Práctica 08

Lenguaje de Manipulación de Datos (DML) Avanzado

1. Objetivo General

Profundizar en el uso de palabras reservadas de SQL para construir consultas específicas.

2. Objetivos Secundarios

- Crear scripts (código en SQL) de mayor nivel para el caso de inserciones, actualizaciones y consultas.
- Incrementar la probabilidad de dar respuesta a diversos problemas a través de una base de datos.

3. Introducción

Sabemos que uno de los objetivos de las bases de datos es ofrecer soluciones a través del correcto y oportuno manejo de información. Sin embargo, en muchas ocasiones el entorno del problema o la naturaleza de éste incrementan su complejidad, lo que obliga a los usuarios de las base de datos a proveer soluciones eficaces.

Para crear estas soluciones mediante una base de datos y un SMBD que maneje la solución, utilizaremos el lenguaje SQL. Este lenguaje nos permite manipulaciones básicas como lo es la inserción, actualización, consulta y eliminación de los datos, tal como se trabajo en *Lenguaje de Manipulación de Datos (DML) Básico* (Práctica 07).

Construir consultas de mayor nivel requiere de algunas destrezas que se adquieren a través del estudio del problema y la experiencia personal en el manejo de éste lenguaje. Algunas de estas destrezas son:

1. *Entendimiento del problema*
   El primer paso para construir una solución es entender cuál es el problema, en qué contexto se encuentra inmerso y cuáles son los beneficios deseados por los involucrados.

2. *Planteamiento de posibles soluciones*
   Teniendo claro el punto anterior, se pueden construir secuencias de pasos lógicos que nos lleven a resolverlo. El conjunto de estas secuencias formará un conjunto de posibles soluciones.

3. *Traducción del planteamiento a código SQL*
   Los pasos lógicos identificados se traducen formalmente a código SQL como instrucciones precisas dirigidas al SMBD para que las ejecute.

4. *Análisis de resultados*

Los resultados que arroja el código en forma de tuplas pueden mostrar diferencias debido a los diversos caminos que se siguieron. Es por eso que es de suma importancia validar los resultados obtenidos contra los resultados esperados.

5. *Post optimización de código*
   En el caso de que los resultados obtenidos nos sean los esperados, deberán de replantearse los pasos 2 y 3. Por otra parte pudiese ser que los resultados sí sean los esperados pero el desempeño del SMBD no cumpla con las expectativas de tiempo o espacio, en estos casos el código SQL deberá ser optimizado como se explicará posteriormente en *Optimización de Consultas* (Práctica 09).

3.1. Condiciones y Agrupamientos

SQL cuenta con operadores que permiten construir consultas más complejas. En la Figura 8.1 se muestra ésta sintaxis.

```
SELECT nombre_columna1, nombre_columna2, ...
FROM nombre_tabla
WHERE condición
GROUP BY atributo_agrupador
HAVING condición_de_agrupación;
```

Figura 8.1 - Sintaxis avanzada de consultas en SQL.

En la sintaxis de la Figura 8.1 encontramos:

1. SELECT
   Palabra reservada para indicar que atributos se mostrarán como resultado de la consulta.
2. nombre_columna
   Es el nombre de la columna, o columnas separadas por comas, que queremos consultar.
3. FROM
   Palabra reservada para indicarle al SMBD la tabla de la cual obtener la columna que necesitamos.
4. nombre_tabla
   Es el nombre de la tabla que contiene la columna o columnas que solicitamos.
5. WHERE
   Palabra reservada para indicar bajo qué condición seleccionaremos los registros que queremos consultar.
6. condicion
   Es una característica o conjunto de características que tiene que cumplir los registros para ser seleccionados.

7. GROUP BY

   Palabras reservadas para agrupar los datos de una columna dependiendo de su valor.

8. atributo_agrupador

   Es el nombre de la columna, o columnas separadas por comas, cuyos valores dirigen la agrupación.

9. HAVING

   Palabra reservada para indicar que se requiere filtrar el o los grupos que se formaron en el punto 7 y 8. Su función es equivalente a WHERE con la diferencia de que únicamente es usado para grupos.

10. condición_de_agrupación

    Característica o conjunto de características que tiene que cumplir un grupo para ser seleccionado.

11. ;

    Indica la finalización de la instrucción.

Para explicar con claridad lo descrito anteriormente, se tomará como ejemplo el conjunto de las siguientes tablas mostradas en las Tablas 8.1 a 8.5 obtenidas del ejemplo de la Pizzería.

Tabla 8.1 - Tabla Cliente de la Pizzería

CLIENTE			
Id_Cliente	***Nombre***	***Ap_Materno***	***Ap_Paterno***
108	Isidro	Juárez	Altamirano
105	Jorge	Arzola	Cervantes
100	Edna	Pérez	León
109	Estefanía	González	Leyva
101	David	Avila	Martínez
106	Carlos	Lozano	Morales
104	Norma	Ponce	Noriega
110	María	Meza	Segura
107	Ricardo	Navarro	Tapia
103	Sara	Monroy	Vega
102	Genaro	López	Villegas

