



Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 3
Consultas avanzadas
Parte 3

5. Eliminación de duplicados - Agregación - Grupos

Antes que nada tenemos que asumir el hecho de que SQL hace uso de relaciones que son bags y no sets, y que por tanto una tupla puede aparecer más de una vez en una relación.

Eliminación de duplicados

Una relación, siendo un set, no puede tener más de una copia de una tupla dada. Cuando un consulta SQL crea una nueva relación, el sistema SQL **no elimina duplicados**. De esta manera, la respuesta SQL a una consulta puede listar la misma tupla **varias veces**.

Una de las varias definiciones equivalentes del significado de un query **select-from-where** es que comenzamos con el producto Cartesiano de las relaciones referenciadas en la cláusula `FROM`.

Cada tupla del producto es testeada por la condición de la cláusula `WHERE`, y las que pasan el test son proporcionadas a la salida para su projection se acuerdo a la cláusula `SELECT`.

Esta projection puede provocar que obtengamos -de diferentes tuplas en el producto- **una misma tupla** en el resultado.

Ni siquiera es necesario que esto suceda: como nada impide que una relation SQL **posea duplicados**, las relaciones a partir de las cuales se forma el producto Cartesiano **ya pueden tener duplicados**, y cada copia idéntica será "emparejada" con las tuplas de las otras relaciones, derivando en una **proliferación de duplicados** en el producto.

Si no queremos duplicados en el resultado, debemos agregar a la cláusula `SELECT` la keyword `DISTINCT`. Esta keyword le indica a SQL que produzca sólo una copia de cualquier tupla.

Ejemplo

Supongamos que queremos obtener un listado de los tipos de publicaciones (atributo type). Podemos ejecutar la siguiente consulta:

```
SELECT type
FROM titles
```

Obtenemos:

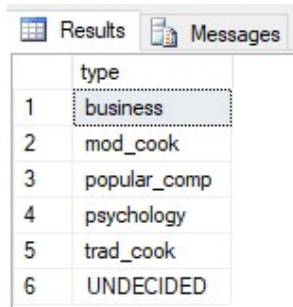
	type
1	business
2	business
3	business
4	business
5	mod_cook
6	mod_cook
7	UNDECIDED
8	popular_comp
9	popular_comp
10	popular_comp
11	psychology
12	psychology
13	psychology
14	psychology
15	psychology
16	trad_cook
17	trad_cook
18	trad_cook

Sucede que un tipo de publicación aparece tantas veces como publicaciones de ese tipo se hayan editado, obtendremos cada tipo de publicación varias veces.

Si queremos obtener cada tipo de publicación una única vez, podemos cambiar la consulta por:

```
SELECT DISTINCT type
FROM titles
```

...y obtenemos:



	type
1	business
2	mod_cook
3	popular_comp
4	psychology
5	trad_cook
6	UNDECIDED

Lo que está haciendo `DISTINCT` es forzar a que el resultado de una operación sea un **set**.

Ejemplo

Supongamos que queremos obtener los nombres de la editoriales que han editado publicaciones del autor con código '998-72-3567'.
Podríamos definir la siguiente consulta:

```
Select P.pub_name
FROM authors A INNER JOIN titleauthor TA
ON A.au_id = TA.au_id
INNER JOIN Titles T
ON TA.title_id = T.title_id
INNER JOIN Publishers P
ON T.pub_id = P.pub_id
WHERE A.au_id = '998-72-3567'
```

...pero el autor ha publicado más de un libro para la misma editorial, obtendremos en la salida, el nombre de la editorial, tantas veces como libros tenga el autor publicados en ella:

Results		Messages	
	pub_name		
1	New Moon Books		
2	New Moon Books		

Si queremos obtener cada editorial una única vez, podemos cambiar la consulta por:

```
SELECT DISTINCT P.pub_name
FROM authors A INNER JOIN titleauthor TA
ON A.au_id = TA.au_id
INNER JOIN Titles T
ON TA.title_id = T.title_id
INNER JOIN Publishers P
ON T.pub_id = P.pub_id
WHERE A.au_id = '998-72-3567'
```

...y obtendremos cada editorial una única vez:

Results		Messages	
	pub_name		
1	New Moon Books		

6. Agregación en SQL.

Supongamos la siguiente tabla de Alumnos de un curso:

```
CREATE TABLE Alumnos
(
    IDAlumno integer,
    Apellido varchar(30),
    Nombres VARCHAR(40),
    LocOrigen VARCHAR(30),
    Edad SMALLINT,
    CalFinal SMALLINT
)
```

