# BASE DE DATOS ACTIVIDAD A DESARROLLAR

## PARTE 1 - INVESTIGACIÓN

Los estudiantes deben investigar y explicar con sus propias palabras:

**1. Significado de la sentencia**

CREATE DATABASE llantas_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

Deben interpretar:

- Qué es **CHARACTER SET**

  Un carácter es la regla que define las características como las letras, números y símbolos y se puede almacenar y representar de un formato binario para su almacenamiento.

- Qué es **COLLATE**

  Collate define las reglas para comparar y ordenar caracteres, ósea como maneja las diferencias entre mayúscula, minúsculas, acentos y otros caracteres.

- Por qué utf8mb4 permite emojis y caracteres universales

  Utf8mb4 permite utilizar los emojis y caracteres, a diferencia de la versión utf8 de MYSQL que solo usa hasta 3 bytes por carácter, utf8mb4 permite hasta 4 bytes por carácter.

- Qué significa "**case-insensitive**"

  Case-insensitive significa que las comparaciones de texto no distinguen entre letras mayúsculas y minúsculas. Es decir para una búsqueda este mayúscula y minúscula es exactamente lo mismo.

- Cómo afectan estas configuraciones a consultas como:
  - LIKE

    Una consulta con like puede verse afectada porque la comparación de texto no funciona de manera uniforme. Esto puede hacer que no coincidan palabras por diferencias de mayúsculas y minúsculas, por acentos, por caracteres especiales o incluso interpreta los símbolos de otra forma. Puede devolver datos incorrectos.

  - ordenamiento ORDER BY

    En el ordenamiento puede quedar incorrecto o no coincidir con el idioma. Ejemplo las letras con acentos pueden ordenarse aparte de las sin acento.

  - agrupamiento GROUP BY

    El agrupamiento también cambia según la colación Llanta y llanta se agrupan en grupos distintos, palabras con y sin acento pueden tratarse como diferentes.

**2. ENGINE = InnoDB**

Investigar:

- Qué es InnoDB

InnoDB es un motor de almacenamiento diseñar para ofrecer un equilibrio entre alto rendimiento y alta fiabilidad. Como se gestiona las transacciones, índices, bloques, etc.

- Por qué se usa

Se usa porque es un motor seguro, moderno y confiable, con funciones que otros motores como MyISAM no ofrecen. MyISAM es más antiguo.

Ofrece integridad de los datos no perder información, aunque haya fallos.

Transacciones permite agrupar varias operaciones como una sola acción que se confirma o deshace completamente.

Llaves foranes mantienen la consistencia entre tablas relacionadas.

- En qué parte se coloca (en la definición de tablas)

Se coloca al final de la definición de la tabla

```
CREATE TABLE usuarios (
    id INT PRIMARY KEY,
    nombre VARCHAR(100)
) ENGINE = InnoDB;
```

La palabra clave es ENGINE = InnoDB indica que esta tabla usara InnoDB. Si no se pone nada, MySQL puede usar el motor predeterminado que usualmente es InnoDB en versiones modernas.

- Si es obligatorio o puede omitirse

No, no es obligatorio, puedes usar otros motores, pero si quieres transacciones, integridad referencial o cumplimiento ACID necesitas un InnoDB

- Ventajas de InnoDB (transacciones, FK, ACID)

Transacciones -> Permite agrupar varias operaciones y confirmarlas o revertirlas todas juntas.

Llaves Foráneas -> Garantiza consistencia entre tablas relacionadas, evitados errores de datos.

Cumplimiento ACID -> Atomicidad, Consistencia, Aislamiento y Durabilidad.

Bloqueo a nivel de fila -> Mejora la concurrencia de usuarios en tablas con muchas escrituras.

- Un ejemplo simple indicando su utilidad

```
CREATE TABLE Clientes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100) UNIQUE
) ENGINE=InnoDB;

CREATE TABLE Pedidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT,
    total DECIMAL(10,2),
    FOREIGN KEY (cliente_id) REFERENCES Clientes(id)
) ENGINE=InnoDB;
```

Aquí se intenta crear un pedido cliente_id = 99 y ese cliente no existe InnoDB rechaza la operación gracias a la FK.

Si estas procesando un pago y ocurre un erro, puedes revertir la transacción para que no se cobre ni se registre un pedido incompleto

**3. Datos Huérfanos**

Investigar:

- Qué significa un dato huérfano

  Un dato huérfano es un registro en una tabla secundaria que hace referencia a un registro que ya no existe en la tabla principal.

- En qué situación ocurre (cuando se elimina un registro padre con hijos dependientes)

  Ocurre principalmente cuando se elimina un registro padre que tiene registros hijos dependientes, sin actualizar o borrar sus hijos.

- Por qué es un problema en bases de datos relacionales

  Pueden representar problemas de escalabilidad ya que su rendimiento se degrada con grandes volúmenes de datos y se vuelven inflexibles ante camios debido a su esquema rígido.

  Error de consulta de datos, si intentan unir tabla con join para mostrar pedidos con sus clientes, los huérfanos pueden no tener correspondencia, generando resultados incompletos o incorrectos,

- Cómo se evita

  Utilizar llaves foráneas con restricciones, definir relaciones entre tablas usando llaves foranes foreign key garantica que los registros hijos siempre tengan un padre valido.

Acciones al eliminar padre, on delete cascade si eliminas el registro padre, los hijos también se elimina automáticamente.

**4. Integridad Referencial**

Investigar:

- Qué es

La integridad referencial garantiza que las relaciones entre tablas se mantengan coherentes.