Tabla 8.2 - Tabla Orden de la Pizzería

ORDEN			
Id_Orden	Fecha	Numero_Articulos	Total
20507	19/04/2013	1	62.72
20511	21/04/2013	4	292.04
20501	15/04/2013	5	347.90
20504	16/04/2013	6	486.08
20517	24/04/2013	7	583.10
20505	16/04/2013	8	568.40
20513	22/04/2013	9	899.64
20514	22/04/2013	11	846.72
20502	15/04/2013	15	1014.30
20515	22/04/2013	15	911.40
20510	21/04/2013	17	1123.08
20508	19/04/2013	19	1229.90
20503	15/04/2013	23	2325.54
20516	23/04/2013	23	1865.92
20509	19/04/2013	33	3350.62
20512	22/04/2013	36	2380.42
20506	18/04/2013	42	2840.04

Tabla 8.3 - Tabla Detalle de Orden de la Pizzería

DETALLE_DE_ORDEN			
Id_Orden	Id_Articulo	Cantidad	Total_Por_Art
20511	A-3	1	59.78
20501	B-3	1	62.72
20507	B-3	1	62.72
20511	C-4	1	67.62
20501	A-5	1	82.32
20505	B-4	1	99.96
20504	C-4	2	135.24
20503	C-5	2	150.92
20504	C-5	2	150.92
20505	A-5	2	164.64
20511	A-5	2	164.64
20504	B-4	2	199.92
20501	C-4	3	202.86
20514	B-5	4	270.48
20516	C-5	4	301.84
20505	A-4	5	303.8
20502	B-5	5	338.1
20508	A-3	7	418.46
20510	C-4	8	540.96
20514	A-5	7	576.24
20510	C-2	9	582.12
20517	C-6	7	583.1
20502	C-4	10	676.2
20503	A-6	7	679.14
20512	B-3	11	689.92
20508	C-4	12	811.44
20513	B-4	9	899.64
20515	A-4	15	911.4
20509	A-6	14	1358.28
20503	A-2	14	1495.48
20516	A-5	19	1564.08
20512	B-5	25	1690.5
20509	B-2	19	1992.34
20506	C-4	42	2840.04

Tabla 8.4 - Tabla Costo de la Pizzería

COSTO	
Id_Articulo	***Costo_Unitario***
A-3	59.78
A-4	60.76
B-3	62.72
C-2	64.68
B-5	67.62
C-4	67.62
C-5	75.46
A-5	82.32
C-6	83.3
A-6	97.02
B-4	99.96
B-2	104.86
A-2	106.82

Tabla 8.5 - Tabla Orden por Cliente de la Pizzería

ORDEN_POR_CLIENTE	
Id_Orden	***Id_Cliente***
20506	101
20517	102
20507	102
20511	103
20512	104
20516	105
20514	105
20502	106
20509	107
20510	107
20503	108
20504	108
20515	109
20501	109
20513	110
20508	110
20505	110

La Figura 8.2 muestra una consulta de varias columnas que pertenecen a una misma tabla con dos condiciones.

```
SELECT Id_Orden, Id_Articulo, Cantidad, Total_Por_Art
FROM Detalle_De_Orden
WHERE Cantidad > 4
AND Total_Por_Art > 500;
```

Figura 8.2. Consulta de la tabla Detalle_Orden.

Como resultado esta consulta arroja una tabla con todos los registros que cumplen que el valor de Cantidad sea mayor a 4 y que el Total por Art sea mayor a 500. Ver Tabla 8.6.

Tabla 8.6 - Tabla resultante de la consulta de la Figura 8.2.

DETALLE_DE_ORDEN			
Id_Orden	Id_Articulo	Cantidad	Total_Por_Art
20510	C-4	8	540.96
20514	A-5	7	576.24
20510	C-2	9	582.12
20517	C-6	7	583.1
20502	C-4	10	676.2
20503	A-6	7	679.14
20512	B-3	11	689.92
20508	C-4	12	811.44
20513	B-4	9	899.64
20515	A-4	15	911.4
20509	A-6	14	1358.28
20503	A-2	14	1495.48
20516	A-5	19	1564.08
20512	B-5	25	1690.5
20509	B-2	19	1992.34
20506	C-4	42	2840.04

En la consulta de la Figura 8.2, se muestra una consulta sencilla que utiliza dos condiciones mediante el uso del operador lógico AND (Y en inglés); este operador permite definir una nueva condición cada que vez que se incluye en el código, con la restricción de que todas las condiciones definidas seguidas de un AND tienen que resultar verdaderas simultáneamente para que el registro sea seleccionado y mostrado como resultado.

Existen otros operadores lógicos, el operador OR (O en inglés) permite definir una nueva condición cada vez que se incluye en el código, a diferencia de AND, en este caso al menos una de las condiciones definidas tiene que resultar verdadera para que el registro sea seleccionado y mostrado como resultado.

En otras palabras, si se usa AND todas las condiciones tienen que resultar verdaderas para seleccionar el registro; en el caso de OR, bastará con que una condición resulte verdadera para que el registro sea seleccionado.

El operador NOT (NO en inglés) permite cambiar el resultado de la condición, es decir, si la condición resultara verdadera, la palabra NOT la convertirá en falsa y viceversa.

Dentro de las condiciones se utilizan operadores matemáticos, que sirven como herramientas para comparar dos valores, es decir, es un operador lógico que regresa *Verdadero* o *Falso* dependiendo si se cumple o no la comparación matemática de ambos valores. En la Figura 8.3 se muestran los posibles operadores:

=	Compara si dos valores son iguales
<	Compara si el valor de la izquierda es menor que el de la derecha
>	Compara si el valor de la izquierda es mayor que el de la derecha
< >	Compara si dos valores son diferentes

Figura 8.3 - Operadores Básicos para comparación.

