Universidad Autónoma de Querétaro

Facultad de Contabilidad y Administración

MANUAL / HANDBOOK DE SQL

Alumno: Bravo Salcedo David.

Catedrático: Edgar Ruiz

Temas Selectos de Estadística

Lic. En Actuaría

Grupo 17

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

Fecha de actualización: 21/Octubre/2024

Contenido

1. Introducción a Bases de Datos
   1.1. Conceptos básicos

2. Consulta con SQL:
   2.1. Operadores de Comparación
   2.2. Comandos
   2.3. Operadores lógicos
   2.4. Operadores IN y BETWEEN
   2.5. ORDER BY y DISTINCT
   2.6. Paginación

3. Agregaciones y agrupaciones:
   3.1. Funciones de agregación
   3.2. GROUP BY

4. Tablas
   4.1. Crear Tablas
   4.2. Insertar filas
   4.3. Recuperar datos
   4.4. Actualizar filas
   4.5. Eliminar filas
   4.6. Modificar tablas

5. Claves.

6. Bibliografía consultada:

1.- Introducción a Bases de Datos

Conceptos que giran en torno a:

- Datos
- Base de datos
- Sistema de Gestión de Bases de Datos (DBMS)
    - Ventajas
- Tipos de Bases de Datos
    - Base de Datos Relacional
    - Base de Datos No-Relacional

- **Data:**

Cualquier tipo de información almacenada es llamada Data.
Ejemplos:

1. Mensajes multimedia en WhatsApp
2. Productos y pedidos en Amazon
3. Información de contacto del directorio telefónico, etc.

- **Database:**

  Un conjunto organizado de datos es llamada Base de Datos.

- **Database Management System (DBMS):**

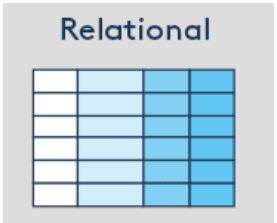
  Software que se utiliza para almacenar y accesar fácilmente a los datos de una Base de Datos de una manera segura.

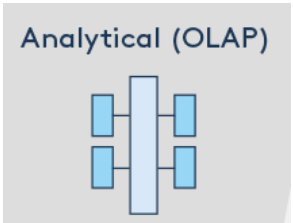
  Ventajas

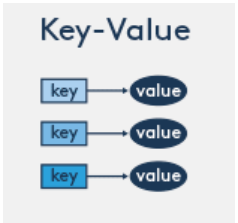
  - **Seguridad:** Los datos son almacenados de manera segura.
  - **Facilidad de uso:** Permite formas más sencillas de crear y actualizar datos al ritmo en que se generan y actualizan respectivamente.
  - **Durabilidad y disponibilidad:** Durable y permite acceso a todos los clientes en cualquier momento.
  - **Rendimiento:** Rápidamente accesible para todos los clientes (aplicaciones e interesados)

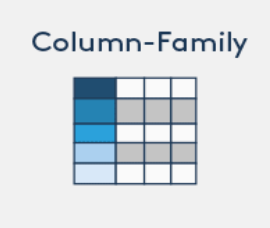
- **Types of Databases:**

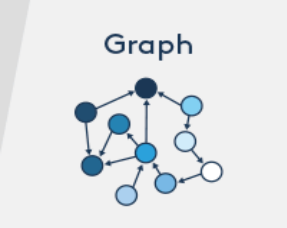
  Existen diferentes tipos de Bases de Datos de acuerdo en como se organizan los datos:

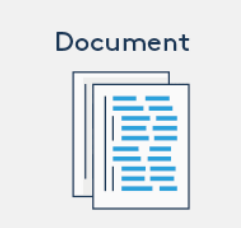












- **Base de Datos Relacional**
  Los datos están organizados en forma de tablas.

- **Base de Datos No-Relacional**
  Gráfico, Clave-Valor, Familia de Columnas, Documento.
  Estos cuatro tipos son comúnmente denominadas Bases de Datos No-Relacionales.