- Se aplica cuando tienes clave primaria (PK) en una tabla y clave foránea (FK) en otra tabla.
- Ejemplo: Una tabla Clientes y otra tabla Pedidos. Cada pedido debe tener un cliente válido.
- Por qué es importante

  Evita datos huérfanos: pedidos que no pertenecen a ningún cliente.

  Mantiene la consistencia de la base de datos.

  Permite controlar qué sucede cuando se elimina o actualiza un registro relacionado.

- Acciones referenciales:

| Acción | Qué hace |
|---|---|
| **CASCADE** | Elimina o actualiza automáticamente los registros relacionados. |
| **RESTRICT** | No permite eliminar o actualizar si hay registros relacionados. |
| **SET NULL** | Pone NULL en la FK de la tabla hija si el registro de la tabla padre se elimina o actualiza. |
| **SET DEFAULT** | Pone un valor por defecto en la FK si el registro padre se elimina o actualiza. |
| **NO ACTION** | No hace nada, pero impide cambios si violan la integridad (similar a RESTRICT). |

**Deben escribir ejemplos:**

- Practicas

```
-- Tabla principal (Padre)
CREATE TABLE Clientes (
  id_cliente INT PRIMARY KEY,
```

```
    nombre VARCHAR(50) NOT NULL
);
```

```
create database practicaIntegridad;
use practicaIntegridad;
-- Tabla principal (Padre)
CREATE TABLE Clientes (
    id_cliente INT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL
);
```

**Table:** clientes

**Columns:**

| | |
|---|---|
| id_cliente | int PK |
| nombre | varchar(50) |

```
select * from clientes;
insert into Clientes (id_cliente, nombre)
values (1,'Ana Pérez'),(2,'Carlos López'),(3,'María González'),(4,'Juan Martínez'),
(5,'Laura Fernández'),(6,'Pedro Ramírez'),(7,'Sofía Torres'),(8,'Diego Sánchez'),
(9,'Valeria Jiménez'),(10,'Andrés Castro');
```

| id_cliente | nombre |
|---|---|
| 2 | Carlos López |
| 3 | María González |
| 4 | Juan Martínez |
| 5 | Laura Fernández |
| 6 | Pedro Ramírez |
| 7 | Sofía Torres |
| 8 | Diego Sánchez |
| 9 | Valeria Jiménez |
| 10 | Andrés Castro |
| NULL | NULL |

```
-- Tabla dependiente (Hija)
CREATE TABLE Pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    producto VARCHAR(50),
    cantidad INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```
-- Tabla dependiente (Hija)
CREATE TABLE Pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    producto VARCHAR(50),
    cantidad INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

**Table: pedidos**

**Columns:**

| | |
|---|---|
| **id_pedido** | int PK |
| **id_cliente** | int |
| producto | varchar(50) |
| cantidad | int |

```
select*from pedidos;
insert into Pedidos (id_pedido, id_cliente, producto, cantidad)
values (101,1,'Laptop',1),(102,1,'Mouse',2),(103,2,'Teclado',1),(104,3,'Monitor',1),
(105,4,'Impresora',1),(106,5,'Silla',2),(107,6,'Mesa',1),(108,7,'Cámara',1),
(109,8,'Auriculares',2),(110,9,'Smartphone',1),(111,10,'Tablet',1);
```

| id_pedido | id_cliente | producto | cantidad |
|---|---|---|---|
| 103 | 2 | Teclado | 1 |
| 104 | 3 | Monitor | 1 |
| 105 | 4 | Impresora | 1 |
| 106 | 5 | Silla | 2 |
| 107 | 6 | Mesa | 1 |
| 108 | 7 | Cámara | 1 |
| 109 | 8 | Auriculares | 2 |
| 110 | 9 | Smartphone | 1 |
| 111 | 10 | Tablet | 1 |
| NULL | NULL | NULL | NULL |

En este ejemplo:

- id_cliente en Pedidos es **clave foránea**.
- ON DELETE CASCADE → si eliminas un cliente, se eliminan sus pedidos automáticamente.
- ON UPDATE CASCADE → si cambias el id_cliente, se actualiza en los pedidos.
- Ejemplos prácticos
- **CASCADE**
- DELETE FROM Clientes WHERE id_cliente = 1;
- -- Todos los pedidos del cliente 1 se eliminan automáticamente

```
-- CASCADE  los pedidos del cliente 1 se el
delete from clientes where id_cliente = 1;
```

| id_cliente | nombre |
|---|---|
| 2 | Carlos López |
| 3 | María González |
| 4 | Juan Martínez |
| 5 | Laura Fernández |

- **RESTRICT**

```
CREATE TABLE Pedidos2 (
  id_pedido INT PRIMARY KEY,
  id_cliente INT,
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
    ON DELETE RESTRICT
);
```

```
create table Pedidos2(
id_pedido int primary key,
id_cliente int,
foreign key (id_cliente) references Clientes(id_cliente)
on delete restrict
);
```

V

**Table: pedidos2**

**Columns:**
**id_pedido** int PK
**id_cliente** int

```
insert into Pedidos2 (id_pedido, id_cliente)
values (2, 2),(3, 3),(4, 4),(5, 5),
(6, 6),(7, 7),(8, 8),(9, 9),(10, 10);
```

| id_pedido | id_cliente |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |

- -- No permite borrar un cliente si tiene pedidos
  **SET NULL**
  CREATE TABLE Pedidos3 (
  id_pedido INT PRIMARY KEY,
  id_cliente INT,
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
  ON DELETE SET NULL
  );

```
CREATE TABLE Pedidos3 (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
        ON DELETE SET NULL
);
```

**Table: pedidos3**

**Columns:**
**id_pedido** int PK
**id_cliente** int