Otros operadores funcionan también comparando valores de manera más explícita, lo que permite crear consultas más especializadas. Algunos ejemplos de estos se muestran en la Figura 8.4.

BETWEEN	Compara si un valor se encuentra dentro de los parámetros que el usuario puede definir
LIKE	Compara si un valor cumple con un patrón que el usuario puede definir
SIMILAR TO	Compara si un valor cumple con un patrón que el usuario puede definir de manera más general
IN	Compara si un valor existe dentro de una lista de posibles valores definida por el usuario
NOT IN	Compara si un valor no existe dentro de una lista de posibles valores definida por el usuario

Figura 8.4 - Operadores Explícitos para comparación.

Para ejemplificar algunos de estos operadores, en la Figura 8.5, se muestra una consulta que regresa los registros cuyo nombre comienza con la letra S o el nombre es Edna o Carlos.

```
SELECT Nombre
FROM Cliente
WHERE Nombre LIKE 'S%'
OR Nombre IN ('Edna' , 'Carlos');
```

Figura 8.5 - Consulta de la tabla Nombre.

En esta sintaxis analizaremos únicamente la clausula WHERE y sus operadores:

1. WHERE
   Palabra reservada para indicar bajo qué condición seleccionaremos los registros que queremos consultar.
2. Nombre
   Es la columna que queremos comparar, es decir, los valores de cada registro de la columna Nombre de la tabla Cliente serán comparados.

3. LIKE
   Es el operador de comparación que utilizaremos.
4. '
   La comilla simple indica la apertura para escribir una cadena de caracteres.
5. S
   La letra S mayúscula indica que solo se seleccionarán aquellos registros en los que el atributo nombre empiece con la letra S mayúscula.
6. %
   El porcentaje, es un comodín que deja libre el número y tipo de carácter que siga de la letra “S” del punto anterior. Es decir, discrimina el resto de la cadena. Existe el símbolo “_” (guión bajo) que indica solo la posición de un carácter dentro de una cadena, es decir, el lugar que ocupa un determinado carácter dentro de una cadena. Por ejemplo, LIKE ‘_a%’ regresaría todos los registros cuyos valores coincidan con que su segundo carácter sea la letra “a” minúscula, no importando cual sea el primer carácter ni cuales sean del tercero al último.
7. '
   La comilla simple indica el cierre de la escritura de la cadena de caracteres.
8. OR
   Indica que se definirá una nueva condición y que si un registro cumple al menos una de ellas, entonces será seleccionado.
9. Nombre
   Es la columna que queremos comparar, es decir, los valores de cada registro de la columna Nombre de la tabla Cliente serán comparados.
10. IN
    Indica que a continuación se definirá una lista de valores. Si un valor de la columna Nombre de la tabla Cliente se encuentra en ésta lista, será seleccionado por el SMBD.
11. (
    La apertura de paréntesis indica el comienzo de la lista de valores a buscar en la columna Nombre de la tabla Cliente.
12. ‘Edna’ , ‘Carlos’
    Son los valores que se deberán buscar en la columna Nombre de la tabla Cliente.
13.)
    El cierre de paréntesis indica el término de la lista de valores.
14. ;
    Indica la terminación de la instrucción.

Esta consulta seleccionará de los registros de la tabla Cliente que se muestran en la Tabla 8.1 El resultado final de la consulta se muestra en la Tabla 8.7, estos registros son mostrados ya que cumplen con alguna de las condiciones.

Tabla 8.7 - Tabla final de la consulta de la Figura 8.5.

CLIENTE
Nombre
Sara
Edna
Carlos

Notemos que solo nos regresa una columna ya que en la consulta solo se especificó la columna *Nombre* de la tabla *Cliente*.

El operador NOT IN es la negación del operador IN, es decir, el SMBD seleccionará todos los registros excepto los que se encuentren en la lista definida en la consulta.

A manera de repaso, la consulta mostrada en la Figura 8.6 utiliza diferentes operadores.

```
SELECT *
FROM Orden
WHERE Numero_Articulos BETWEEN 5 AND 11
AND Fecha NOT IN ('22/04/2013', '18/04/2013');
```

Figura 8.6 - Consulta a la tabla *Orden.*

En esta sintaxis únicamente se detallarán los elementos nuevos, tenemos entonces:

1. *

   El asterisco es el método abreviado para seleccionar todas las columnas de una tabla.
2. BETWEEN

   Es el operador de comparación que se utilizará y verificará que el valor del atributo se encuentre dentro de un rango especificado.
3. 5

   Primer parámetro del rango para comparar los valores de la columna en cuestión.
4. AND

   Palabra reservada para definir el segundo parámetro del rango del operador BETWEEN. No se debe confundir con el operador AND para agregar una nueva condición.
5. 11

   Segundo parámetro del rango para comparar los valores de la columna en cuestión.
6. AND

Indica que se agregará una nueva condición. La condición previa, en este caso BETWEEN, y la que se definirá a continuación deberán de resultar verdaderas para que el registro sea seleccionado.

7. Fecha
   Nombre de la columna cuyos valores se compararán.
8. NOT IN
   Indica que se definirá una lista de valores. Los registros, cuyos valores de la columna Fecha que se encuentren en ésta lista, no se seleccionarán.
9. ('22/04/2013' , '18/04/2013')
   Listado de valores a comparar, nótese que al ser una fecha ésta sigue un formato establecido.

La tabla resultante de la consulta que se encuentra en la Figura 8.6, se muestra en la Tabla 8.8.

Tabla 8.8 - Tabla resultante de la consulta de la Figura 8.7.