  Nota:
  La elección de la Base de Datos depende de nuestros requerimientos.
  La Base de Datos Relacional es la más usada comúnmente.

- **Relational DBMS**
  Un **DBMS Relacional** es un DBMS designado específicamente para bases de datos relacionales. Las bases de datos relacionales organizan los datos en forma de tablas.
  Por ejemplo: Oracle, PostgreSQL, MySQL, SQLite, SQLServer, IBM DB2

- **Non – Relational DBMS:**
  Es un **DBMS** diseñado específicamente para bases de datos no-relacionales. Las bases de datos no-relacionales almacena los datos en una forma no-tabular.
  Por ejemplo: Elasticsearch, CouchDB, DynamoDB, MongoDB, Cassandra, Redis, etc.

1.1.- (Conceptos básico)

- SQL significa Structured Query Lenguage (Lenguaje de Consulta Estructurada).
- SQL es usado para trabajar en DBMS
- SQL es declarativo, lo que significa que es fácil de aprender.
- SQL proporciona múltiples comandos para llevar a cabo diversas operaciones como crear, recuperar, actualizar y borrar los datos.

2.- Consultas con SQL

2.1.- Operadores de Comparación

Los operadores de comparación, o también llamados operadores relacionales o booleanos, comparan valores en una base de datos y determinan si son iguales, no iguales, mayores que, menores que, mayores o iguales a y menores o iguales a.

Operador	Descripción
=	Igual que
<>	Distinto de
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Por ejemplo:

Para mostrar los detalles de los productos de la compra del supermercado, que no pertenecen a la categoría de "**Food**", se realiza lo siguiente:

```
SELECT
FROM
        product
WHERE
        category <>"Food";
```

2.2.- Comandos

- **LIKE**
  Es usado para realizar consultas en cadenas.
  Este comando es usado especialmente en la cláusula **WHERE** para devolver todas las filas que coinciden con el patrón dado.
  Escribimos **patrones** usando los siguientes **caracteres comodín:**

Símbolo	Descripción
Signo de porcentaje %	Hace coincidir una cantidad arbitraria de caracteres (incluido el cero).
Guión bajo _	Hace coincidir cualquier carácter individual.

Patrones comunes:

Patrón	Ejemplo	Descripción
Coincidencia exacta	WHERE name LIKE “Morado”	Recupera productos cuyo nombre es exactamente “Morado”
Comienza con	WHERE name LIKE “Morado%”	Recupera productos cuyo nombre comience con “Morado”
Termina con	WHERE name LIKE “%Morado”	Recupera productos cuyo nombre termina con “Morado”
Contiene	WHERE name LIKE “%Morado%”	Recupera productos cuyo nombre contiene “Morado”
Coincidencia de patrones	WHERE name LIKE “m_%”	Recupera productos cuyo nombre comienza con “m” y tiene al menos 2 caracteres de longitud.

Nota: El signo de porcentaje (%) se utiliza cuando no se esta seguro de el numero de caracteres presentes en la cadena.
Si se conoce el numero total de la cadena, entonces el carácter comodín que se usará será el guión bajo (_)

2.3.- Operadores lógicos

Operador	Descripción
AND	Evalúa dos o más expresiones para obtener un resultado lógico.
OR	Evalúa al menos una de las condiciones dadas.
NOT	Solo tiene como entrada una expresión y se utiliza para negar una condición de la clausula WHERE.

- **Multiple Logical Operators**
  También podemos usar combinaciones de operadores lógicos para combinar dos o más condiciones. Estas permiten ajustar con precisión los requisitos de recuperación de datos.

  - **Procedence**
    Cuando una consulta contiene varios operadores, la procedencia de los operadores determinará la secuencia de operaciones.

NOT ↑ High
AND
OR ↓ Low

Orden de procedencia:

- NOT
- AND
- OR

2.4.- Operadores IN y BETWEEN

- **IN Operator**
  Determina si el valor de una expresión es igual a uno de varios valores en una lista especificada.

