas entre la SELECT y el GROUP BY

Si las consultas contienen una cláusula GROUP BY, algunas combinaciones de sintaxis no son válidas y el resultado será un error de Oracle, por ejemplo:

ORA-00937: not a single-group group function.

Esto significa que hay un desajuste entre la cláusula SELECT y su cláusula GROUP BY.
Veamos la siguiente tabla para observar la sintaxis válida frente a la inválida. Suponemos que tenemos una tabla T con 4 columnas: A, B, C y D.

SINTAXIS	VALIDO?
select a, b, max(c) from t ... group by a	No
select a, b, max(c) from t ... group by a,b	Yes
select a, count(b), min(c) from t ... group by a	Yes
select count(c) from t ... group by a	Yes

Los ejemplos en esta tabla muestran las siguientes dos reglas generales:

- **No es necesario seleccionar la expresión de la columna por la que se agrupa (véase el último ejemplo).**
- **Cualquier expresión de columna que no es parte de la cláusula GROUP BY sólo puede estar en la cláusula SELECT como un argumento de una función de grupo. Es por eso que el primer ejemplo no es válido.**

Hasta ahora, todos los ejemplos de GROUP BY que hemos hecho sólo mostraban los nombres de columna, pero también se pueden agrupar las expresiones de columna:

```
select case mod(empno,2)
        when 0 then 'EVEN '
                else 'ODD '
        end as empno
        , sum(msal)
from employees
group by mod(empno,2);
```

6.18. LA CLAÚSULA HAVING

Es posible que deseemos filtrar el resultado de una consulta sobre las filas agregadas, permitiendo sólo ciertos grupos en el resultado final. Esto lo podemos hacer mediante la cláusula HAVING.

7. Mostar los departamentos con más de cuatro empleados:

```
select deptno, count(empno)
from employees
group by deptno
having count(empno) > 4;
```

SQL permite escribir consultas con HAVING sin un GROUP BY anterior. En ese caso, Oracle asume como GRUPO implícito una expresión constante, al igual que cuando se utiliza las funciones de grupo en la cláusula SELECT sin especificar una cláusula GROUP BY, es decir, la tabla completa se trata como un solo grupo.

La diferencia entre el WHERE y el HAVING

Es importante que diferenciemos entre el WHERE y el HAVING. Fijaros:

```
select deptno, count(empno)
from employees
where bdate > date '1960-01-01'
group by deptno
having count(*) >= 4;
```

La condición del where siempre busca en las líneas individuales. Por otro lado, la condición donde aparece el COUNT(*) sólo tiene sentido a nivel de grupo. Es por esto que nunca debe aparecer una función de grupo en una cláusula WHERE.

HAVING sin funciones de grupo

El uso del HAVING sin funciones de grupo es muy raro. En el siguiente ejemplo se muestran 2 consultas, igualmente válidas, pero la segunda es más eficiente que la primera:

1)
```
select deptno, count(*)
        from employees
group by deptno
having deptno <= 20;
```

2)
```
select deptno, count(*)
        from employees
where deptno <= 20
group by deptno;
```

Un clásico error en SQL

Echa un vistazo a la siguiente consulta. Parece muy lógico, ¿no? ¿Quién gana más que el salario promedio?

```
select empno
from employees
where msal > avg(msal);
```

Sin embargo, si lo pensamos, el problema es evidente: en primer lugar, en la cláusula WHERE no es posible poner una función de grupo, y aunque se pudiera no puede comparar cada valor de cada registro con un valor que tiene que estar calculando del conjunto!!!

Consulta correcta:

```
select e.empno
from employees e
where e.msal > (select avg(x.msal)
from employees x );
```

Agrupando columnas adicionales

A veces, necesitamos realizar grupos en columnas adicionales que no parecen necesarias en el GROUP BY. Por ejemplo, supongamos que queremos ver el número de empleado y el nombre del empleado, seguido del número total de registros en los cursos; podríamos realizar algo así:

```
select e.empno, e.ename, count(*)
from employees e, registrations r
where e.empno = r.attendee
group by e.empno;
```

Vemos el error que nos da:

ERROR at line 1:
ORA-00979: not a GROUP BY expression

Efectivamente, para que resultara correcto, deberíamos haber agrupado por nombre.

Veamos el siguiente ejemplo:

```
select deptno
, sum(msal)
from employees;
```

ERROR at line 1:
ORA-00937: not a single-group group function

En ausencia de una cláusula GROUP BY, la función SUMA obtendrá una sola fila, mientras que DEPTNO produciría 14 números de departamento. Dos columnas con recuentos de filas diferentes no pueden se pueden mostrar al lado de un solo resultado.

Realizar la consulta correctamente.

```
select deptno, sum(msal)
from employees
group by deptno;
```

6.19. CARACTERÍSTICAS AVANZADAS DEL GROUP BY

En los apartados anteriores hemos visto ejemplos de las cláusulas GROUP BY "estándar". Vamos a avanzar un poco más con el GROUP BY.

Comencemos con un ejemplo: Sacar el departamento y el trabajo y el número de trabajadores que realizan cada trabajo en cada departamento:

```
select deptno, job, count(empno) numTrabaj
from employees
group by deptno, job;
```

Estamos obteniendo número de empleados por departamento y dentro de cada departamento por trabajo.

GROUP BY ROLLUP

Realicemos la siguiente consulta:

```
select deptno, job, count(empno) numTrabaj
from employees
group by ROLLUP(deptno, job);
```

Observemos el resultado y comparémoslo con el anterior. Se muestra el recuento de todos los puestos de trabajo por cada departamento y en la última fila se muestra el número total de empleados.
Es una cláusula muy útil para ciertas sentencias de este tipo.