ORDEN			
Id_Orden	***Fecha***	***Numero_Articulos***	***Total***
20501	15/04/2013	5	347.90
20504	16/04/2013	6	486.08
20517	24/04/2013	7	583.10
20505	16/04/2013	8	568.40

Es posible observar que se incluyeron los resultados de las órdenes que tienen entre 5 y 11 artículos. Por otro lado los registros cuya fecha haya sido el 22/04/2013 o 18/04/2013 no se incluyeron en la lista debido a la cláusula AND de la consulta NOT IN.

3.2. Funciones de Agregación

Cuando el dato no se encuentra implícito en la base, es decir, que no está físicamente almacenado pero es posible derivarlo a partir del valor de otro atributo, se deberá calcular mediante funciones de agregación. Por ejemplo para contar los registros que se obtuvieron de la consulta anterior o para sumar los totales de las órdenes.

Las funciones de agregación se indican en el renglón de SELECT. En la Figura 8.7 se muestra una consulta que utiliza funciones de agregación y la cláusula GROUP BY.

```
SELECT Fecha, SUM(Total)
FROM Orden
GROUP BY Fecha;
```

Figura 8.7 - Consulta con GROUP BY a la tabla Orden.

En esta sintaxis encontramos:

1. SUM
   Palabra reservada, en este caso representa a la función suma. Es importante mencionar que la función SUM solo recibe valores de tipo numérico, en otro caso se provocará un error.
2. (Total)
   Indica la columna cuyos valores serán sumados.
3. GROUP BY
   Palabras reservadas para indicar qué columna tomará para realizar la agrupación. Este operador selecciona todos los valores diferentes que existen en una columna e indica a la función de agregación que realice su operación agrupando los resultados por cada valor de la columna que encontró el GROUP BY.
4. Fecha
   Nombre de la columna sobre la cual se realizará el agrupamiento.

La consulta de la Figura 8.8, que arrojará la suma de los totales agrupados por fecha de las Órdenes. El resultado de esta consulta se muestra en la Tabla 8.9.

Tabla 8.9 - Resultado de la consulta de la Figura 8.8.

ORDEN	
Fecha	***SUM (Total)***
15/04/2013	3687.74
16/04/2013	1054.48
18/04/2013	2840.04
19/04/2013	4643.24
21/04/2013	1415.12
22/04/2013	5038.18
23/04/2013	1865.92
24/04/2013	583.1

Otras funciones de agregación existentes son: COUNT, MIN, MAX y AVG. COUNT regresa el número de registros en la columna señalada. Si se utiliza COUNT (*) se cuentan todos los registros de la tabla. Por otra parte si se utiliza COUNT DISTINCT sólo se contarán los registros que son diferentes; es decir, que si un valor en una columna se encuentra repetido, esta función de agregación solo lo contará una vez.

MIN y MAX traen solo el registro cuyo valor sea el mínimo o máximo de una columna determinada o una agrupación. Esta función sólo puede recibir datos de tipo numérico. AVG es la función de agregación para determinar el promedio de los valores de una columna o agrupación.

3.3. Funciones de Composición

De no existir los datos en una misma tabla, existen operadores que permiten integrar la información de dos o más tablas. A estos operadores se les conoce como JOINs. En la Figura 8.8 se muestra un ejemplo de un NATURAL JOIN entre dos tablas.

```
SELECT Id_Orden, Id_Cliente, Total
FROM Orden NATURAL JOIN Orden_por_Cliente;
```

Figura 8.8 - Sintaxis de NATURAL JOIN entre tablas.

En esta sintaxis encontramos:

1. Orden
   Nombre de una de las tablas necesarias para obtener las columnas que se definieron en la cláusula SELECT.
2. NATURAL JOIN
   Palabra reservada para indicar que se realizará una composición entre dos tablas.
3. Orden_por_Cliente
   Nombre de una de las tablas necesarias para obtener las columnas que se definieron en la cláusula SELECT.

Esta consulta arrojará como resultado lo definido en la Tabla 8.10.

Tabla 8.10 - Resultado de la consulta de la Figura 8.8.

Id_Orden	***Id_Cliente***	***Total***
20501	109	347.90
20502	106	1014.30
20503	108	2325.54
20504	108	486.08
20505	110	568.40
20506	101	2840.04
20507	102	62.72
20508	110	1229.90
20509	107	3350.62
20510	107	1123.08
20511	103	292.04
20512	104	2380.42
20513	110	899.64
20514	105	846.72
20515	109	911.40
20516	105	1865.92
20517	102	583.10

Existen otros tipos de conectores de tablas como lo son LEFT OUTER JOIN, RIGHT OUTER JOIN, INNER JOIN y CROSS JOIN, mismos que serán descritos a continuación.

- **INNER JOIN**
  Permite especificar una condición para realizar la composición de registros. Para determinar ésta condición se utiliza la palabra reservada **ON**. Es importante notar que en la siguiente sintaxis en la condición **ON**, se encuentra el nombre de la tabla que contiene la columna a la cual queremos hacer referencia separados por un punto (Costo.Id_Articulo). Esto se hace para evitar ambigüedad en los nombres de las columnas que se encuentran en ambas tablas. Ver Figura 8.9.

```
SELECT *
FROM Costo INNER JOIN Detalle_de_Orden
ON Costo.Id_Articulo = Detalle_de_Orden.Id_Articulo
```

Figura 8.9 - Sintaxis de INNER JOIN entre tablas.