  Por ejemplo:

  Syntax:

```
SELECT
    column1, column2, …
FROM
    table_name
WHERE
    column_name

IN

     (value1_value2, …);
```

- **BETWEEN Operator**
  Usado para indicar que deseamos recuperar de los registros según el intervalo de valores de un campo.

  Por ejemplo:

  Syntax:

```
SELECT
    column_name(s)
FROM
    table_name
WHERE
    column_name

BETWEEN value1

AND

    value2;
```

  Nota:

  Al utilizar el operador BETWEEN, el primer valor debe de ser menor que el segundo, de lo contrario, se obtendrá un resultado incorrecto dependiendo del DBMS.

2.5.- ORDER BY y DISTINCT

- **ORDER BY**
  Usamos esta cláusula para ordenar los registros seleccionados de acuerdo con un orden específico. Por default, ORDER BY ordena los datos en orden ascendente.

  Por ejemplo:

  Syntax

```
SELECT
    column1, column2, …
FROM
    table_name
ORDER BY
    column1 asc/desc,
    column2 asc/desc;
```

- **DISTINCT**
  Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción **SELECT** se incluyan en la consulta deben ser únicos:

  Por ejemplo:

  Syntax:

```
SELECT DISTINCT
    column1,
    column2,
FROM
    table_name
```

2.6.- Paginación

Es una estrategia que se emplea al consultar cualquier conjunto de datos que contenga más de unos pocos cientos de registros. Gracias a esta, podemos dividir el gran conjunto de datos en fragmentos que podemos recuperar y mostrar gradualmente al usuario, lo que reduce la carga en la base de datos.

Usamos las cláusulas **LIMIT** y **OFFSET** para seleccionar un fragmento de resultados.

- **LIMIT**
  Es una clausula que es usada para especificar el numero de filas que nos gustaría obtener en el resultado.

Por ejemplo:

Syntax:

```
SELECT
        column1,
        column2,
        columnN…
FROM
        table_name
LIMIT n;
```

Nota: Si el valor limite es mayor que el numero total de filas, se mostrarán todas las filas.

- **OFFSET**

  Utilizada para omitir un número determinado de filas en el resultado de una consulta. A menudo se combina con la cláusula LIMIT para la paginación de datos.

  Por ejemplo:

  Syntax:

```
SELECT
        column1,
        column2,
        columnN
FROM
        table_name
LIMIT m
OFFSET n;
```

  Nota:
  No usar OFFSET antes de LIMIT.
  No usar solo OFFSET.

3.- Agregaciones y agrupaciones:

3.1.- Funciones de agregación

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

Funciones de Agregación	Descripción
COUNT	Contar el número de valores.
SUM	Añadir todos los valores.
MIN	Devuelve el valor mínimo-
MAX	Devuelve el valor máximo.
AVG	Calcula el promedio de los valores.

Por ejemplo:

Syntax

```
SELECT COUNT (column_name)
FROM table_name;
```

Nota: se pueden calcular distintas funciones agregadas en una solo consulta.

- **Difference between COUNT**
  - **COUNT(*)**
    Nos permite calcular el número de filas de la tabla.
  - **COUNT(column_name)**
    Esta función ayuda a contar el numero de valores no nulos en una columna especificada.

- **Alias**
  Si se usa la palabra clave **AS,** podemos proporcionar nombres temporales alternativos a las columnas en el resultado.

  Por ejemplo:

  Syntax:

```
SELECT
        c1 AS a1
        c2 AS a2,
FROM
        table_name
```

3.2.- GROUP BY: Esta cláusula se utiliza para agrupar filas en una tabla en función del valor de una columna determinada

Por ejemplo:

Syntax:

```
SELECT
        Column1,
        Column2,
        columnN
FROM
        Table_name
GROUP BY columnX
```

- **GROUP BY WITH WHERE**

  Se puede hacer uso del comando **WHERE** para filtrar los datos antes de realizar la agregación.