Alumnos		
IDAlumno	integer	<pk>
Apellido	varchar(40)	
Nombres	varchar(40)	
LocOrigen	varchar(30)	
Edad	smallint	
CalFinal	smallint	

```
INSERT Alumnos
SELECT 1, 'Lopez', 'Juan', 'Santa Fe', 19, 8

INSERT Alumnos
SELECT 2, 'Cuatrini', 'Jose', 'Sunchales', 20, 7

INSERT Alumnos
SELECT 3, 'Bordon', 'Carlos', 'Paraná', 21, 9

INSERT Alumnos
SELECT 4, 'Tinta', 'Raul', 'Santa Fe', 21, 8

INSERT Alumnos
SELECT 5, 'Castellanos', 'Rodrigo', 'Paraná', 19, 6
```

A estos atributos -Apellido, Edad, Localidad origen, etc.- les podemos llamar "atributos de tupla" o de entidad.

Si en cambio nos interesa saber cuál es el promedio de edades de los alumnos del curso, ese ya no es un atributo de tupla. Es un atributo de grupo.

Lo mismo si quiero saber cuál es la edad mínima de los alumnos o si quiero saber calificación media al final del cursado.

Este tipo de consultas se resuelven con operadores de agregación, que son todos **atributos de grupo**.

Operadores de agregación.

SQL usa los cinco operadores de agregación: SUM, AVG, MIN, MAX y COUNT.

Estos operadores son usados aplicándolos a una expresión de valor escalar -normalmente un nombre de columna- en una cláusula SELECT.

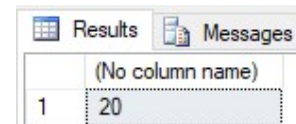
Una excepción es la expresión COUNT(*), que cuenta todas las tuplas en la relación que es construida a partir de la cláusula FROM y la cláusula WHERE del query.

Además, usando la keyword DISTINCT tenemos la opción de eliminar duplicados de la columna **antes de aplicar el operador de agregación**.

Es decir, una expresión como COUNT(DISTINCT x) cuenta la cantidad de valores diferentes en la columna x.

Volviendo a nuestro ejemplo de tabla Alumnos, podemos obtener el promedio de edades con:

```
SELECT AVG(Edad)
FROM Alumnos
```



Results		Messages
	(No column name)	
1	20	

No existen otros atributos que puedan acompañar al operador de agregación en la lista de salida del SELECT, simplemente porque todos los demás atributos de los que disponemos son atributos de tupla y no de grupo.

Por ejemplo, en:

```
SELECT Apellido, AVG(Edad)
FROM Alumnos
```

...obtenemos un error, ya que estamos mezclando un atributo de tupla (Apellido) con un atributo de grupo (AVG(Edad))

Otros ejemplos:

```
SELECT MIN(Edad)
FROM Alumnos
```

Results		Messages
(No column name)		
1	19	

```
SELECT AVG(CalFinal)
FROM Alumnos
```

Results		Messages
(No column name)		
1	8	

```
SELECT COUNT(*)
FROM Alumnos;
```

```
SELECT COUNT(type)
FROM Titles;
```

Results		Messages
(No column name)		
1	18	

Si queremos asegurarnos de que no se cuenten más de una vez los valores duplicados, podemos usar la keyword `DISTINCT` antes del atributo agregado:

```
SELECT COUNT(DISTINCT type)
FROM Titles;
```

Results		Messages
(No column name)		
1	6	

Así, cada tipo de publicación es contado una única vez, más allá de que aparezca en varias publicaciones.

7. Grupos

En SQL, la cláusula `GROUP BY` nos permite agrupar filas en base a los valores de una o más columnas. `GROUP BY` divide el resultado de la sentencia `SELECT` en un conjunto de grupos, de manera que para cada grupo los valores de la o las columnas que definen el grupo son idénticos. `GROUP BY` se ubica luego de la cláusula `WHERE`:

```
SELECT columnal
FROM tabla
GROUP BY columnal
```

La cláusula `GROUP BY` es una cláusula que, de aparecer en una consulta, la modifica radicalmente: una vez definidos grupos en SQL, solo podemos especificar *atributos de grupo* en la *lista de salida* del `SELECT`.

La keyword `GROUP BY` es seguida por una lista de **grouping attributes**. En la situación más simple, habrá una referencia a una única relación en la cláusula `FROM`, y esta relación tendrá sus tuplas agrupadas de acuerdo a los valores de sus **grouping attributes**.

Cualquier operador de agregación que utilicemos en la cláusula `SELECT` se aplicará solo dentro de los grupos.