- **LEFT OUTER JOIN** y **RIGHT OUTER JOIN**

  Realiza la composición entre dos tablas, es decir, regresa un registro de la primera tabla por cada match que haga con la segunda tabla, asignando NULL a todas las columnas de aquellos registros que no tengan correspondencia con la segunda tabla, ver Figura 8.10.

```
SELECT *
FROM Cliente LEFT JOIN Orden_por_Cliente
ON Cliente.Id_Cliente = Orden_por_Cliente.Id_Cliente
```

Figura 8.10 - Sintaxis de NATURAL JOIN entre tablas.

Para el caso de LEFT los valores NULL serían asignados a los atributos sin correspondencia de la segunda tabla. Ver Tabla 8.11

Tabla 8.11 - Resultado aplicar un LEFT OUTER JOIN.

Id_Cliente	***Nombre***	***Ap_Materno***	***Ap_Paterno***	***Id_Orden***	***Id_Cliente***
101	David	Avila	Martínez	20506	101
102	Genaro	López	Villegas	20517	102
102	Genaro	López	Villegas	20507	102
103	Sara	Monroy	Vega	20511	103
104	Norma	Ponce	Noriega	20512	104
105	Jorge	Arzola	Cervantes	20516	105
105	Jorge	Arzola	Cervantes	20514	105
106	Carlos	Lozano	Morales	20502	106
107	Ricardo	Navarro	Tapia	20509	107
107	Ricardo	Navarro	Tapia	20510	107
108	Isidro	Juárez	Altamirano	20503	108
108	Isidro	Juárez	Altamirano	20504	108
109	Estefanía	González	Leyva	20515	109
109	Estefanía	González	Leyva	20501	109
110	María	Meza	Segura	20513	110
110	María	Meza	Segura	20508	110
110	María	Meza	Segura	20505	110
100	Edna	Pérez	León		

Para la tabla anterior se puede apreciar que el valor del último registro para las columnas Id_Orden y la segunda de Id_Cliente aparecen en blanco (NULL para PostgreSQL). Esto quiere decir que el cliente con id = 100 no tiene ordenes expedidas.

Mientras que para RIGHT JOIN, los valores NULL se asignarán a los atributos sin correspondencia de la primera tabla. Ver Tabla 8.12.

Tabla 8.12 - Resultado de aplicar un RIGHT OUTER JOIN.

Id_Orden	***Id_Articulo***	***Cantidad***	***Total_por_Art***	***Id_Articulo***	***Costo_Unitario***
20508	A-3	7	418.46	A-3	58.78
20511	A-3	1	59.78	A-3	58.78
20515	A-4	15	911.4	A-4	60.76
20505	A-4	5	303.8	A-4	60.76
20512	B-3	11	689.92	B-3	62.72
20507	B-3	1	62.72	B-3	62.72
20501	B-3	1	62.72	B-3	62.72
20510	C-2	9	582.12	C-2	64.68
20512	B-5	25	1690.5	B-5	67.62
20502	B-5	5	338.1	B-5	67.62
20514	B-5	4	270.48	B-5	67.62
20506	C-4	42	2840.04	C-4	67.62
20508	C-4	12	811.44	C-4	67.62
20502	C-4	10	676.2	C-4	67.62
20510	C-4	8	540.96	C-4	67.62
20501	C-4	3	202.86	C-4	67.62
20504	C-4	2	135.24	C-4	67.62
20511	C-4	1	67.62	C-4	67.62
20516	C-5	4	301.48	C-5	75.46
20504	C-5	2	150.92	C-5	75.46
20503	C-5	2	150.92	C-5	75.46
20516	A-5	19	1564.08	A-5	82.32
20514	A-5	7	576.24	A-5	82.32
20511	A-5	2	164.64	A-5	82.32
20505	A-5	2	164.64	A-5	82.32
20501	A-5	1	82.32	A-5	82.32
20517	C-6	7	582.12	C-6	83.3
20509	A-6	14	1358.28	A-6	97.02
20503	A-6	7	679.14	A-6	97.02
20513	B-4	9	899.64	B-4	99.96
20504	B-4	2	199.92	B-4	99.96
20505	B-4	1	99.96	B-4	99.96
20509	B-2	19	1992.34	B-2	104.86
20503	A-2	14	1495.48	A-2	106.82
				Z-99	666

- **CROSS JOIN** es una unión entre LEFT y RIGHT OUTER JOIN, regresando el producto cartesiano de los registros de ambas tablas. Es decir, si la primera tabla almacena 5 registros y la segunda almacena 7 registros, el total de registros será 5 x 7 = 35. Ver Tabla 8.13

Tabla 8.13 - Resultado de aplicar un CROSS JOIN.