  Por ejemplo:

  Syntax:

```
SELECT
        C1,
        Aggregate_function(c2)
FROM
        Table_name
WHERE
        C3=v1
GROUP BY c1;
```

- **HAVING**

  Esta clausula se es utilizada para filtrar las filas resultantes después de aplicar la cláusula **GROUP BY**.

  Por ejemplo:

  Syntax

```
SELECT
        c1,
        c2,
        aggregate_fuction(c1)
```

```
FROM
        table_name
GROUP BY
        c1, c2
HAVING
        condition;
```

Nota:
WHERE es usado para filtrar filas y esta acción se realiza antes de agrupar.
HAVING es usado para filtrar grupos y esta acción se realiza después de agrupar

- **CONCEPTO DE NULL:**

Definición: En SQL, **NULL** representa un valor desconocido, ausente. No debemos confundirlo con el cero o una celda que tengamos vacía o cualquier otro valor. Es simplemente la ausencia de un dato.

  - Ausencia de valor:
  - Indica que un valor no está definido o no se conoce.
  - No es lo mismo que cero o una cadena vacía.

Funciones:

Evitar suposiciones: En lugar de asignar valores arbitrarios (como el cero o incluso poner algo com N/A), se usa **NULL** para mantener la integridad de los datos.

Facilitar consultas: Identificar y trabajar con registros incompletos en la base de datos.

4.- TABLAS

4.1.- Crear Tabla

Crear una nueva tabla en la base de datos.

Syntax:

```
CREATE TABLE table_name (
        columna1 type1,
        column2 type2,
);
```

Aquí, **type1** y **type2** en el **syntax** son los tipos de datos de **column1** y **column2** respectivamente. Los tipos de datos que se permiten en SQL se mencionan a continuación:

Por ejemplo:

```
CREATE TABLE player(
        name VARCHAR(200),
        age INTEGER,
        score INTEGER
);
```

Podemos revisar los detalles de la tabla creada en cualquier momento usando el comando “PRAGMA”

- **Data Types:**
  Tipos de datos usados frecuentemente en SQL:

Datatype	Syntax
Integer	INT/INTEGER
Float	FLOAT
String	VARCHAR
Text	TEXT
Date	DATE
Time	TIME
Datetime	DATETIME
Boolean	BOOLEAN

Nota:

- Los valores Boolean se guardan como números enteros 0 (FALSE) y 1 (TRUE)
- Fecha es representado como: “YYYY-MM-DD”
- Fecha y hora son representados como: “YYYY-MM-DD HH:MI:SS”

- **PRAGMA**
  El comando **PRAGMA_TABLE_INFD** devuelve la información sobre una tabla especifica en una base de datos.

  Syntax:

  PRAGMA TABLE INFD(table_name);

  Por ejemplo:

  PRAGMA TABLE INFD(desserts);

  Nota: Si el nombre de la tabla no existe, no se dará ningún resultado.

4.2.- Insertar filas

- **INSERT**

Comando que sirve para insertar una nueva fila en una tabla.

Syntax:

```
INSERT INTO
        Table_name(columna, column2, columnN)
VALUES
        (value1, value2, valueN),
        (value1, value2, valueN),
);
```

Cualquier numero de filas del 1 a N se puede insertar en una tabla especifica utilizando la sintaxis anterior.

4.3.- Recuperar datos

El comando **SELECT** es usada para recuperar filas de una tabla.

- **Selecting Sepecific Columns**
  Para recuperar datos meramente de columnas específicas de una tabla, se deberá agregar los respectivos nombres de las columnas en el comado

SELECT.

Syntax:

```
SELECT
    column1,
    column2,
    column N
FROM
    table_name:
```

Por ejemplo:

```
SELECT
    name,
    age
FROM
    player
```

- **Selecting All Columns**
  A veces es posible que queramos seleccionar todas las columnas de una tabla. Debemos hacer lo siguiente:

  Syntax:

  ```
  SELECT*
  FROM table_name;
  ```

  Por ejemplo:

  ```
  SELECT*
  FROM player;
  ```

- **UNIONES (Joins)**

Los joins combinan datos de dos o más tablas basándose en una relación común. Aquí están los principales:

INNER JOIN: Devuelve los registros que tienen coincidencias en ambas tablas.