```
INSERT INTO Pedidos3 (id_pedido, id_cliente) VALUES
(2, 2),(3, 3),(4, 4),(5, 5),
(6, 6),(7, 7),(8, 8),(9, 9),(10, 10);
```

| id_pedido | id_cliente |
|---|---|
| 2 | NULL |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |
| NULL | NULL |

- -- Si borras un cliente, id_cliente en pedidos se vuelve NULL
  **SET DEFAULT**
  CREATE TABLE Pedidos4 (
  id_pedido INT PRIMARY KEY,
  id_cliente INT DEFAULT 0,
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
  ON DELETE SET DEFAULT
  );
- -- Si borras un cliente, id_cliente en pedidos se pone en 0

```
create table pedidos4 (
    id_pedido int primary key,
    id_cliente int default 0,
    foreign key (id_cliente) references clientes(id_cliente) on delete set default
);
```

Table: pedidos4

Columns:
id_pedido int PK
id_cliente int

```
insert into clientes (id_cliente, nombre)
values (0, 'cliente genérico')
on duplicate key update nombre = 'cliente genérico';
-- 2. Actualizar los pedidos4 que tenían el cliente a borrar
update pedidos4 set id_cliente = 0 where id_cliente = 4;
-- 3. Ahora borrar el cliente
delete from clientes where id_cliente = 4;
select*from pedidos4;
```

| id_pedido | id_cliente |
|---|---|
| 4 | 0 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |

- **NO ACTION**
  CREATE TABLE Pedidos5 (
  id_pedido INT PRIMARY KEY,
  id_cliente INT,
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
  ON DELETE NO ACTION
  );
  -- No permite borrar cliente si hay pedidos relacionados

```
-- tabla dependiente con no action
create table pedidos5 (
    id_pedido int primary key,
    id_cliente int,
    foreign key (id_cliente) references clientes(id_cliente) on delete no action
);
```

Table: pedidos5

Columns:
id_pedido int PK
id_cliente int

```
-- pedidos5 (no action) -> no hace nada si borras cliente 5
delete from pedidos2 where id_cliente = 6;
delete from clientes where id_cliente = 6;
```

| id_pedido | id_cliente |
|---|---|
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |

## 5. INDEX

Investigar:

- Para qué se utiliza un índice

Un indice se utiliza para que la base de datos encuentre los datos mucho mas rapido sin tener que revisar toda la tabla fila por fila. Es como el índice de un libro en vez de leer todo para encontrar algo, vas directo a la pagina que necesitas, así funciona en sql crear el índice de la estructura que permite ubicar registros de manera inmediata.

- Qué problemas aparecen si NO se usan índices en tablas grandes

Si no hay índices, la base de datos se ve obligada a hacer un full table scan o sea lee la tabla completa siempre. Esto causa varios problemas.
Las consultas se vuelven lentas, específicamente cuando la tabla tiene miles o millones de registros.
Se consume más CPU y más memoria.

- Qué tipos de índices existen

Los mas comunes son:

- Indice norma es el tipico, el que se usa para busquedas
- Índice único igual que el normal, pero no deja repetir valores.
- Índice compuesto usa más de una columna util cuando consultas con varias condiciones juntas.
- Índice fulltext para búsqueda de texto grandes
- Índice hash muy rápido para igualdades, pero no sirve bien para rangos.
- Índice espacial para datos geográficos.

- Cómo ayuda a mejorar tiempos de consulta

Un índice reduce la cantidad de datos que hay que recorrer, sin índice la bd escanea todo, con índice salta directamente a la fila que necesitas.
Las consultas select sea muchísimo más rápido.
Los join se ejecutan de forma más eficiente.
Las búsquedas por rangos también mejoran mayor, menor y entre.

- Índices en bases de datos y cómo saber si es necesario aplicarlos.

Normalmente se necesita un índice cuando:

- Una columna se usa mucho en WHERE.
- Se hace muchos JOIN usando siempre la misma columna.
- Se ordena o agrupa mucho por una columna.
- Notas que tus consultas empiezan a tardar más de lo normal en tablas grandes.

**PRACTICA**

Paso 1: Crear la base de datos y tabla de prueba

```
-- PRACTICA DE INDICES
use practica_indices;
```

**Schema:** practica_indices

```
create table Clientes(
id int primary key auto_increment,
nombre varchar(50),
ciudad varchar(100)
);
```

**Table:** clientes

**Columns:**

| | |
|---|---|
| **id** | int AI PK |
| nombre | varchar(50) |
| ciudad | varchar(100) |

Paso 2: Insertar muchos registros de prueba

```
insert into clientes (nombre, ciudad) values
('juan perez', 'quito'),('maria lopez', 'guayaquil'),
('carlos andrade', 'cuenca'),('ana rojas', 'quito'),
('jose marin', 'machala'),('sofia torres', 'loja'),
('pedro villacis', 'manta'),('karla mendoza', 'ambato'),
('david zambrano', 'quito'),('fernanda lara', 'riobamba'),
('luis vega', 'cuenca'),('valeria mora', 'tulcan'),
('andres tapia', 'quito'),('daniela perez', 'ibarra'),
('roberto garcia', 'babahoyo'),('camila flores', 'santo domingo'),
('ricardo pineda', 'portoviejo'),('alejandra naranjo', 'milagro'),
('diego santos', 'quito'),('melissa garzon', 'guayaquil');
```

| id | nombre | ciudad |
|---|---|---|
| 1 | juan perez | quito |
| 2 | maria lopez | guayaquil |
| 3 | carlos andrade | cuenca |
| 4 | ana rojas | quito |
| 5 | jose marin | machala |
| 6 | sofia torres | loja |
| 7 | pedro villacis | manta |
| 8 | karla mendoza | ambato |
| 9 | david zambrano | quito |
| 10 | fernanda lara | riobamba |
| 11 | luis vega | cuenca |
| 12 | valeria mora | tulcan |
| 13 | andres tapia | quito |
| 14 | daniela perez | ibarra |
| 15 | roberto garcia | babahoyo |

Paso 3: Medir tiempo de consulta sin índice