Id_Cliente	*Nombre*	*Ap_Materno*	*Ap_Paterno*	*Id_Orden*	*Id_Cliente*
108	Isidro	Juárez	Altamirano	20506	101
105	Jorge	Arzola	Cervantes	20506	101
100	Edna	Pérez	León	20506	101
109	Estefanía	González	Leyva	20506	101
101	David	Avila	Martínez	20506	101
106	Carlos	Lozano	Morales	20506	101
104	Norma	Ponce	Noriega	20506	101
110	María	Meza	Segura	20506	101
107	Ricardo	Navarro	Tapia	20506	101
103	Sara	Monroy	Vega	20506	101
102	Genaro	López	Villegas	20506	101
108	Isidro	Juárez	Altamirano	20517	102
105	Jorge	Arzola	Cervantes	20517	102
100	Edna	Pérez	León	20517	102
109	Estefanía	González	Leyva	20517	102
101	David	Avila	Martínez	20517	102
106	Carlos	Lozano	Morales	20517	102
104	Norma	Ponce	Noriega	20517	102
110	María	Meza	Segura	20517	102
107	Ricardo	Navarro	Tapia	20517	102
103	Sara	Monroy	Vega	20517	102
102	Genaro	López	Villegas	20517	102
⋮	⋮	⋮	⋮	⋮	⋮
108	Isidro	Juárez	Altamirano	20508	110
105	Jorge	Arzola	Cervantes	20508	110
100	Edna	Pérez	León	20508	110
109	Estefanía	González	Leyva	20508	110
101	David	Avila	Martínez	20508	110
106	Carlos	Lozano	Morales	20508	110
104	Norma	Ponce	Noriega	20508	110
110	María	Meza	Segura	20508	110
107	Ricardo	Navarro	Tapia	20508	110
103	Sara	Monroy	Vega	20508	110
102	Genaro	López	Villegas	20508	110
108	Isidro	Juárez	Altamirano	20505	110
105	Jorge	Arzola	Cervantes	20505	110
100	Edna	Pérez	León	20505	110
109	Estefanía	González	Leyva	20505	110
101	David	Avila	Martínez	20505	110
106	Carlos	Lozano	Morales	20505	110
104	Norma	Ponce	Noriega	20505	110
110	María	Meza	Segura	20505	110
107	Ricardo	Navarro	Tapia	20505	110
103	Sara	Monroy	Vega	20505	110
102	Genaro	López	Villegas	20505	110

3.4. Vistas.

Una vista se puede definir como una “tabla virtual” ya que se comporta como una tabla al realizar consultas sobre ella pero, a diferencia de una tabla, no siempre es posible realizar modificaciones sobre los datos de ella.

Dicho de otra manera, la información que conforma una vista es extraída de las tablas existentes en la base de datos, se puede decir que es información virtual, ya que es una vista de los datos reales de la base de datos.

Las vistas también pueden ser utilizadas para simplificar consultas complejas, al crear “tablas virtuales” sobre las cuales se realizarán consultas más simples.

3.4.1. Creación de Vistas.

La creación de Vistas presenta la siguiente sintaxis.

Sintaxis:

```
CREATE OR REPLACE VIEW nombredevista (columnavirtual1,
columnavirtual2) AS consulta
```

Argumentos:

- `CREATE OR REPLACE`
  Palabra reservada en PostgreSQL para la creación de Vistas. CREATE OR REPLACE indica que se desea crear o reemplazar, en caso de que exista previamente, la definición de una vista anterior por la actual. OR REPLACE se puede omitir si no se desea sobrescribir una vista que ya existe.
- `VIEW`
  Palabra reservada en PostgreSQL que indica que el objeto que se creará o reemplazará es una vista.
- `nombredevista`
  Indica el nombre de la vista que será creada.
- `(`
  Paréntesis que abre, indica que se iniciará el nombrado de columnas virtuales (estas columnas son independientes de las columnas “reales” que ya existen en las tablas de la base de datos) que conformarán la vista.
- `columnavirtual1`
  Indica el nombre de la primera columna virtual en la vista, se tiene la posibilidad de poner una o más columnas virtuales como argumentos.
- `,`
  El símbolo ‘coma’ indica la separación de los argumentos.
- `)`

Paréntesis que cierra indica el final de la instrucción de los argumentos de las columnas virtuales.

- AS
  Palabra reservada en PostgreSQL, que permite dar un nombre a la vista creada, mediante este nombre la vista puede ser utilizada posteriormente.
- consulta
  Esta consulta es una sentencia, o también llamado query, que permite la extracción de la información en la base de datos, generalmente usada con un SELECT.

Uso:

El uso de la creación de una vista se presenta a continuación, ver Figura 8.1.

En donde se utilizó la tabla “Alumnos” con “nombre”, “apellidopaterno”, “numerocuenta”, “promedio” y “domicilio” como columnas. Generando así la Vista PROMEDIOALUMNOS con NOMBRE, APELLIDO_PATERNO, NUMERO_CUENTA y PROMEDIO como columnas virtuales y restringiendo el promedio a números mayores a 5.

```
CREATE VIEW PROMEDIOALUMNOS (NOMBRE, APELLIDO_PATERNO, NUMERO_CUENTA, PROMEDIO)
       AS SELECT nombre, apellidopaterno, numerocuenta, promedio
       FROM "Alumnos" WHERE promedio > 5;
```

Figura 8.1 Creación de Vista PROMEDIOALUMNOS.

Para consultar la Vista se hace una selección de todas las columnas en PROMEDIOALUMNOS, ver Figura 8.2.

```
SELECT * FROM PROMEDIOALUMNOS;
```

Figura 8.2 Consulta de Vista PROMEDIOALUMNOS.

