Universidad Nacional Autónoma De México
Facultad De Ciencias, 2025-I
Fundamentos De Bases De Datos

PRÁCTICA 09:

DML. Consultas.

Profesor:
Gerardo Avilés Rosas

Ayudantes de teoría:
Luis Enrique García Gómez
Kevin Jair Torres Valencia

Ayudantes de laboratorio:
Ricardo Badillo Macías
Rocío Aylin Huerta González

DML

Operadores

Elementos básicos para el procesamiento de datos. Son **usados en expresiones**, **toman uno o dos argumentos y regresan un valor**. *PostgreSQL* soporta operadores para todos los tipos de datos.

Lista de Operadores

+ - * / < > = ~ ! @ # % ^ & | ` ?

Operadores como multiplicación, división, tienen mayor precedencia que otros, los operadores lógicos o de comparación tienen la menor precedencia. Operadores con la misma precedencia son ejecutados de izquierda a derecha. Mas información en: https://www.postgresql.org/docs/current/sql-syntax-lexical.html

Caracteres Especiales

El lenguaje *SQL* incluye los siguientes caracteres especiales:

- **Parentesis** () Controlan la precedencia de los operaciones en un grupo de expresiones. También identifican tuplas o atributos en un tipo compuesto, o parámetros de una función.
- **Corchete** [] Elementos de un arreglo.
- **Dos puntos :** Para acceder a elemento de arreglo.
- **Doble dos puntos** :: Para *castear* tipos.
- **Coma ,** Separar elementos de una lista.
- **Punto .** Para separar nombre de *schemas*, tablas y columnas una de otra.
- **Punto y coma ;** Terminar una instrucción.
- **Asterisco** * Referirse a todos los campos de una tabla.

Lenguaje de Manipulación de Datos

El *Lenguaje de Manipulación de Datos (Data Manipulation Language, DML)*, es un idioma proporcionado por los *Sistemas Manejadores de Bases de Datos (SMBD)* que **permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en la Bases de Datos**. El lenguaje de manipulación de datos más popular hoy en dia es *SQL*, usado para recuperar y manipular datos en una base de datos relacional.

Elementos del lenguaje de manipulación de datos:

- `INSERT`: Es una sentencia de insersión (`INSERT`) de *SQL* la cual se encarga de agregar uno o más registros a una (y sólo una) tabla en una base de datos relacional.

```
INSERT INTO nombreTabla(columna1, columna2, ...)
VALUES (valor1, valor2, ...);
```

- `DELETE`: Es una sentencia de eliminación de borrado (`DELETE`) de *SQL* la cual es la encargada de borra uno o más registros existentes en una tabla.

```
DELETE FROM nombreTabla
WHERE condición;

DELETE FROM nombreTabla;
```

- **UPDATE:** Es una sentencia de actualización o modificación (`UPDATE`) de *SQL*, es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

```
UPDATE nombreTabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condición;
```

SELECT

`SELECT` es la instrucción de `SQL` que devuelve un conjunto de resultados de registros de una o más tablas.

El *SMBD* se encarga de traducir la consulta en un "plan de consulta"que puede variar entre ejecuciones, versiones de base de datos y *software* de base de datos. Esta funcionalidad se denomina .ºptimizador de consultas", ya que es responsable de encontrar el mejor plan de ejecución posible para la consulta, dentro de las restricciones aplicables.

La instrucción `SELECT` tiene muchas cláusulas opcionales:

- **WHERE:** Especifica una condición al recuperar un conjunto de datos de una tabla o conjunto de tablas.

```
SELECT columna1, columna2, ...
FROM nombreTabla
WHERE condición;
```

- **ORDER BY:** Establece el orden de las filas de resultado en función de las columnas que se indiquen.

```
SELECT columna1, columna2, ...
FROM nombreTabla
ORDER BY columna1, columna2 ASC, columna3 DESC, ...;
```

- **AS:** Proporciona un alias que se puede usar para cambiar el nombre temporalmente de tablas o columnas.

```
SELECT columna1 AS algunNombre1, columna2 AS algunNombre2, ...
FROM nombreTabla;

SELECT columna1, columna2, ...
FROM nombreTabla AS algunNombre;
```

Combinación

La cláusula **JOIN** se utiliza para combinar registros de dos o más tablas de una base de datos. **JOIN** es un medio para combinar campos de dos tablas mediante el uso de valores comunes a cada una.