GROUP BY CUBE

Veamos el siguiente ejemplo:

```
select deptno, job
, count(empno) headcount
from employees
group by CUBE(deptno, job);
```

Nos muestra más filas, porque en primer lugar te da el número de empleados en cada trabajo, y después te lo desglosa por departamento.

CUBE, ROLLUP Y LOS VALORES NULOS

Las sentencias CUBE y ROLLUP pueden generar algún valor nulo en los resultados, como hemos visto en los anteriores ejemplos; director no hay en el departamento 20. Podemos distinguir estos valores nulos con funciones como el GROUPING y el GROUPING_ID. Veamos la sintaxis:

GROUPING

```
select deptno
        , case GROUPING(job)
                when 0 then job
                when 1 then '**total**'
        end job
        , count(empno) headcount
from employees
group by rollup(deptno, job);
```

Desafortunadamente, la función GROUPING sólo puede tomar dos valores: 0 ó 1.

GROUPING_ID

Esta función es más flexible que el GROUPING, ya que puede devolver más valores:

```
select deptno
        , case GROUPING_ID(deptno, job)
                when 0 then job
                when 1 then '**dept **'
                when 3 then '**total**'
        end job
        , count(empno) headcount
from employees
group by rollup(deptno, job);
```

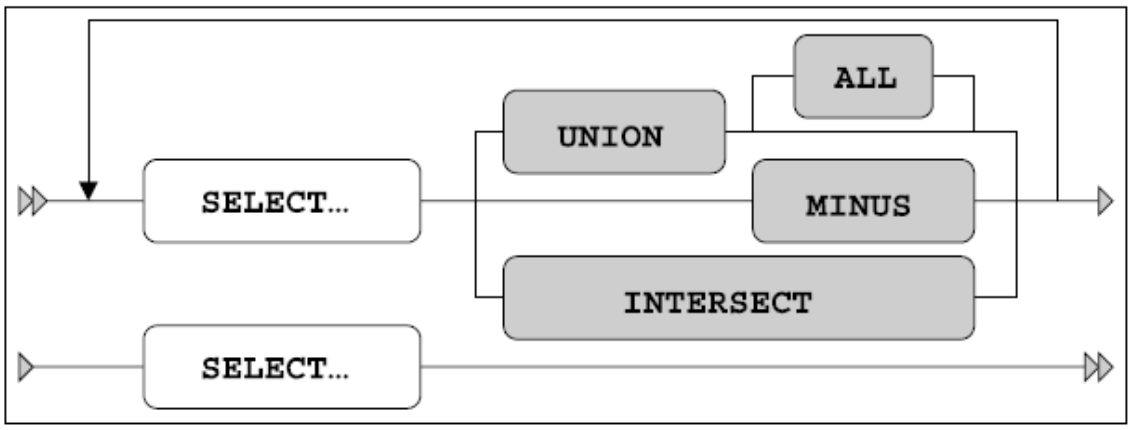
Se va evaluando tanto la salida del deptno como la del job. Cuando ha llegado al final de un departamento, el job será NULL (con lo que que corresponderá: deptno:0, job:1 (1 es NULL), es decir, en decimal: 01=1). Si ha llegado al final de todos los departamentos y de los trabajos, las dos salidas son NULL (11), y por eso será un 3, porque 11 binario en decimal es 3. En todos los demás casos, cuando saca deptno y job la salida será 0 (00), porque ninguno es NULL.

Realizar la siguiente consulta y observad resultados:

```
select deptno, job
                , GROUPING_ID(deptno, job) gid
from employees
group by cube(deptno, job);
```

6.20. LOS OPERADORES DE CONJUNTO: UNION, MINUS e INTERSECT

Se pueden usar estos operadores para combinar resultados de dos queries independientes y dar un único resultado.



Estos operadores corresponden con la unión, la intersección y la resta que conocemos de las matemáticas.

Operador	Resultado
Q1 UNION Q2	Todas las filas que están en Q1 o en Q2 o en ambas
Q1 UNION ALL Q2	Como la UNION, incluyendo duplicados
Q1 MINUS Q2	Las filas de Q1 sin las de Q2
Q1 INTERSECT Q2	Las filas que están en Q1 y en Q2

Por defecto los tres operadores suprimen las filas duplicadas en el resultado, con la excepción del UNION ALL, claro, que no las elimina. Esto supone para Oracle una ventaja porque no tiene que ordenar las filas. En los otros casos sí necesita la ordenación para saber cuáles están duplicadas y poder eliminarlas.
La UNION, RESTA e INTERSECCIÓN no pueden ser aplicadas arbitrariamente en dos queries cualesquiera. Los resultados por separado de las queries Q1 y Q2 deben ser compatibles

- Q1 y Q2 deben seleccionar el mismo número de expresiones de columnas.
- Los tipos de datos de las expresiones de columnas deben coincidir.
- La tabla resultado tomará los nombres o alias de Q1.
- Q1 no puede contener un ORDER BY
- Si ponemos un ORDER BY al final de la query, éste no se refiere a Q2, sino que afectará al resultado total.

Ejemplo: Mostrar las localidades donde se ofrecen cursos y en las cuales no hay departamento

```
select o.location from offerings o
MINUS
select d.location from departments d;
```

Lo mismo pero con una subquery:

```
select location
from offerings
where location not in (select location
                       from departments);
```

Comparar los resultados

Vemos como el minus automáticamente quita los duplicados.

¿Són las dos consultas equivalentes?, la lógica parece ser la misma, pero vemos que no son iguales. La primera consulta devuelve dos filas, Seattle y un valor nulo, que representa uno de los cursos que se ofrece en un lugar desconocido. El operador MINUS no quita el valor nulo.