CBTis#42

DESARROLLA SOFTWARE DE APLICACIÓN WEB CON ALMACENAMIENTO PERSISTENTE DE DATOS.

Actividad 3-Unidad 2

Programación

MÓDULO IV:
DESARROLLA SOFTWARE DE APLICACIÓN WEB CON ALMACENAMIENTO PERSISTENTE DE DATOS.

SUBMÓDULO 1:
CONSTRUYE BASES DE DATOS PARA APLICACIONES WEB.

Elaborado por: Abraham Castañeda Quintero
Docente: Alejandra Serrano Luna
Semestre y grupo: 5° “B”

Índice.

Contenido

Introducción.

En esta la actividad No. 3 unidad 2 del modulo IV de la carrera de programación, Investigaremos sobre como se manipulan los datos del lenguaje SQL, es decir como hacerle cambios a nuestra base de datos, que puede ir desde vistas de tablas o datos específicos o alterar dichos datos agregando modificando o borrando cualquiera de los datos.

Desarrollo.

1. Qué es el lenguaje de manipulación de datos?

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.
El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

1.1. Operaciones básicas:

En general a las operaciones básicas de manipulación de datos que podemos realizar con SQL se les denomina operaciones CRUD (de Create, Read, Update and Delete, o sea, Crear, Leer, Actualizar y Borrar, sería CLAB en español, pero no se usa). Lo verás utilizado de esta manera en muchos sitios, así que apréndete ese acrónimo.

1.1.1 ¿Para qué sirve el comando Select?

La sentencia SELECT nos permite consultar los datos almacenados en una tabla de la base de datos.

```
SELECT IF(500<1000, "YES", "NO");



SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN "Over 30"
    WHEN Quantity = 30 THEN "Equal 30"
    ELSE "Under 30"
END AS QuantityText
```

1.1.2 ¿Para qué sirve el comando Insert?

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

```
INSERT INTO 'tablatura' ('columna1',['columna2,... '])
VALUES ('valor1', ['valor2,...'])

O tambien se puede utilizar como:
INSERT INTO tablatura VALUES ('valor1','valor2')
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia `INSERT` deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

1.1.3 ¿Para qué sirve el comando Delete?

Una sentencia *DELETE* de SQL borra uno o más registros existentes en una tabla.

Forma básica

```
DELETE FROM tabla WHERE columna1 = 'valor1';
```

Ejemplo

```
DELETE FROM My_table WHERE field2 = 'N';
```

1.1.4 ¿Para qué sirve el comando Update?

Una sentencia *UPDATE* de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo

```
UPDATE My_table SET field1 = 'updated value asd' WHERE field2 = 'N';
```

1.2. ¿Qué se utilizan las funciones agregadas?

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Es decir, nos permiten obtener medias, máximos, etc... sobre un conjunto de valores.

1.2.1. ¿Para qué sirve la función AVG?

AVG: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

Otra definición:
Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta.

Avg(expr)

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

1.2.2. ¿Para qué sirve la función COUNT?

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

Count(expr)

En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total FROM Pedidos;
```

1.2.3. ¿Para qué sirve la función SUM?

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(expr)

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM DetallePedido;
```

1.2.4. ¿Para qué GROUP BY?

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

```
SELECT CAMPOS FROM TABLA WHERE CRITERIO GROUP BY CAMPOS DEL GRUPO
```

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no esta incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

```
SELECT ID_FAMILIA, SUM(STOCK)FROM PRODUCTOS GROUP BY ID_FAMILIA;
```

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
SELECT ID_FAMILIA SUM(STOCK) FROM PRODUCTOS GROUP BY ID_FAMILIA

HAVINGSUM(STOCK) > 100 AND NOMBREPRODUCTO LIKE BOS*;
```

1.3. ¿Qué se utilizan los joins en SQL?

Los JOINs en SQL sirven para combinar filas de dos o más tablas basándose en un campo común entre ellas, devolviendo por tanto datos de diferentes tablas. Un JOIN se produce cuando dos o más tablas se juntan en una sentencia SQL.

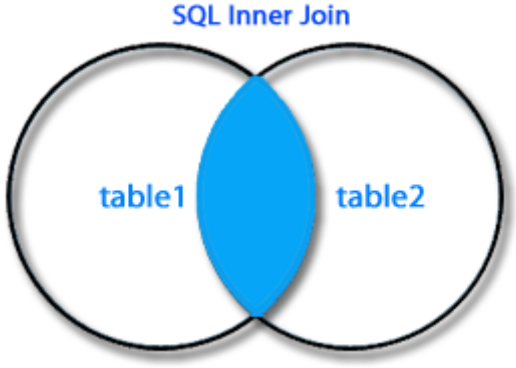
Existen más tipos de joins en SQL que los que aquí se explican, como CROSS JOIN, O SELF JOIN, pero no todos ellos están soportados por todos los sistemas de bases de datos.

