Material Imprimible

Curso de SQL Server

Módulo 6: Conectando distintos conjuntos de datos I

DISTINCT

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviados en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros, si utilizamos esta sentencia:

select autor from libros;

Aparecen repetidos. Para obtener la lista de autores sin repetición usamos:

select distinct autor from libros;

UNION Y UNION ALL

El propósito del comando SQL **UNION** es combinar los resultados de dos consultas juntas. En este sentido, **UNION** es parecido a **Join**, ya que los dos se utilizan para información relacionada en múltiples tablas. Una restricción de **UNION** es que todas las columnas correspondientes necesitan ser del mismo tipo de datos. También, cuando utilizamos **UNION**, sólo se seleccionan valores distintos (similar a **SELECT DISTINCT**).

La sintaxis es la siguiente:

[Instrucción SQL 1]
UNION
[Instrucción SQL 2];

Supongamos que tenemos las siguientes dos tablas,

Tabla ***Store_Information***

Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

Tabla ***Internet_Sales***

07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320
12-Jan-1999	750

y deseamos saber de todas las fechas donde hay una operación de venta. Para hacerlo, utilizamos la siguiente instrucción SQL:

```
SELECT Txn_Date FROM Store_Information
UNION
SELECT Txn_Date FROM Internet_Sales;
```

Resultado:

Txn_Date

05-Jan-1999

07-Jan-1999

08-Jan-1999

10-Jan-1999

11-Jan-1999

12-Jan-1999

El propósito del Comando SQL **UNION ALL** es también combinar los resultados de dos consultas juntas. La diferencia entre **UNION ALL** y **UNION** es que, mientras **UNION** sólo selecciona valores distintos, **UNION ALL** selecciona todos los valores.

La sintaxis para **UNION ALL** es la siguiente:

[Instrucción SQL 1]
UNION ALL
[Instrucción SQL 2];

Utilicemos el mismo ejemplo de la sección anterior para ilustrar la diferencia. Supongamos que tenemos las siguientes dos tablas,

Tabla ***Store_Information***

Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

Tabla ***Internet_Sales***

07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320
12-Jan-1999	750

y deseamos encontrar las fechas en donde se realizó una operación de venta en un negocio como así también las fechas donde hay una venta a través de Internet. Para hacerlo, utilizamos la siguiente instrucción SQL:

```
SELECT Txn_Date FROM Store_Information
UNION ALL
SELECT Txn_Date FROM Internet_Sales;
```

Resultado:

Txn_Date
05-Jan-1999
07-Jan-1999
08-Jan-1999
08-Jan-1999
07-Jan-1999
10-Jan-1999
11-Jan-1999
12-Jan-1999

CASE

CASE se utiliza para brindar un tipo de lógica "si-entonces-otro" para SQL. Su sintaxis es:

```
SELECT CASE ("nombre_columna")
  WHEN "condición1" THEN "resultado1"
  WHEN "condición2" THEN "resultado2"
  ...
  [ELSE "resultadoN"]
  END
FROM "nombre_tabla";
```

"condición" puede ser un valor estático o una expresión. La cláusula ELSE es opcional.

En nuestra Tabla ***Store_Information*** de ejemplo,

Tabla ***Store_Information***

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

si deseamos multiplicar las sumas de ventas de 'Los Angeles' por 2 y las sumas de ventas de 'San Diego' por 1,5, ingresamos,

```
SELECT Store_Name, CASE Store_Name
  WHEN 'Los Angeles' THEN Sales * 2
  WHEN 'San Diego' THEN Sales * 1.5
  ELSE Sales
  END
"Nuevas Ventas",
Txn_Date
FROM Store_Information;
```

"Nuevas Ventas" es el nombre que se le otorga a la columna con la instrucción **CASE**. Resultado:

Store_name	Nuevas Ventas	Txn_Date
Los Angeles	**3000**	**05-Jan-1999**
San Diego	**375**	**07-Jan-1999**
San Francisco	**300**	**08-Jan-1999**
Boston	**700**	**08-Jan-1999**