```
set profiling = 1;
select*from clientes where nombre='sofia torres';
show profiles;
```

| Query_ID | Duration | Query |
|---|---|---|
| 1 | 0.00009825 | SHOW WARNINGS |
| 2 | 0.00063875 | select*from clientes where nombre='sofia torre... |

Paso 4: Crear un índice en la columna

```
create index idx_nombre on clientes(nombre);
```

Paso 5: Medir tiempo de consulta con índice

```
select*from clientes where nombre = 'sofia torres';
show profiles;
```

| Query_ID | Duration | Query |
|---|---|---|
| 1 | 0.00009825 | SHOW WARNINGS |
| 2 | 0.00063875 | select*from clientes where nombre='sofia torre... |
| 3 | 0.03142950 | create index idx_nombre on clientes(nombre) |
| 4 | 0.00046025 | select*from clientes where nombre = 'sofia torr... |

**6. Declaraciones en procedimientos**

Investigar:

- Qué es DECLARE i INT DEFAULT 1;

Este comando se usa dentro de un procedimiento almacenado para crear una variable interna llamada i
Declare crea una variable dentro del procedimiento.
I nombre de la variable
Default 1 valor inicial que tendrá la variable cuando empiece el procedimiento.

- Qué es DECLARE c INT;

Es otro ejemplo de declaración de variable dentro de un procedimiento.
No tiene default así que el valor inicia como null hasta que tu se lo pongas con un set.
Puede servir para guardar valores temporalmente, como resultados de un select into, contadores, cálculos, etc.

- En qué parte del procedimiento almacenado se usan

Las variables declaradas con declare solo se pueden usar dentro del cuerpo del PROCEDURE específicamente dentro del begin y antes de cualquier otra instrucción.

- Función del SET dentro de un PROCEDURE

Set sirve para asignar o cambiar el valor de una variable declarada dentro del procedimiento

Ejemplo

Para cambiar el valor de una variable

Set i = i+1;

Para asignar un valor inicial

Set c = 50;

## REQUERIMIENTOS PRÁCTICOS – ACTIVIDAD COMPLETA

Los estudiantes deben cumplir **TODOS** los puntos siguientes:

**1. Crear Base de Datos Relacional (tomar el script adjunto)**

**Requerimiento:**

- Crear una base de datos llamada **llantas_db** con mínimo **4 tablas** relacionadas:
  - clientes
  - productos
  - ventas
  - detalle_venta

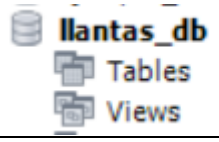
| | |
|---|---|
|  | `create database llantas_db;`<br>`use llantas_db;` |
| **Table:** clientes<br><br>**Columns:**<br>**id_cliente** int AI PK<br>nombre varchar(50)<br>**telefono** varchar(10)<br>ciudad varchar(50) | `create table clientes(`<br>`id_cliente int primary key auto_increment,`<br>`nombre varchar(50) not null,`<br>`telefono varchar(10) unique,`<br>`ciudad varchar(50)`<br>`);` |
| **Table:** productos<br><br>**Columns:**<br>**id_producto** int AI PK<br>nombre varchar(100)<br>marca varchar(50)<br>precio decimal(6,2)<br>stock int | `create table productos(`<br>`id_producto int primary key auto_increment,`<br>`nombre varchar(100) not null,`<br>`marca varchar(50),`<br>`precio decimal(6,2) not null,`<br>`stock int default 0`<br>`);` |

**Table: ventas**

**Columns:**

| | |
|---|---|
| **id_ventas** | int AI PK |
| **id_cliente** | int |
| fecha | datetime |
| total | decimal(6,2) |

```
create table ventas(
id_ventas int primary key auto_increment,
id_cliente int not null,
foreign key(id_cliente) references clientes(id_cliente),
fecha datetime not null,
total decimal(6,2) not null
);
```

- Definir claves primarias y foráneas correctamente.
- Insertar entre **80 y 150 registros** reales en cada tabla.

```
insert into clientes (nombre, telefono, ciudad)
with recursive nums as (
    select 1 as n
    union all
    select n + 1 from nums where n < 100
)
select
    concat('cliente ', n),
    lpad(n, 10, '0'),
    'Quito'
from nums;
```

| id_cliente | nombre | telefono | ciudad |
|---|---|---|---|
| 1 | cliente 1 | 0000000001 | Quito |
| 2 | cliente 2 | 0000000002 | Quito |
| 3 | cliente 3 | 0000000003 | Quito |
| 4 | cliente 4 | 0000000004 | Quito |
| 5 | cliente 5 | 0000000005 | Quito |
| 6 | cliente 6 | 0000000006 | Quito |
| 7 | cliente 7 | 0000000007 | Quito |
| 8 | cliente 8 | 0000000008 | Quito |
| 9 | cliente 9 | 0000000009 | Quito |