Ejemplo: **SELECT * FROM empleados INNER JOIN departamentos ON empleados.id_dep = departamentos.id;**

LEFT JOIN: Muestra todos los registros de la tabla izquierda y las coincidencias de la derecha (si existen).

Ejemplo**: SELECT * FROM empleados LEFT JOIN proyectos ON empleados.id = proyectos.id_emp;**

FULL JOIN: Combina todos los registros, coincidan o no.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

```
SELECT Nombre_Categoría, NombreProducto FROM Categorias INNER JOIN
Productos
ON Categorias.IDCategoria = Productos.IDCategoria;
```

En el ejemplo anterior, IDCategoria es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción SELECT. Para incluir el campo combinado, incluir el nombre del campo en la instrucción SELECT, en este caso, Categorias.IDCategoria.

También se pueden enlazar varias cláusulas ON en una instrucción JOIN, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1 INNER JOIN tabla2
ON tb1.campo1 comp tb2.campo1 AND ON tb1.campo2 comp tb2.campo2)
OR ON tb1.campo3 comp tb2.campo3)];
```

También puede anidar instrucciones JOIN utilizando la siguiente sintaxis:

```
SELECT campos
FROM tb1 INNER JOIN (tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)] ON tb2.campo2 comp tb3.campo3) ON
tb1.campo1 comp tb2.campo2;
```

Un LEFT JOIN o un RIGHT JOIN puede anidarse dentro de un INNER JOIN, pero un

INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.

Por ejemplo:

```
SELECT DISTINCTROW Sum([Precio unidad] * [Cantidad]) AS [Ventas],
[Nombre] & " " & [Apellidos] AS [Nombre completo] FROM [Detalles de
pedidos],
Pedidos, Empleados, Pedidos INNER JOIN [Detalles de pedidos] ON Pedidos.
[ID de pedido] = [Detalles de pedidos].[ID de pedido], Empleados INNER JOIN
Pedidos ON Empleados.[ID de empleado] = Pedidos.[ID de empleado]
GROUP BY
[Nombre] & " " & [Apellidos];
```

Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales.

Si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que

contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: LEFT OUTER JOIN y RIGHT OUTER JOIN. En la práctica funciona correctamente de una u otra forma.

No obstante, los INNER JOIN ORACLE no es capaz de interpretarlos, pero existe una sintaxis en formato ANSI para los INNER JOIN que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

```
SELECT Facturas.*, Albaranes.*
FROM Facturas INNER JOIN Albaranes ON Facturas.IdAlbaran = Albaranes.IdAlbaran
WHERE Facturas.IdCliente = 325
```

La transformación de esta sentencia a formato ANSI sería la siguiente:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran AND Facturas.IdCliente = 325
```

Como se puede observar los cambios realizados han sido los siguientes:

Todas las tablas que intervienen en la consulta se especifican en la cláusula FROM.

Las condiciones que vinculan a las tablas se especifican en la cláusula WHERE y se vinculan mediante el operador lógico AND.

SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran (+) AND Facturas.IdCliente = 325

Y esto a un RIGHT JOIN:

SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes WHERE Facturas.IdAlbaran (+) = Albaranes.IdAlbaran AND Facturas.IdCliente = 325

En SQL-SERVER se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres (+)

sino los caracteres =* para el LEFT JOIN y *= para el RIGHT JOIN.

- **Selecting Specific Rows**
  Usamos la cláusula **WHERE** solo para recuperar filas específicas.

  Syntax:

  ```
  SELECT
  FROM table_name
  WHERE condiction;
  ```

  WHERE especifica una condición que debe cumplirse para recuperar los datos de una base de datos.

  Por ejemplo:

  SELECT

  FROM player

  WHERE name=”David”

2.4.- Actualizar filas

UPDATE es el comando utilizado para actualizar los datos de una tabla existente. Podemos actualizar todas las filas o solo columnas específicas.

