
PRÁCTICA: 5. LENGUAJE DE CONSULTAS (SELECT) - III

1. <u>SOBRE LA PRÁCTICA</u>	2
1.1. OBJETIVOS	2
1.2. MATERIAL	2
1.3. CONTENIDOS	2
1.4. BIBLIOGRAFÍA.....	2
2. <u>AUTOCOMPOSICIÓN</u>	3
3. <u>CLÁUSULA ORDER BY</u>	4
4. <u>CLÁUSULA GROUP BY</u>	4
5. <u>CLÁUSULA HAVING</u>	8
6. <u>EJERCICIOS</u>	9

1. Sobre la Práctica

1.1. Objetivos

Conocer cómo se realiza la autocomposición de tablas. Conocer otras partes de la sentencia SELECT: ORDER BY, GROUP BY y HAVING

1.2. Material

ORACLE9 versión para WINDOWS XP

1.3. Contenidos

- Autocomposición
- Ordenación de Resultados de Consulta,
- Agrupación
- Funciones de Agregación
- Condiciones a la agrupación.

1.4. Bibliografía

GUÍA DE SQL
JAMES R. GROFF, PAUL N. WEINBERG
MCGRAW-HILL, 1998

ORACLE 9I: GUIA DE APRENDIZAJE
ABBEY, MICHAEL Y COREY, MIKE Y ABRAMSON, IAN
MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, S.A.

PROFESSIONAL ORACLE PROGRAMMING
GREENWALD, STACKOWIAK, DODGE, KLEIN, SHAPIRO, CHELLIAH
WROX 2005

2. Autocomposición

Es una composición de una tabla consigo misma. Para poder realizarla, obligatoriamente hay que emplear alias dentro de la orden SELECT que eviten los problemas de ambigüedad:

Por ejemplo, si tenemos la tabla paciente:

```
Paciente(nro, nombre, direccion, medico)
```

Medico → paciente

```
CREATE TABLE paciente(  
Nro INTEGER PRIMARY KEY,  
Nombre VARCHAR2(40),  
Direccion VARCHAR2(40),  
Medico INTEGER REFERENCES paciente);
```

```
INSERT INTO paciente VALUES (4, 'Lucía', 'Alarcos', NULL);  
INSERT INTO paciente VALUES (1, 'Paco', 'Alarcos', 4);  
INSERT INTO paciente VALUES (2, 'Luis', 'Azucena', 4);  
INSERT INTO paciente VALUES (3, 'Maria', 'Postas', 4);
```

Si se quisiera obtener el nombre de los médicos que viven en la misma dirección que sus pacientes, tendríamos que utilizar una autocomposición. Para ello, como ya hemos dicho, tendremos que utilizar alias para el select:

```
SELECT doctor.nombre  
FROM paciente doctor, paciente enfermo  
WHERE doctor.nro=enfermo.medico  
AND doctor.direccion=enfermo.direccion;
```

La salida sería:

NOMBRE
Lucía

3. Cláusula ORDER BY

Permite ordenar el resultado de la consulta.

ORDER BY <nombre de columna> [ASC | DESC] [...]

ASC: *Ascending* – Los valores se ordenaran ascendentemente

DESC: *Descending*- Los valores de la columna afectada estarán en orden descendente.

Se considera **null** el valor más alto posible.

Si se quisiera ordenar los resultados siguiendo varios principios, simplemente se tendrían que poner separados por comas y en el orden deseado las columnas. Es posible indicar para cada columna si el orden es ascendente o descendente, siempre teniendo en cuenta la prioridad al ordenar.