Tenemos varios tipos de JOIN:

- **CROSS JOIN**: Para cada combinación de las filas de T1 y T2, la tabla derivada va a contener a todas las filas que consta de todas las columnas de T1 seguidas de todas las columnas de T2. Si las tablas tienen N y M filas respectivamente, la tabla unida tendra N*M filas.
- **INNER JOIN**: Para cada fila R1 de T1, la tabla resultante, tiene una fila para cada fila en T2 que satisface la condición de Unión con R1.
- **LEFT OUTER JOIN**: Primero se hace un INNER JOIN, despues para cada fila en T1 que no satisfaga la condición de unión con ninguna fila en T2, se agrega una fila unida con valores nulos en las columnas de T2. Por lo tanto, la tabla unida tiene incondicionalmente al menos una fila por cada fila en T1.
- **RIGHT OUTER JOIN**: Primero se hace un INNER JOIN, despues para cada fila en T2 que no satisfaga la condición de unión con ninguna fila en T1, se agrega una fila unida con valores nulos en las columnas de T2. Por lo tanto, la tabla unida tiene incondicionalmente al menos una fila por cada fila en T2. Esto es lo contrario a un LEFT JOIN,
- **FULL OUTER JOIN**: Primero se hace un INNER JOIN. Entonces, por cada fila en T1 que no satisfaga con la condición del JOIN con alguna de las filas de T2, se agrega como una fila con valores null con las columnas de T2. Ademas, por cada una de las filas de T2 que no satisfagan con la condición del join con alguna de las filas de T1, se agrega como una fila con valores null con las columnas de T1.

Formas de definirlas:

```
T1 CROSS JOIN T2;

T1 {[INNER]|{LEFT | RIGHT| FULL}[OUTER]} JOIN T2 ON expresion_booleana;

T1 {[INNER]|{LEFT | RIGHT| FULL}[OUTER]} JOIN T2 USING (lista de columnas del join);

T1 NATURAL {[INNER]|{LEFT | RIGHT| FULL}[OUTER]} JOIN T2;
```

Donde:

- Las palabras **INNER** y **OUTER** son opcionales. Si no se especifica **INNER** es que se utiliza por defecto. **LEFT**, **RIGHT** y **FULL** implica utilizar **OUTER JOIN**.
- Las condiciones del **JOIN** son especificada con **ON o USING**, o de manera implicita con la palabra **NATURA**L. La condición del **JOIN** determina que filas de las dos tablas, son considerada para hacer "match".
- **ON** Es la condición mas general. Toma una expresión booleana. Y si un par de filas de T1 y T2 coinciden con la expresión **ON**, se evalua como **TRUE** para ellas.
- **USING** es una notación abreviada, toma una lista de nombres de columnas separados por comas, que las tablas que se van a unir, deben tener en común, y forma una condición de unión que especifica la igualdad de cada uno de estos pares de columnas. La principal diferencia con ON, es que aparecera doblemente las columnas que se comparara, por mientras **USING** solo aparecera una vez.
- **NATURAL** es una forma abreviada de **USING**, las cuales forman un **USING** que consta exactamente con los nombres de las columnas que aparecen en ambas tablas de entrada.

Operaciones de Conjuntos

Estas consultas utilizan al menos dos **SELECT** cuyos, resultados se pueden combinar para formar una única consulta. Se basan en los operadores matemáticos de conjuntos (unión, intersección y diferencia).

Lo más importante a tener en cuenta es que en estas operaciones: el número de columnas, el tipo y el orden de dichas columnas deben ser el mismo en todas las consultas que se combinan.

```
SELECT valor1, valor2, ..., valorn FROM nombreTabla
WHERE condicion
[UNION|INTERSECT|EXCEPT]
SELECT valor1, valor2, ..., valorn FROM
nombreTabla
WHERE condicion;
```

Donde:

- **UNION** permite añadir el resultado de un **SELECT** a otro **SELECT**. Para ello ambas instrucciones tienen que utilizar el mismo número y tipo de columnas. Al utilizar **UNION** se elimina los elementos duplicados, si se desea mantener los duplicados se utiliza **UNION ALL**.
- **INTERSECT** permite unir dos consultas SELECT de modo que el resultado serán las filas que estén presentes en ambas consultas.
- **EXCEPT** se combina dos consultas SELECT de forma que aparecerán los registros del primer **SELECT** que no estén presentes en el segundo.