El resultado obtenido se muestra como sigue, ver Figura 8.3.

	nombre character varying(255)	apellido_paterno character varying(255)	numero_cuenta integer	promedio integer
1	Amena	Darrel	284281500	9
2	Ivor	Ryan	234834590	9
3	Zia	Merrill	345571442	7
4	Lunea	Cathleen	324852628	10
5	Tate	Ivory	326224860	7
6	Grace	Abdul	245966651	6
7	Duncan	Fuller	252106361	7
8	Emily	Kay	383860378	7
9	Zephania	Shad	340190434	7
10	Mufutau	Jameson	246584589	7
11	Leandra	Shaine	203616233	7
12	Cally	Illiana	294213447	6
13	Perry	Jordan	312988387	7
14	Ashton	Raya	230347956	8
15	Jenette	Alexandra	230630067	9
16	Reagan	Scarlet	337614845	10
17	Finn	Kevin	392244393	6
18	Brynn	Tucker	284693824	6
19	Jasmine	Violet	314753806	10
20	Ira	Brynne	290008151	8
21	Brett	Herrod	232108729	10

Figura 8.3 Resultado de consulta en Vista PROMEDIOALUMNOS.

3.4.2. Manejo de Vistas.

El modificar las vistas permite reemplazar una consulta ya creada por una nueva. Si se desea modificar la Vista PROMEDIOALUMNOS creada anteriormente, se deberá colocar la instrucción OR REPLACE como sigue, ver Figura 8.4.

```
CREATE OR REPLACE VIEW PROMEDIOALUMNOS (NOMBRE, APELLIDO_PATERNO, NUMERO_CUENTA, PROMEDIO, DOMICILIO)
       AS SELECT nombre, apellidopaterno, numerocuenta, promedio, domicilio
       FROM "Alumnos" WHERE promedio > 5;
```

Figura 8.4 Reemplazo de Vista PROMEDIOALUMNOS.

En este caso se agregó la columna DOMICILIO quedando el resultado como sigue, ver Figura 8.5.

	nombre character varying(255)	apellido_paterno character varying(255)	numero_cuenta integer	promedio integer	domicilio character varying(255)
1	Amena	Darrel	284281500	9	Ap #438-6519 Null
2	Ivor	Ryan	234834590	9	130-8221 Aenean R
3	Zia	Merrill	345571442	7	P.O. Box 988, 411
4	Lunea	Cathleen	324852628	10	799-9959 Placerat
5	Tate	Ivory	326224860	7	Ap #907-6405 At A
6	Grace	Abdul	245966651	6	Ap #977-3890 Pede
7	Duncan	Fuller	252106361	7	Ap #723-8665 Erat
8	Emily	Kay	383860378	7	7499 Quisque St.
9	Zephania	Shad	340190434	7	Ap #543-2769 Metu
10	Mufutau	Jameson	246584589	7	464-1906 Nec Ave
11	Leandra	Shaine	203616233	7	319-4119 Ornare.
12	Cally	Illiana	294213447	6	P.O. Box 608, 213
13	Perry	Jordan	312988387	7	P.O. Box 322, 649
14	Ashton	Raya	230347956	8	742-6949 Tincidun
15	Jenette	Alexandra	230630067	9	914-6489 Duis Rd.
16	Reagan	Scarlet	337614845	10	466-2519 Fusce St
17	Finn	Kevin	392244393	6	P.O. Box 198, 622
18	Brynn	Tucker	284693824	6	Ap #873-3117 Done
19	Jasmine	Violet	314753806	10	3094 Eu Ave
20	Ira	Brynne	290008151	8	P.O. Box 489, 470
21	Brett	Herrod	232108729	10	Ap #166-1264 Tell

Figura 8.5 Resultado de Vista PROMEDIOALUMNOS.

Para eliminar las vistas existentes, se deberá utilizar la instrucción DROP de la siguiente manera, ver Figura 8.6.

```
DROP VIEW PROMEDIOALUMNOS;
```

Figura 8.6 Eliminación de Vista PROMEDIOALUMNOS.

PostgreSQL en sus versiones anteriores a 9.3 no permitía al usuario realizar ninguna modificación sobre los datos de la “tabla virtual” que arrojaba una vista. A manera de ejemplo, en la Figura 8.7, se muestra un intento de eliminación de tuplas de una vista en su versión de PostgreSQL 9.2.

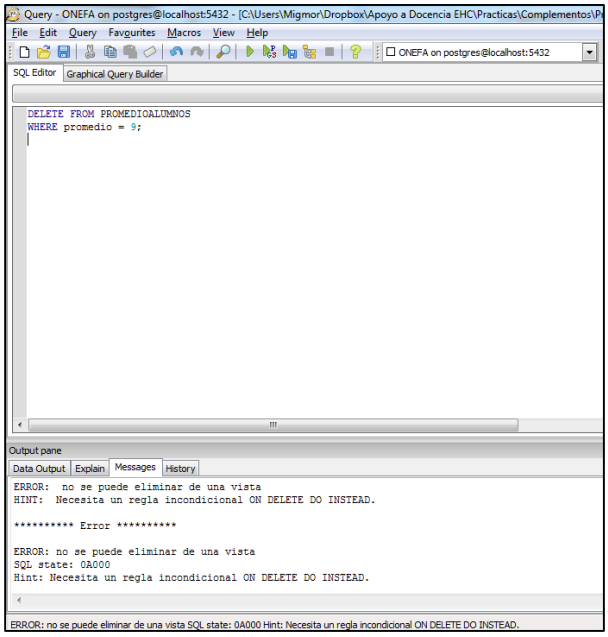

Figura 8.7 Error de Eliminación de tuplas PROMEDIOALUMNOS en versión 9.2 de PostgreSQL.