Ejemplo

Supongamos que queremos encontrar el promedio de precios de cada tipo de publicación, podemos escribir la siguiente consulta:

```
SELECT type, AVG(price)
FROM titles
GROUP BY type
```

	type	(No column name)
1	business	13,73
2	mod_cook	11,49
3	popular_comp	21,475
4	psychology	13,504
5	trad_cook	15,9633
6	UNDECIDED	NULL

Podemos imaginar a las tuplas de la relación Titles reorganizadas y agrupadas de manera tal que todas las tuplas del tipo "business" están juntas, todas las de "popular_comp" están juntas, etc., tal como se sugiere en la siguiente figura:

	type	
	business	
	business	
	business	
	mod_cook	
	mod cook	
	...	
	...	

Se calcula entonces el promedio de los componentes price de todas las tuplas en cada grupo, y, por cada grupo, se da salida al atributo type junto con la cifra obtenida.

Si bien las consultas que involucran `GROUP BY` generalmente poseen grouping attributes y agregaciones en su cláusula `SELECT`, no es necesario que existan ambos. Por ejemplo, podemos escribir:

```
SELECT type
FROM Titles
GROUP BY type;
```

7.1. Condiciones de grupo

De manera análoga a como `WHERE` nos permite especificar una condición que deben satisfacer las filas que formarán parte de la lista de salida, la cláusula `HAVING` nos permite especificar una condición que deben cumplir los grupos para formar parte de la lista de salida:

```
SELECT columnal
FROM tabla
GROUP BY columnal
HAVING condicion-de-grupo
```

Debemos tener en cuenta que, la condición de una cláusula `HAVING` sólo puede incluir comparaciones contra *atributos de grupo*.

Ejemplo

Volvamos a nuestra consulta anterior sobre la relación Titles, donde queríamos encontrar el promedio de precios de cada tipo de publicación:

Titles		
<u>title_id</u>	varchar(6)	<pk>
title	varchar(80)	
type	char(12)	
pub_id	char(4)	
price	float	
advance	float	
royalty	integer	
ytd_sales	integer	
notes	varchar(200)	
pubdate	datetime	

```
SELECT type, AVG(price)
FROM titles
GROUP BY type
```

Puede que queramos filtrar las filas por algún atributo de tupla **antes** de agrupar. Por ejemplo, si solo queremos considerar las publicaciones de la editorial con código '0877', podemos modificar la consulta por:

```
SELECT type, AVG(price)
FROM titles
WHERE pub_id = '0877'
GROUP BY type
```

Results		
Messages		
	type	(No column name)
1	mod_cook	11,49
2	psychology	21,59
3	trad_cook	15,9633
4	UNDECIDED	NULL

Como estamos eliminando tuplas -a través de la cláusula WHERE- **antes** de agrupar, habrán grupos que simplemente quedarán sin tuplas (vacíos)

A veces, en cambio queremos elegir nuestros grupos en base a un **atributo de grupo**.

Entonces debemos seguir la cláusula `GROUP BY` por una cláusula `HAVING`.

Esta última cláusula consiste de la keyword `HAVING` seguida de una condición acerca del grupo.

Ejemplo

Supongamos que queremos modificar nuestra consulta anterior sobre la relación `Titles`, a fin de considerar solo los grupos en los cuales su fecha de publicación más antigua sea posterior al 1 de octubre de 1991:

```
SELECT type, AVG(price)
FROM titles
WHERE pub_id = '0877'
GROUP BY type
HAVING MIN(pubdate) > '1991-10-01'
```

Donde antes obteníamos:

	type	(No column name)
1	mod_cook	11,49
2	psychology	21,59
3	trad_cook	15,9633
4	UNDECIDED	NULL

...ahora obtenemos:

	type	(No column name)
1	psychology	21,59
2	UNDECIDED	NULL

Los grupos correspondientes a los tipos `mod_cook` y `trad_cook` han sido descartados de la lista de salida del `SELECT`.

Esto se debe a que **no cumplen con la condición del grupo**.