Por ejemplo, si queremos saber el nombre y las direcciones de los pacientes (que no son médicos), ordenados alfabéticamente por el nombre:

```
SELECT nombre, direccion
FROM paciente
WHERE medico IS NOT NULL
ORDER BY nombre;
```

La salida sería:

NOMBRE	DIRECCION
Luis	Azucena
Maria	Postas
Paco	Alarcos

4. Cláusula GROUP BY

Se utiliza cuando se quieren realizar consultas agrupadas por una columna. Así, se puede pensar en estos conjuntos de filas como grupos, utilizándose **funciones** de agrupación en la forma:

```
Select      [column1, ] FuncionesDeAgrupacion(columna)
From        table
[where      condition]
[group by   column]
[order by   column]
```

Las funciones de agregación más importantes son las siguientes:

Función	Descripción
AVG([DISTINCT ALL] n)	Valor medio de n, ignorando los valores nulos
COUNT ({* [DISTINCT ALL] expresion})	Número de filas donde la expresión evalúa algo distinto de null . Cuenta todas las filas seleccionadas cuando se usa con '*' incluyendo las repetidas.
MAX ([DISTINCT ALL] expr.)	Máximo valor de expr., ignorando valores nulos
MIN ([DISTINCT ALL] expr.)	Mínimo valor de expr., ignorando valores nulos
STDDEV ([DISTINCT ALL] n)	Desviación estándar de n ignorando valores nulos
SUM([DISTINCT ALL] expr.)	Suma de los valores de n, ignorando valores nulos
VARIANCE ([DISTINCT ALL] n)	Varianza de n ignorando valores nulos.

Tabla 1. Funciones de Agregación en SQL.

En las expresiones anteriores es importante tener claro el sentido de DISTINCT y el de ALL. Cuando se usa DISTINCT se tienen en cuenta sólo la primera vez que aparecen los valores, mientras que con ALL se tienen en cuenta todas las veces que aparecen.

Estas operaciones se pueden hacer sobre atributos que sean del tipo de dato: *char*, *varchar2*, *number* or *Date*.

Por defecto, el servidor de Oracle ordena los valores en orden ascendente cuando se usa la cláusula GROUP BY

(que también pueden ser utilizadas sin GROUP BY):

A continuación se van a proponer algunos ejemplos para ilustrar el uso de estas funciones de agregación. Supóngase la siguiente tabla de clientes, que representa a todos los comercios que pertenecen a una misma cadena de ámbito nacional con la estructura y el contenido mostrado:

ID	NOMBRE	CANT_VENTAS	FECHA	CIUDAD
1	Pepe	100	12/01/99	Toledo
2	Paco	200	13/08/98	Teruel
3	Lolo	400	01/06/78	Toledo
4	Lola	500	01/05/90	Toledo
5	Pepa	600	03/05/98	Teruel
6	Paca	200	02/07/97	Cuenca
7	Lali	100	06/05/99	Cuenca

Si se quiere saber el número de tiendas que tiene la cadena se podría poner:

SELECT count(id) as "Número de Tiendas" from cliente

Numero de Tiendas
7

Si se quisiera saber en cuántas ciudades distintas hay tiendas:

SELECT count(distinct ciudad) as "Número de Ciudades Con Tiendas de la Cadena" from

cliente

Y se obtendría:

NúmeroCiudadesConTiendasCadena
3

Se va a introducir ahora un ejemplo de utilización de funciones de agregación. Se está interesado en saber el número de tiendas que hay por ciudad. Para esto, tendríamos que agrupar los resultados de la consulta por ciudades. Para que quede más claro se va a incorporar en la consulta el nombre de la ciudad. Así la consulta:

SELECT ciudad, count(distinct id) as "NumeroTiendasPorCiudad" from cliente group by ciudad

Darían como resultado:

CIUDAD	NumeroTiendasPorCiudad
Cuenca	2
Teruel	2
Toledo	3

Si se quisiera obtener la media de las ventas realizadas por la cadena puedo obtenerlo con un select sobre toda la tabla:

select avg(cant_ventas) from clientes;

Obteniendo como salida:

AVG(CANT_VENTAS)
300

pero si lo que quiero son las ventas totales agrupadas por ciudades, entonces se debe realizar un agrupamiento por la columna ciudad.