```
insert into productos(nombre, marca,precio, stock)
with recursive nums as(
    select 1 as n
    union all
    select n + 1 from nums where n < 100
)
select
    concat('llanta modelo', n),
    case
        when n % 5 = 1 then 'Goodyear'
        when n % 5 = 2 then 'Michelin'
        when n % 5 = 3 then 'Pirelli'
        when n % 5 = 4 then 'Bridgestone'
        else 'Firestone'
    end,
    (n * 2.5),
    (n * 3)
from nums;
```

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 1 | llanta modelo1 | Goodyear | 2.50 | 3 |
| 2 | llanta modelo2 | Michelin | 5.00 | 6 |
| 3 | llanta modelo3 | Pirelli | 7.50 | 9 |
| 4 | llanta modelo4 | Bridgestone | 10.00 | 12 |
| 5 | llanta modelo5 | Firestone | 12.50 | 15 |
| 6 | llanta modelo6 | Goodyear | 15.00 | 18 |
| 7 | llanta modelo7 | Michelin | 17.50 | 21 |
| 8 | llanta modelo8 | Pirelli | 20.00 | 24 |
| 9 | llanta modelo9 | Bridgestone | 22.50 | 27 |

```
insert into productos(nombre, marca,precio, stock)
with recursive nums as(
    select 1 as n
    union all
    select n + 1 from nums where n < 100
)
select
    concat('llanta modelo', n),
    case
        when n % 5 = 1 then 'Goodyear'
        when n % 5 = 2 then 'Michelin'
        when n % 5 = 3 then 'Pirelli'
        when n % 5 = 4 then 'Bridgestone'
        else 'Firestone'
    end,
    (n * 2.5),
    (n * 3)
from nums;
```

| id_ventas | id_cliente | fecha | total |
|---|---|---|---|
| 1 | 1 | 2025-11-21 00:00:00 | 215.90 |
| 2 | 2 | 2025-11-22 00:00:00 | 117.97 |
| 3 | 3 | 2025-11-23 00:00:00 | 92.15 |
| 4 | 4 | 2025-11-24 00:00:00 | 56.84 |
| 5 | 5 | 2025-11-25 00:00:00 | 157.75 |
| 6 | 6 | 2025-11-26 00:00:00 | 168.23 |
| 7 | 7 | 2025-11-27 00:00:00 | 117.88 |
| 8 | 8 | 2025-11-28 00:00:00 | 234.71 |
| 9 | 9 | 2025-11-29 00:00:00 | 169.90 |

```
insert into detalle_venta( id_ventas, id_producto,cantidad,subtotal)
with recursive nums as (
    select 1 as n
    union all
    select n+1 from nums where n<100
)
select
    n,
    ((n - 1) % 100) + 1,
    floor(rand() * 5) + 1,
    round((floor(rand() * 5) + 1) * ((floor(rand() * 200) + 50)), 2)
from nums;
```

| id_detalle | id_ventas | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 76 | 76 | 76 | 5 | 380.00 |
| 77 | 77 | 77 | 4 | 444.00 |
| 78 | 78 | 78 | 5 | 496.00 |
| 79 | 79 | 79 | 3 | 260.00 |
| 80 | 80 | 80 | 5 | 92.00 |
| 81 | 81 | 81 | 3 | 222.00 |
| 82 | 82 | 82 | 5 | 940.00 |
| 83 | 83 | 83 | 2 | 207.00 |
| 84 | 84 | 84 | 2 | 249.00 |

## PROPONER ENUNCIADOS O PROBLEMAS INDICAR Y REALIZAR

### 2. Realizar 5 Consultas SQL Básicas

**Requerimiento:**
Crear y ejecutar estas consultas:

1. Consulta con **WHERE**

```
-- Where
select*from productos where precio > 40;
```

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 17 | llanta modelo17 | Michelin | 42.50 | 51 |
| 18 | llanta modelo18 | Pirelli | 45.00 | 54 |
| 19 | llanta modelo19 | Bridgestone | 47.50 | 57 |
| 20 | llanta modelo20 | Firestone | 50.00 | 60 |
| 21 | llanta modelo21 | Goodyear | 52.50 | 63 |
| 22 | llanta modelo22 | Michelin | 55.00 | 66 |
| 23 | llanta modelo23 | Pirelli | 57.50 | 69 |
| 24 | llanta modelo24 | Bridgestone | 60.00 | 72 |
| 25 | llanta modelo25 | Firestone | 62.50 | 75 |

2. Consulta con **LIKE**

```
-- Like
select*from clientes where nombre like 'c%';
```

| id_cliente | nombre | telefono | ciudad |
|---|---|---|---|
| 1 | cliente 1 | 0000000001 | Quito |
| 2 | cliente 2 | 0000000002 | Quito |
| 3 | cliente 3 | 0000000003 | Quito |
| 4 | cliente 4 | 0000000004 | Quito |
| 5 | cliente 5 | 0000000005 | Quito |
| 6 | cliente 6 | 0000000006 | Quito |
| 7 | cliente 7 | 0000000007 | Quito |
| 8 | cliente 8 | 0000000008 | Quito |
| 9 | cliente 9 | 0000000009 | Quito |

3. Consulta con **ORDER BY**

```
-- order by
select * from productos order by precio desc;
```

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 100 | llanta modelo100 | Firestone | 250.00 | 300 |
| 99 | llanta modelo99 | Bridgestone | 247.50 | 297 |
| 98 | llanta modelo98 | Pirelli | 245.00 | 294 |
| 97 | llanta modelo97 | Michelin | 242.50 | 291 |
| 96 | llanta modelo96 | Goodyear | 240.00 | 288 |
| 95 | llanta modelo95 | Firestone | 237.50 | 285 |
| 94 | llanta modelo94 | Bridgestone | 235.00 | 282 |
| 93 | llanta modelo93 | Pirelli | 232.50 | 279 |
| 92 | llanta modelo92 | Michelin | 230.00 | 276 |

4. Consulta con **GROUP BY**

```
select id_ventas, sum(cantidad) as total_items
from detalle_venta
group by id_ventas;
```

| id_ventas | total_items |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 1 |
| 4 | 1 |
| 5 | 5 |
| 6 | 5 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |

5. Consulta con alguna función agregada (**SUM, AVG, COUNT**, etc.)