Funciones de Agregación

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos.

```
COUNT(nombreColumna | *)
SUM(nombreColumna)
AVG(nombreColumna)
MIN(nombreColumna)
MAX(nombreColumna)
```

Donde:

- **COUNT**: Devuelve el número total de filas seleccionadas por la consulta.
- **SUM**: Suma los valores de campo que especifiquemos, esta operación solo se puede utilizar en columnas numéricas.
- **AVG**: Devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **MIN**: Devuelve el valor mínimo del campo que especifiquemos.
- **MAX**: Devuelve el valor maximo del campo que especifiquemos.

```
SELECT columnna1, columna2, ...,
funcionAgregacion(columnaN)
FROM nombreTabla
GROUP BY columna1, columna2 ,...
HAVING condicionSobreFuncionAgregacion;
```

Donde:

- **GROUP BY** divide las filas por los valores de las columnas, se utiliza mucho con las funciones de agregación ya que calculara el valor, dependiendo de la columna.
- **HAVING** Filtra los grupos a través de una condición, es parecido al WHERE, solo que el WHERE no se puede utilizar con funciones de agregación.

Actividades.

- Se debe de realizar un script DML llamado Query.sql que contenga la solución a las siguientes consultas:
  - Entrenadores y Atletas, que compartan el apellido y que se encuentren participando en la misma disciplina. Deberan ordenar la información a partir del apellido paterno.
  - La información de eventos cuyo precio base sea mayor a 2500. Deberan ordenar la información a partir del precio.
  - Atletas que hayan participado en más de 1 disciplina.
  - Los Jueces y Entrenadores que tengan la misma nacionalidad pero que no se encuentren participando en el mismo evento.
  - Patrocinadores que solo esten patrocinando a una disciplina.
  - El número de medallas de oro ganadas por México.
  - El número de medallas de plata ganadas por Japon.
  - El número de medallas de bronce ganas por España.
  - La información de los atletas que ganaron medallas en la disciplina halterofilia.
  - La información de todos los atletas que hayan ganado alguna medalla. Asi como un conteo de las medallas de oro, plata y bronce que ganaron. La información debera ser ordenada con respecto a las medallas, es decir primero oro, despues plata y al final bronce.

  Cada consulta debera al menos regresar 5 registros si es que se puede, en el caso de que no, deberan explicar el porque.
- En el documento Práctica09.pdf, deberan agregar la solucion de cada consulta, ademas de una pequeña explicación de como lo solucionaron.

Figura 1: *Actividades.*

Entregables.

Deberán subir un archivo con formato *zip* a *Google Classroom*, de acuerdo a lo indicado en los lineamientos de entrega. Debe de estar organizado de la siguiente manera, (suponiendo que el nombre del equipo que está entregando es *Dream Team*).

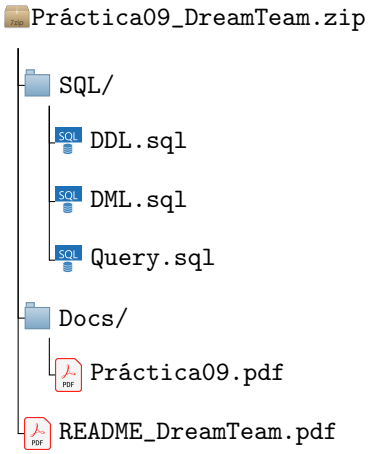

Figura 2: *Entregables.*

Nota.

Para cualquier duda o comentario que pudiera surgirles al hacer este trabajo, recuerden que cuentan con la asignación de este entregable en el grupo de *Classroom*, en donde seguramente encontrarás las respuestas que necesites.

Figura 3: *Nota.*