select ciudad, avg(cant_ventas) from clientes group by ciudad;

Obtendré como salida:

CIUDAD	AVG(CANT_VENTAS)
Cuenca	150
Teruel	400
Toledo	333,333333

Siendo la desviación estándar

select ciudad, stddev(cant_ventas) from clientes group by ciudad

CIUDAD	STDDEV(CANT_VENTAS)
Cuenca	70,7106781
Teruel	282,842712
Toledo	208,1666

Y todo junto:

CIUDAD	AVG(CANT_VENTAS)	STDDEV(CANT_VENTAS)
Cuenca	150	70,7106781
Teruel	400	282,842712
Toledo	333,333333	208,1666

Hay que tener en cuenta que hay que agrupar todas las columnas que no aparezcan mencionadas en la cláusula group_by.

Ventas totales agrupadas por ciudad:

CIUDAD	SUM(CANT_VENTAS)
Cuenca	300
Teruel	800
Toledo	1000

Si se quisiesen conocer los máximos de ventas por ciudad:

select ciudad, max(cant_ventas) from cliente group by ciudad

CIUDAD	MAX(CANT_VENTAS)
Cuenca	200
Teruel	600
Toledo	500

MediaVentas
300

Y los mínimos:

select ciudad, min(cant_ventas) from cliente group by ciudad

CIUDAD	MIN(CANT_VENTAS)
Cuenca	100
Teruel	200
Toledo	100

Y en la misma consulta:

select ciudad, min(cant_ventas), max(cant_ventas) from cliente group by ciudad

Por ejemplo:

CIUDAD	MIN(CANT_VENTAS)	MAX(CANT_VENTAS)
Cuenca	100	200
Teruel	200	600
Toledo	100	500

```
select nombre, ciudad, sum(cant_ventas) from clientes group by nombre;
```

daría error ya que ciudad no está incluido en el group by. Correctamente sería:

```
select nombre, ciudad, sum(cant_ventas) from clientes group by nombre, ciudad;
```

5. Cláusula HAVING

No se pueden usar funciones de agrupación en la cláusula WHERE de un SELECT. O sea, no se puede usar el WHERE para, de forma selectiva eliminar datos que no interesan del resultado de una consulta agrupada.

Por ejemplo, en la tabla

```
exámenes(id asignatura, nro estudiante, nota, fecha)
```

Si hacemos:

```
SELECT nro_estudiante, avg(nota)
FROM exámenes
WHERE avg (nota) >6
GROUP BY nro_estudiante;
```

Darían un error por usar una función de agrupamiento en el WHERE

La cláusula HAVING hace una función parecida a la del WHERE cuando se trabaja con este tipo de funciones. Así, para listar aquellos alumnos cuya media es mayor que 6 sería:

```
SELECT nro_estudiante, avg(nota)
FROM exámenes
GROUP BY nro_estudiante
HAVING avg (nota) >6;
```

El campo referenciado en la cláusula HAVING no puede tener más de un valor por grupo. Esto significa que, en la práctica, HAVING sólo puede referenciar a funciones de agregación y columnas que se están usando en el GROUP by

6. EJERCICIOS

1. Crear una tabla EMPLEADO que contenga los siguientes campos:

empleado (cod, nombre, apellido, calle, jefe)
jefe → empleado

Rellenar la tabla.

Obtener el nombre y apellido de un empleado y su jefe siempre y cuando ambos tengan el mismo apellido.

Borrar la tabla empleado.

2. Obtener el número de profesores en activo de cada área agrupados según su categoría.
3. Obtener el número de profesores en activo de cada departamento y área agrupados según su categoría
4. Listar el código y nombre de cada departamento y el número de profesores numerarios en activo que tiene.
5. Obtener todos los posibles tríos de profesores de la misma área de conocimiento. Se debe tener en cuenta que no deben aparecer tríos repetidos, por ejemplo (a,b,c)=(c,b,a).