1.3.1. ¿Para qué sirve Inner Join? (Añadir ejemplo de sintaxis)

INNER JOIN selecciona todas las filas de las dos columnas siempre y cuando haya una coincidencia entre las columnas en ambas tablas. Es el tipo de JOIN más común.

```
SELECT nombreColumna(s)
FROM tabla1
INNER JOIN tabla2
ON tabla1.nombreColumna=table2.nombreColumna;
```

Se ve más claro utilizando una **imagen**:



Vamos a verlo también con un ejemplo, mediante las tablas **Clientes** y **Pedidos**:

Clientes:

ClienteID	NombreCliente	Contacto
1	Marco Lambert	456443552
2	Lydia Roderic	445332221
3	Ebbe Therese	488982635
4	Sofie Mariona	412436773

Pedidos:

PedidoID	ClienteID	Factura
234	4	160
235	2	48
236	3	64
237	4	92

La siguiente sentencia SQL devolverá todos los clientes con pedidos:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID FROM Clientes
INNER JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID
ORDER BY Clientes.NombreCliente;
```

Si hay filas en Clientes que no tienen coincidencias en Pedidos, los Clientes no se mostrarán. La sentencia anterior mostrará el siguiente resultado:

NombreCliente	PedidoID
Ebbe Therese	236
Lydia Roderic	235
Sofie Mariona	234
Sofie Mariona	237

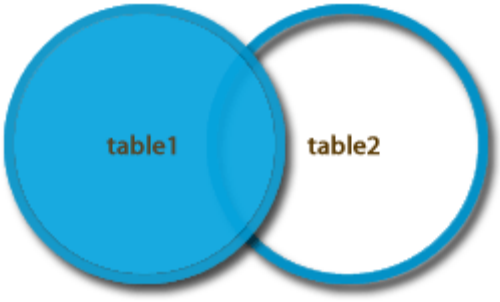
Sofie Mariona aparece dos veces ya que ha realizado dos pedidos. No aparece *Marco Lambert*, pues no ha realizado ningún pedido.

1.3.2. ¿Para qué sirve Left Join? (Añadir ejemplo de sintaxis)

LEFT JOIN mantiene todas las filas de la tabla izquierda (la tabla1). Las filas de la tabla derecha se mostrarán si hay una coincidencia con las de la izquierda. Si existen valores en la tabla izquierda pero no en la tabla derecha, ésta mostrará null.

```
SELECT nombreColumna(s)
FROM tabla1
LEFT JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La **representación de LEFT JOIN en una imagen** es:



Tomando de nuevo las tablas de Productos y Pedidos, ahora queremos mostrar todos los clientes, y cualquier pedido que pudieran haber encargado:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes LEFT JOIN Pedidos
ON Clientes.ClienteID=Pedidos.ClienteID
ORDER BY Clientes.NombreCliente;
```

La sentencia anterior devolverá lo siguiente:

NombreCliente	PedidoID
Ebbe Therese	236
Lydia Roderic	235
Marco Lambert	(null)
Sofie Mariona	234
Sofie Mariona	237

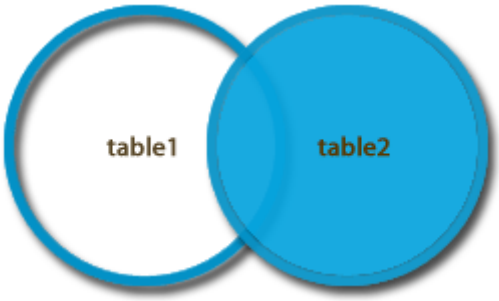
Ahora vemos que se muestran todas las filas de la tabla Clientes, que es la tabla de la izquierda, tantas veces como haya coincidencias con el lado derecho. Marco Lambert no ha realizado ningún pedido, por lo que se muestra null.

1.3.3. ¿Para qué sirve Rigth Join? (Añadir ejemplo de sintaxis)

Es igual que LEFT JOIN pero al revés. Ahora se mantienen todas las filas de la tabla derecha Las filas de la tabla izquierda se mostrarán si hay una coincidencia con las de la derecha. Si existen valores en la tabla derecha pero no en la tabla izquierda, ésta se mostrará null.