GROUP BY

Ahora regresamos a las funciones de agregados. ¿Recuerda que utilizamos la palabra clave **SUM** para calcular las ventas totales para todos los negocios? ¿Y si quisiéramos calcular el total de ventas para *cada* negocio? Entonces, necesitamos hacer dos cosas: Primero, necesitamos asegurarnos de que hayamos **seleccionado** el nombre del negocio, así como también las ventas totales. Segundo, debemos asegurarnos de que todas las sumas de las ventas estén **GROUP BY** negocios. La sintaxis SQL correspondiente es,

```
SELECT "nombre1_columna", SUM("nombre2_columna")
FROM "nombre_tabla"
GROUP BY "nombre1-columna";
```

Ilustremos utilizando la siguiente tabla,

Tabla ***Store_Information***

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

Deseamos saber las ventas totales para cada negocio. Para hacerlo, ingresaríamos,

```
SELECT Store_Name, SUM(Sales)
FROM Store_Information
GROUP BY Store_Name;
```

Resultado:

Store_Name	SUM(Sales)
Los Angeles	**1800**
San Diego	**250**
Boston	**700**

La palabra clave **GROUP BY** se utiliza cuando estamos seleccionado columnas múltiples desde una tabla (o tablas) y aparece al menos un operador aritmético en la instrucción **SELECT**. Cuando esto sucede, necesitamos **GROUP BY** todas las otras columnas seleccionadas, *es decir*, todas las columnas excepto aquella(s) que se operan por un operador aritmético.

HAVING

Otra cosa que la gente puede querer hacer es limitar el resultado según la suma correspondiente (o cualquier otra función de agregado). Por ejemplo, podríamos desear ver sólo los negocios con ventas mayores a 1 500 €, dólares. En vez de utilizar la cláusula WHERE en la instrucción SQL, a pesar de que necesitemos utilizar la cláusula HAVING, que se reserva para funciones de agregados. La cláusula HAVING se

coloca generalmente cerca del fin de la instrucción SQL, y la instrucción SQL con la cláusula HAVING. puede o no incluir la cláusula GROUP BY sintaxis para HAVING es,

SELECT "nombre1_columna", SUM("nombre2_columna")
FROM "nombre_tabla"
GROUP BY "nombre1_columna"
HAVING (condición de función aritmética);
Nota: La cláusula GROUP BY es opcional.

En nuestro ejemplo, tabla ***Store_Information***,

Tabla ***Store_Information***

Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

ingresaríamos,

```
SELECT Store_Name, SUM(Sales)
FROM Store_Information
GROUP BY Store_Name
HAVING SUM(Sales) > 1500;
```

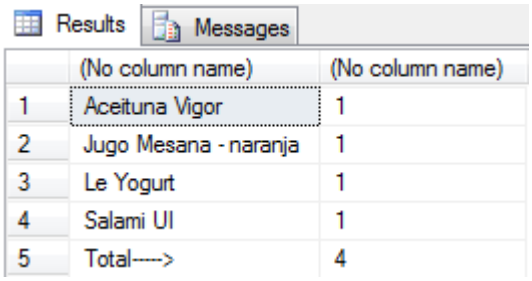
Resultado:

Store_Name	SUM(Sales)
Los Angeles	**1800**

ROLL UP

ROLLUP es muy útil cuando se requiere generar reportes que contengan totales o subtotales. Este trabaja generando un conjunto de resultados. Agrega una nueva fila con este resultado general. Ejemplo:

```
SELECT
CASE WHEN nombre IS NULL THEN 'Total—–>' ELSE nombre END,COUNT(*)
FROM Producto
GROUP BY nombre
WITH ROLLUP
```



Results | Messages

	(No column name)	(No column name)
1	Aceituna Vigor	1
2	Jugo Mesana - naranja	1
3	Le Yogurt	1
4	Salami UI	1
5	Total----->	4

Fuentes: http://cpcppw.es/mysql-registros-duplicados.html
https://sites.google.com/site/mgvtjjbtbdd/union
https://www.1keydata.com/es/sql/sql-union.php