A partir de la versión 9.3 de PostgreSQL las vistas simples pueden ser actualizadas, permitiendo insertar, actualizar y borrar, esto solo si cumple las siguientes condiciones:

- La vista debe tener exactamente una entrada en su lista de FROM, que debe ser una tabla u otra vista actualizable.
- La definición de vista no debe contener COUNT, DISTINCT, GROUP BY, HAVING o LIMIT.
- La definición de la vista no debe contener operaciones de conjuntos (UNION, INTERSECT o EXCEPT) en el nivel superior.
- Todas las columnas de lista de selección de la vista deben ser simples referencias a columnas de la relación subyacente. No pueden ser expresiones, literales o funciones. Las columnas no pueden ser referenciadas.
- La columna de la relación subyacente no puede aparecer más de una vez en la lista de la selección de la vista.
- La vista no debe tener la prioridad *security_barrier*.

El siguiente ejemplo muestra la eliminación de tuplas con promedio igual a 9 directamente en la vista esto en la versión 9.3 de PostgreSQL, Figura 8.8.

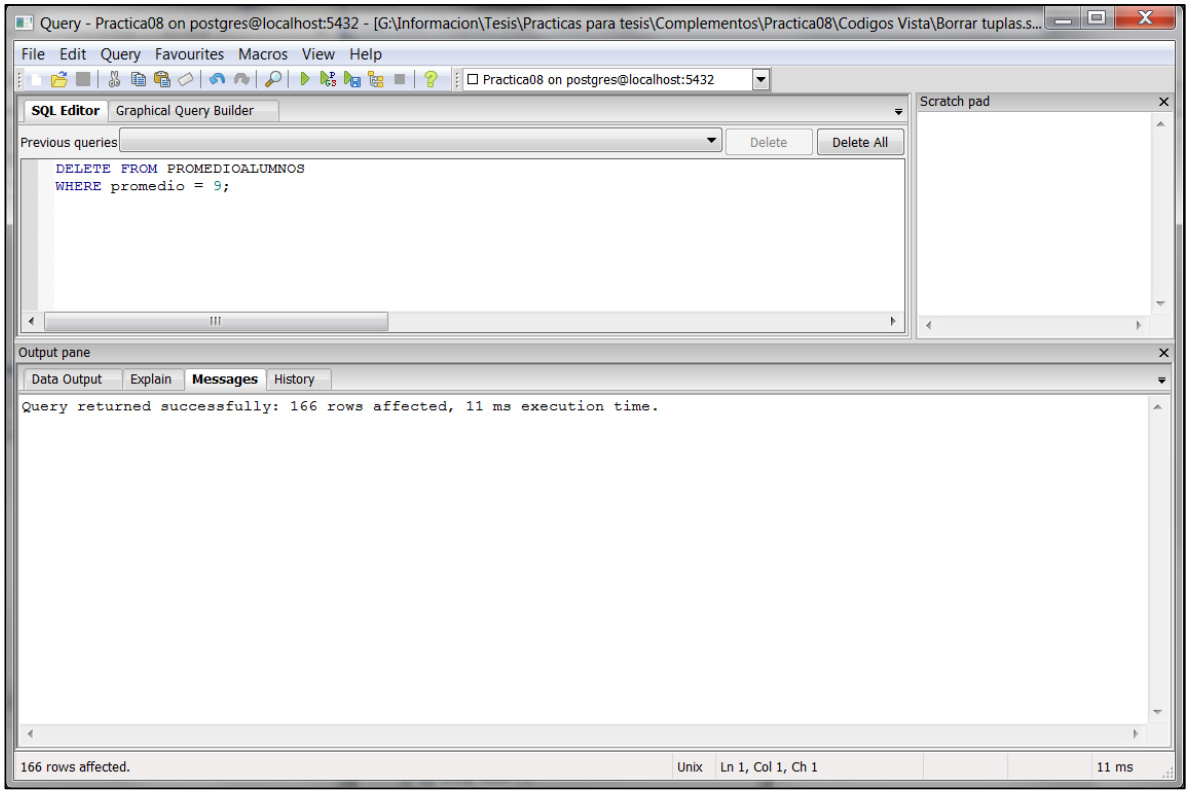

Figura 8.8 Eliminación de tuplas en vista PROMEDIOALUMNOS en versión 9.3 de PostgreSQL.

Para mayor información se puede consultar la documentación oficial de PostgreSQL en la siguiente liga:

http://www.postgresql.org/docs/9.3/static/sql-createview.html

4. Ejercicios

(NOTA: Resuelve los siguientes ejercicios en relación al proyecto que realizarás durante el curso, en dado caso que no tengas un proyecto, utiliza la información en el apéndice NFL-ONEFA parte 08 al final de esta práctica para realizarlos)

1. Realiza 3 consultas a diferentes tablas donde utilices diferentes operadores de comparación.
2. Realiza 3 consultas a diferentes tablas donde utilices funciones de agregación y agrupación.
3. Realiza 3 consultas que conecten diferentes tablas utilizando diferentes tipos de JOIN.
4. Realiza una consulta que utilice funciones de agregación, operadores de comparación y JOINs entre tablas.

Entregables requeridos para prácticas subsecuentes:

- Comprensión adecuada de funciones de agregación, operadores de comparación y reuniones entre tablas

5. Apéndice NFL-ONEFA parte 08

En la base de datos que se creó en el Apéndice NFL-ONEFA parte 07 inserta los siguientes registros y después realiza los ejercicios del punto 4 de ésta práctica.

NOTA: por el tamaño de los archivos no se colocarán dentro de la práctica, pero pueden ser consultados en los archivos respectivos dentro de la carpeta Anexos/NFL-ONEFA/LotesNFL y /LotesONEFA de este Manual de Prácticas.