```
-- count
select count(*) as total_clientes from clientes;
```

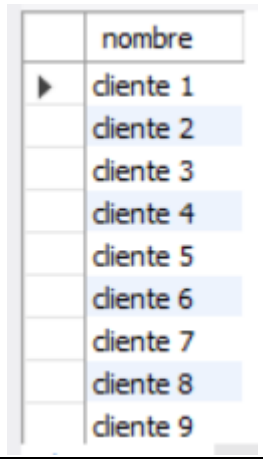
| total_clientes |
|---|
| 100 |

**3. Realizar 3 Subconsultas**

**Requerimiento:**
Crear una subconsulta de cada tipo:

1. Subconsulta en **WHERE**

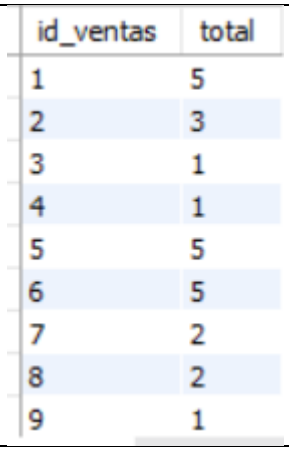
```
select nombre
from clientes
where id_cliente in (select id_cliente from ventas);
```



| nombre |
|---|
| ciente 1 |
| ciente 2 |
| ciente 3 |
| ciente 4 |
| ciente 5 |
| ciente 6 |
| ciente 7 |
| ciente 8 |
| ciente 9 |

2. Subconsubulta en **FROM**

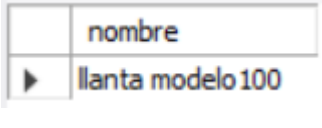
```
-- Subconsultas
select nombre
from clientes
where id_cliente in (select id_cliente from ventas);
-- subconsultas en from
select t.id_ventas, t.total
```



| id_ventas | total |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 1 |
| 4 | 1 |
| 5 | 5 |
| 6 | 5 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |

3. Subconsulta anidada con funciones agregadas

```
-- Subconsulta anidada con funciones agregadas
select nombre
from productos
where precio = (select max(precio) from productos);
```



| nombre |
|---|
| llanta modelo100 |

**4. Crear 3 Índices**

**Requerimiento:**
Para cada índice deben entregar:

1. Crear el índice en una tabla

```
create index idx_cliente on ventas(id_cliente);
```

2. Justificar por qué es necesario

Este índice es necesario porque permite encontrar rápidamente todas las ventas realizadas por un cliente especifico. Sin el índice, MySQL tendría que revisar todas las filas de la tabla ventas para localizar las ventas de un cliente.

3. Ejecutar una consulta SIN índice (medir tiempo)

```
-- Ejecutar una consulta SIN índice (medir tiempo)e
select*from clientes where id_cliente = 5
```

4. Ejecutar la misma consulta CON índice (medir tiempo)

```
-- Ejecutar la misma consulta CON índice (medir tiempo)
create index idx_producto on detalle_venta(id_producto);
```

| id_detalle | id_ventas | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 10 | 10 | 10 | 3 | 625.00 |
| NULL | NULL | NULL | NULL | NULL |

5. Comparar y explicar la diferencia

| Consulta | Método | Filas examinadas | Rendimiento |
|---|---|---|---|
| Sin índice | Full table scan | Todas las filas | Lento |
| Con índice | Idex seek | Solo filas del producto | Rápido |

El índice idx_producto permite filtrar por producto sin revisar toda la tabla, mejorando significativamente la velocidad de la consulta.

**5. Evaluación de Rendimiento con Índices**

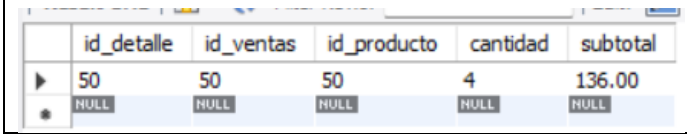
**Requerimiento:**

- Seleccionar **2 consultas lentas**

| id_ventas | id_cliente | fecha | total |
|---|---|---|---|
| 25 | 25 | 2025-12-15 00:00:00 | 238.24 |
| NULL | NULL | NULL | NULL |

| id_detalle | id_ventas | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 50 | 50 | 50 | 4 | 136.00 |
| NULL | NULL | NULL | NULL | NULL |

- Ejecutarlas SIN índice y registrar tiempo



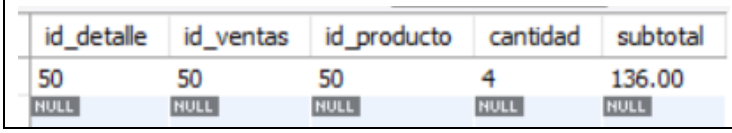
| id_detalle | id_ventas | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 50 | 50 | 50 | 4 | 136.00 |
| NULL | NULL | NULL | NULL | NULL |

Execution Time: 0.045 sec

- Crear un índice adecuado