- **Update all Rows**

  Actualizar todas las filas:

  Syntax:

```
UPDATE
    Table_name
SET
    Column1=value1
```

  Por ejemplo:

```
UPDATE
    Player
SET
    Score=100;
```

- **Update Specific Rows**

  Actualizar filas especificas:

  Syntax:

```
UPDATE
    Table_name
SET
    Column1=value1
WHERE
    Column1=value2;
```

  Por ejemplo:

```
UPDATE
    Player
SET
    Score=150
WHERE
    Name=”Ram”;
```

4.5.- Eliminar filas

DELETE es el comando que se utiliza para eliminar datos existentes de una tabla.

- **Delete All Rows**

  Para eliminar todas las filas:

  Syntax:

  ```
  DELETE FROM
      table_name
  ```

  Por ejemplo:

  ```
  DELETE FROM
      player;
  ```

- **Delete Specific Rows**

  Eliminar filas especificas:

  Syntax:

  ```
  DELETE FROM
      table_name
  WHERE
      column1=value1
  ```

  Por ejemplo:

  ```
  DELETE FROM
      player
  WHERE
      name=”David”;
  ```

  Nota: no se pueden recuperar los datos una vez que estos se hayan eliminado de la tabla.

- **Drop Table**

  DROP es el comando que es usado para eliminar una tabla de la base de datos.

  Syntax:

  ```
  DROP TABLE
      Table_name
  ```

  Por ejemplo:

  ```
  DROP TABLE
      player;
  ```

4.6.- Modificar tabla

El comando **ALTER** es usado para agregar, eliminar o modificar columnas en una tabla existente.

- **Add Column**

  Añadir columna:

  Syntax:

```
ALTER TABLE
        table_name
ADD
        column_name datatype;
```

  Por ejemplo:

```
ALTER TABLE
        player
ADD
        jersey_num INT;
```

  Nota: Los valores predeterminados para las columnas recién agregadas en las filas existentes será **NULL.**

- **Rename Column**

  Para renombrar una columna se utiliza el comando **RENAME COLUMN**

  Syntax:

```
ALTER TABLE
        table_name RENAME COLUMN c1 TO c2;
```

  Por ejemplo:

```
ALTER TABLE

        player RENAME COLUMN jersey_num TO jersey_number;
```

- **Drop Column**

  Se usa **DROP COLUMN** para eliminar una columna:

  Syntax:

```
ALTER TABLE
        table_name DROP COLUMN columna_name;
```

Por ejemplo:

```
ALTER TABLE
    Player DROP COLUMN jersey_number;
```

Nota: **DROP COLUMN** no es compatible con algunos DBMS, incluido SQLite.

5.- CLAVES (KEYS)

- **CLAVE PRIMARIA (PRIMARY KEY):**

Es una columna o un conjunto de columnas en una tabla que se utiliza para identificar de manera única cada fila de esa tabla.

Características:

- Los valores en la clave primaria deben ser únicos para cada fila.
- No puede contener valores nulos.
- Cada tabla puede tener solo una clave primaria.
- Se usa para garantizar la unicidad de los registros dentro de la tabla.

- **CLAVE FORÁNEA (FOREIGN KEY):**

Definición: Es una columna o un conjunto de columnas que establece una relación entre dos tablas, haciendo referencia a la clave primaria de otra tabla.

Características:

- Los valores en la clave foránea deben coincidir con los valores en la clave primaria de la tabla referenciada.
- Puede contener valores nulos, salvo que se defina con restricciones adicionales (NOT NULL).
- Se utiliza para garantizar la integridad referencial entre tablas.

Bibliografía Consultada:

- https://learnsql.es/blog/el-operador-sql-between/
- https://snmb-admin.readthedocs.io/en/latest/geotalleres/conceptos-sql/conceptos_sql.html
- https://learnsql.es/blog/cual-es-la-funcion-de-distinct-en-sql/
- https://appwrite.io/blog/post/best-pagination-technique
- https://www.datacamp.com/es/tutorial/sql-offset