```
SELECT nombreColumna(s)
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La **imagen que representa a RIGHT JOIN** es:



De nuevo tomamos el ejemplo de **Clientes** y **Pedidos**, y vamos a hacer el mismo ejemplo anterior, pero cambiado LEFT por RIGHT:

```
SELECT Pedidos.PedidoID, Clientes.NombreCliente
FROM Clientes RIGHT JOIN Pedidos
ON Clientes.ClienteID=Pedidos.ClienteID
ORDER BY Pedidos.PedidoID;
```

Ahora van a aparecer todos los pedidos, y los nombres de los clientes que han realizado un pedido. Nótese que también se ha cambiado el orden, y se han ordenado los datos por PedidoID.

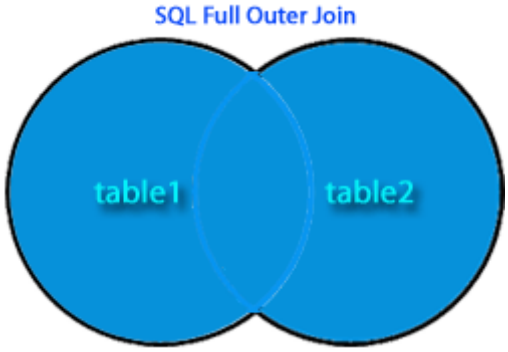
PedidoID	NombreCliente
234	Sofie Mariona
235	Lydia Roderic
236	Ebbe Therese
237	Sofie Mariona

1.3.4. ¿Para qué sirve Outer Join? (Añadir ejemplo de sintaxis)

OUTER JOIN o FULL OUTER JOIN devuelve todas las filas de la tabla izquierda (tabla1) y de la tabla derecha (tabla2). Combina el resultado de los joins LEFT y RIGHT. Aparecerá null en cada una de las tablas alternativamente cuando no haya una coincidencia.

```
SELECT nombreColumna(s)
FROM tabla1
OUTER JOIN tabla2
ON tabla1.nombreColumna=tabla2.nombreColumna;
```

La **imagen que representa el OUTER JOIN** es la siguiente:



Vamos a obtener **todas las filas** de las tablas **Clientes** y **Pedidos**:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes OUTER JOIN Pedidos
ON Clientes.ClienteID=Pedidos.ClienteID
ORDER BY Clientes.NombreCliente;
```

La sentencia devolverá todos los Clientes y todos los Pedidos, si un cliente no tiene pedidos mostrará null en PedidoID, y si un pedido no tuviera un cliente mostraría null en NombreCliente (en este ejemplo no sería lógico que un Pedido no tuviera un cliente).

La sintaxis de OUTER JOIN o FULL OUTER JOIN no existen en MySQL, pero se puede conseguir el mismo resultado de diferentes formas, esta es una:

```
SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
LEFT JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID

UNION

SELECT Clientes.NombreCliente, Pedidos.PedidoID
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID;
```

Conclusión.

En conclusión, en esta actividad aprendí sobre el funcionamiento del lenguaje SQL y otros comandos extras, que son parte de su sintaxis, en esta actividad concretamente hablaba sobre la manipulación de datos en una base de datos, esto es de suma importancia porque de nada sirve tener la estructura de una base de datos si no vamos a poder manipular los datos que tengamos, con todos los comandos anteriaries podemos consultar, insertar, borrar y modificar dichos datos mas aparte de algunos operadores de condiciones.

Referencias.

Agrup. de Regs y funciones agregadas (GROUP BY, AVG, COUNT, MAX, MIN). (s. f.). Agrupamiento de Registros y funciones agregadas. Recuperado 12 de noviembre de 2020, de http://basededatos.umh.es/sql/sql04.htm

C. (2014, 21 julio). *Fundamentos de SQL: Agrupaciones y funciones de agregación*. campusMVP.es. https://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Agrupaciones-y-funciones-de-agregacion.aspx

Diego Lazaro. (s. f.). *Principales tipos de JOINS en SQL*. Diego.com. Recuperado 12 de noviembre de 2020, de https://diego.com.es/principales-tipos-de-joins-en-sql

MySQL. (s. f.). *Lenguaje de manipulación de datos DML(Data Manipulation Language) - SQL Is My Sin*. ¿Que es SQL? Recuperado 12 de noviembre de 2020, de https://sites.google.com/site/sqlismysin/home/lenguaje-de-manipulacion-de-datos-dml-data-manipulation-language