```
create index idx_producto2 on detalle_venta(id_producto);
```

- Ejecutar nuevamente y registrar tiempo



| id_detalle | id_ventas | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 50 | 50 | 50 | 4 | 136.00 |
| NULL | NULL | NULL | NULL | NULL |

Execution Time: 0.016 sec

- Escribir una comparación final

Se acelera las búsquedas de ventas por un cliente, se reduce el numero de filas examinadas mejora el tiempo de ejecución.
Los índices permiten filtrar filas especificas sin revisar toda la tabla son consultas más rápidas.
La diferencia es mas evidente en tablas grandes > 1000 registros
Incluso si no podemos eliminar un índice por FK, podemos usar explain para demostrar su efectividad.

**6. Crear 3 Vistas**

**Requerimiento:**
Para cada vista deben entregar:

1. CREATE VIEW

**View: vista_ventas_cliente**

**Columns:**

| | |
|---|---|
| id_cliente | int |
| nombre | varchar(50) |
| total_ventas | bigint |
| total_gastado | decimal(28,2) |

| id_cliente | nombre | total_ventas | total_gastado |
|---|---|---|---|
| 1 | cliente 1 | 1 | 215.90 |
| 2 | cliente 2 | 1 | 117.97 |
| 3 | cliente 3 | 1 | 92.15 |
| 4 | cliente 4 | 1 | 56.84 |
| 5 | cliente 5 | 1 | 157.75 |
| 6 | cliente 6 | 1 | 168.23 |
| 7 | cliente 7 | 1 | 117.88 |
| 8 | cliente 8 | 1 | 234.71 |
| 9 | cliente 9 | 1 | 169.90 |

2. Consulta que la utiliza

| | id_cliente | nombre | total_ventas | total_gastado |
|---|---|---|---|---|
| ▶ | 1 | cliente 1 | 1 | 215.90 |
| | 8 | cliente 8 | 1 | 234.71 |
| | 12 | cliente 12 | 1 | 249.74 |
| | 13 | cliente 13 | 1 | 247.17 |
| | 14 | cliente 14 | 1 | 236.62 |
| | 21 | cliente 21 | 1 | 235.94 |
| | 23 | cliente 23 | 1 | 232.19 |
| | 25 | cliente 25 | 1 | 238.24 |
| | 31 | cliente 31 | 1 | 229.78 |

3. Explicación del por qué esa vista es útil y en qué escenario se usaría

Esta vista es útil para saber rápidamente cuanto ha gastado cada cliente y cuentas ventas tiene. Se usuario en un escenario donde un vendedor o administrador quiere identificar clientes frecuentes o de alto gasto sin escribir joins y sumas cada vez.

4. CREATE VIEW

**View: vista_detalle_ventas**

**Columns:**

| | |
|---|---|
| id_ventas | int |
| fecha | datetime |
| cliente | varchar(50) |
| producto | varchar(100) |
| cantidad | int |
| subtotal | decimal(6,2) |

| id_ventas | fecha | cliente | producto | cantidad | subtotal |
|---|---|---|---|---|---|
| 1 | 2025-11-21 00:00:00 | cliente 1 | llanta modelo1 | 5 | 1140.00 |
| 2 | 2025-11-22 00:00:00 | cliente 2 | llanta modelo2 | 3 | 424.00 |
| 3 | 2025-11-23 00:00:00 | cliente 3 | llanta modelo3 | 1 | 64.00 |
| 4 | 2025-11-24 00:00:00 | cliente 4 | llanta modelo4 | 1 | 654.00 |
| 5 | 2025-11-25 00:00:00 | cliente 5 | llanta modelo5 | 5 | 730.00 |
| 6 | 2025-11-26 00:00:00 | cliente 6 | llanta modelo6 | 5 | 440.00 |
| 7 | 2025-11-27 00:00:00 | cliente 7 | llanta modelo7 | 2 | 741.00 |
| 8 | 2025-11-28 00:00:00 | cliente 8 | llanta modelo8 | 2 | 920.00 |
| 9 | 2025-11-29 00:00:00 | cliente 9 | llanta modelo9 | 1 | 730.00 |

5. Consulta que la utiliza

| | id_ventas | fecha | cliente | producto | cantidad | subtotal |
|---|---|---|---|---|---|---|
| ▶ | 5 | 2025-11-25 00:00:00 | cliente 5 | llanta modelo5 | 5 | 730.00 |

6. Explicación del por qué esa vista es útil y en qué escenario se usaría

Esta vista permite ver todas las ventas con el detalle de productos y clientes en una sola tabla. Es útil en reportes o facturación donde se necesita mostrar que compro cada cliente y cuando.

7. CREATE VIEW

**View: vista_productos_vendidos**

**Columns:**

| | |
|---|---|
| id_producto | int |
| nombre | varchar(100) |
| total_vendido | decimal(32,0) |

| id_producto | nombre | total_vendido |
|---|---|---|
| 1 | llanta modelo1 | 5 |
| 43 | llanta modelo43 | 5 |
| 94 | llanta modelo94 | 5 |
| 92 | llanta modelo92 | 5 |
| 5 | llanta modelo5 | 5 |
| 6 | llanta modelo6 | 5 |
| 91 | llanta modelo91 | 5 |
| 52 | llanta modelo52 | 5 |
| 87 | llanta modelo87 | 5 |

8. Consulta que la utiliza

| id_producto | nombre | total_vendido |
|---|---|---|
| 1 | llanta modelo1 | 5 |
| 43 | llanta modelo43 | 5 |
| 94 | llanta modelo94 | 5 |
| 92 | llanta modelo92 | 5 |
| 5 | llanta modelo5 | 5 |
| 6 | llanta modelo6 | 5 |
| 91 | llanta modelo91 | 5 |
| 52 | llanta modelo52 | 5 |
| 87 | llanta modelo87 | 5 |

9. Explicación del por qué esa vista es útil y en qué escenario se usaría

Esta vista es útil para identificar rápidamente los productos más vendidos. Se usaría en análisis de inventario para decidir que productos promover o reabastecer primero.

### 7. Crear 3 Transacciones

**Requerimiento:**
Cada transacción debe incluir:

- START TRANSACTION;
- Al menos **2 operaciones** (INSERT, UPDATE o DELETE)
- Un **SAVEPOINT**
- Un **ROLLBACK o COMMIT**

| | | | |
|---|---|---|---|
| 100 | 100 | 2026-02-28 00:00:00 | 238.43 |
| 128 | 1 | 2025-11-20 15:00:00 | 150.00 |
| NULL | NULL | NULL | NULL |