La fecha más antigua de publicación de `mod_cook` es 9 de Junio de 1991 y la fecha más antigua de publicación de `trad_cook` es 12 de junio de 1991

Ejemplo

Supongamos que tenemos la siguiente consulta, que lista el apellido y nombre de los autores junto a la publicación más barata y más cara –de su autoría- a la venta.

```
SELECT A.au_lname, A.au_fname, MIN(price), MAX(price)
FROM authors A INNER JOIN titleauthor TA
ON A.au_id = TA.au_id
INNER JOIN Titles T
ON TA.title_id = T.title_id
GROUP BY A.au_lname, A.au_fname
```

Obtenemos:

	au_lname	au_fname	(No column name)	(No column name)
1	Bennet	Abraham	19,99	19,99
2	Blotchet-Halls	Reginald	11,95	11,95
3	Carson	Cheryl	22,95	22,95
4	DeFrance	Michel	2,99	2,99
5	del Castillo	Innes	19,99	19,99
6	Dull	Ann	20,00	20,00
7	Green	Marjorie	2,99	19,99
8	Gringlesby	Burt	14,99	14,99
9	Hunter	Sheryl	20,00	20,00
10	Karsen	Livia	21,59	21,59
11	Locksley	Charlene	7,99	7,99
12	MacFeather	Steams	11,95	21,59
13	O'Leary	Michael	11,95	14,99
14	Panteley	Sylvia	20,95	20,95
15	Ringer	Albert	7,00	10,95
16	Ringer	Anne	2,99	10,95
17	Straight	Dean	19,99	19,99
18	White	Johnson	19,99	19,99
19	Yokomoto	Akiko	14,99	14,99

Supongamos que ahora se nos solicita que se listen solo los autores que poseen exactamente dos publicaciones. Podríamos escribir:

```
SELECT DISTINCT A.au_lname, A.au_fname, MIN(price), MAX(price)
  FROM authors A INNER JOIN titleauthor TA
                ON A.au_id = TA.au_id
                INNER JOIN Titles T
                ON TA.title_id = T.title_id
 GROUP BY A.au_lname, A.au_fname
HAVING COUNT(T.title_id) = 2
```

...y ahora obtenemos:

	au_lname	au_fname	(No column name)	(No column name)
1	Green	Marjorie	2,99	19,99
2	Locksley	Charlene	7,99	7,99
3	MacFeather	Steams	11,95	21,59
4	O'Leary	Michael	11,95	14,99
5	Ringer	Albert	7,00	10,95
6	Ringer	Anne	2,99	10,95

8. La expresión CASE

8.1. CASE simple

La expresión CASE simple

La expresión CASE simple compara un valor contra una lista de valores y retorna un resultado asociado al primer valor coincidente:

```
CASE expresion0
  WHEN expresion1 THEN ResultadoExpresion1
  [ [WHEN expresion2 THEN ResultadoExpresion2]
    [...]
  [ ELSE ResultadoExpresionN]
End
```

La expresion0 se compara contra expresion1, expresion2, etc. hasta que se encuentra una coincidencia o se encuentra el final de la lista de expresiones. Si CASE encuentra coincidencia, retorna el ResultadoExpresion correspondiente. Si no encuentra coincidencia alguna, retorna ResultadoExpresionN.

```
SELECT CASE type
  WHEN 'business' THEN 'Negocios'
  ELSE 'Otros'
END tipo
FROM titles
```


8.2. CASE searched

La expresión CASE searched

Esta forma de expresión CASE permite el análisis de varias condiciones lógicas y retorna un resultado asociado a la primera que evalúa a verdadero:

```
CASE
  WHEN condicion1 THEN expresion11
  [ [WHEN condicion2 THEN expresion2]
    [...] ]
  [ ELSE expresionN]
End
```

Si se encuentra una condición verdadera, el resultado es la expresión correspondiente. Si todas las condiciones son falsas, el resultado es expresionN.

```
SELECT lname,
       CASE
         WHEN job_lvl < 100 THEN 'Puntaje menor que 100'
         WHEN job_lvl < 200 THEN 'Puntaje entre 100 y 200'
         ELSE 'Puntaje mayor que 200'
       END Nivel
FROM Employee
ORDER BY 2,1
```

```
Select type, price + case TYPE
                        when 'business' then 100
                        else 160
                      end incremento
from titles
```

9. El operador UNION

El operador `UNION` une los resultados (tuplas) de dos o más consultas en un único conjunto.

Dos reglas básicas para combinar los conjuntos de resultados de dos consultas con `UNION` son:

- a) El número y el orden de las columnas deben ser idénticos en todas las consultas.
- b) Los tipos de datos deben ser compatibles.

Por omisión, `UNION` elimina las filas duplicadas (obtenemos un conjunto). Si especificamos la cláusula `ALL`, se incluyen en la salida las filas duplicadas.

Ejemplo

```
SELECT type, title
  FROM titles
 WHERE type = 'business'

UNION ALL

SELECT type, title
  FROM titles
 WHERE type = 'popular_comp'
```