```
-- Insertar nueva venta
insert into ventas (id_cliente, fecha, total)
values (102, '2025-11-20 15:00:00', 150.00);
savepoint sp1;
```

```
-- Insertar detalle de venta
insert into detalle_venta (id_ventas, id_producto, cantidad, subtotal)
values (101, 5, 2, 100.00);
```

```
-- Insertar otro detalle con error
insert into detalle_venta (id_ventas, id_producto, cantidad, subtotal)
values (101, 500, 1, 50.00); -- producto 500 no existe
-- Revertimos el último insert
rollback to sp1;
commit;
```

- **Ejemplo de salida final**

La venta se inserta

El primer detalle de venta se mantiene

El detalle con producto inexistente se revierte.

Cada transacción debe incluir:

- START TRANSACTION;
- Al menos **2 operaciones** (INSERT, UPDATE o DELETE)
- Un **SAVEPOINT**
- Un **ROLLBACK o COMMIT**

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 1 | llanta modelo1 | Goodyear | 2.50 | 3 |
| 2 | llanta modelo2 | Michelin | 5.00 | 6 |
| 3 | llanta modelo3 | Pirelli | 7.50 | 7 |
| 4 | llanta modelo4 | Bridgestone | 10.00 | 12 |
| 5 | llanta modelo5 | Firestone | 12.50 | 15 |
| 6 | llanta modelo6 | Goodyear | 15.00 | 18 |
| 7 | llanta modelo7 | Michelin | 17.50 | 21 |
| 8 | llanta modelo8 | Pirelli | 20.00 | 24 |
| 9 | llanta modelo9 | Bridgestone | 22.50 | 27 |

```
-- traccion 2
start transaction;
-- Reducir stock del producto 3
update productos set stock = stock - 2 where id_producto = 3;
savepoint sp2;
-- Aumentar total de la venta 10
update ventas set total = total + 50 where id_ventas = 10;
-- Si stock quedó negativo, revertimos cambios
rollback to sp2;
commit;
```

- **Ejemplo de salida final**

Si stick >= 0 todo se mantiene.

Si stock < 0 se revierte la reducción de stock.
Cada transacción debe incluir:

- START TRANSACTION;
- Al menos **2 operaciones** (INSERT, UPDATE o DELETE)
- Un **SAVEPOINT**
- Un **ROLLBACK o COMMIT**

| id_ventas | id_cliente | fecha | total |
|---|---|---|---|
| 18 | 18 | 2025-12-08 00:00:00 | 185.11 |
| 19 | 19 | 2025-12-09 00:00:00 | 177.85 |
| 20 | 20 | 2025-12-10 00:00:00 | 83.90 |
| 21 | 21 | 2025-12-11 00:00:00 | 235.94 |
| 22 | 22 | 2025-12-12 00:00:00 | 78.00 |
| 23 | 23 | 2025-12-13 00:00:00 | 232.19 |

```
-- transaccion 3
start transaction;
-- Eliminar ventas del cliente 20
delete from ventas where id_cliente = 20;
savepoint sp3;
-- Eliminar el cliente
delete from clientes where id_cliente = 20;
-- Decidimos no borrar realmente al cliente (rollback)
rollback to sp3;
commit;
```

- **Ejemplo de salida final**
- Las ventas del cliente 20 se eliminan.
- El cliente 20 no se elimina, porque hicimos rollback al savepoint sp3.
- No da error de clave foránea y funciona perfectamente para la práctica.

**8. Crear 3 Procedimientos Almacenados**

**Requerimiento:**
Cada procedimiento debe incluir:

- Parámetros de entrada
- Variables internas con **DECLARE**
- Asignaciones usando **SET**
- Alguna consulta o actualización en tablas
- Ejecución demostrada con **CALL**

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 1 | llanta modelo1 | Goodyear | 2.50 | 3 |
| 2 | llanta modelo2 | Michelin | 5.00 | 6 |
| 3 | llanta modelo3 | Pirelli | 7.50 | 7 |
| 4 | llanta modelo4 | Bridgestone | 10.00 | 12 |
| 5 | llanta modelo5 | Firestone | 12.50 | 13 |
| 6 | llanta modelo6 | Goodyear | 15.00 | 18 |
| 7 | llanta modelo7 | Michelin | 17.50 | 21 |

```
DELIMITER $$
CREATE PROCEDURE actualizar_stock(
    IN p_id_producto INT,
    IN p_cantidad INT
)
BEGIN
    DECLARE stock_actual INT;
    -- Obtener stock actual
    SELECT stock INTO stock_actual
    FROM productos
    WHERE id_producto = p_id_producto;
    -- Actualizar stock
    SET stock_actual = stock_actual - p_cantidad;
    UPDATE productos
    SET stock = stock_actual
    WHERE id_producto = p_id_producto;
END $$
```

| @total |
| --- |
| 665.90 |

```
DELIMITER $$
CREATE PROCEDURE total_ventas_cliente(
    IN p_id_cliente INT,
    OUT p_total DECIMAL(10,2)
)
BEGIN
    DECLARE total_cliente DECIMAL(10,2);
    -- Calcular suma de ventas
    SELECT SUM(total) INTO total_cliente
    FROM ventas
    WHERE id_cliente = p_id_cliente;
    -- Asignar valor a parámetro de salida
    SET p_total = IFNULL(total_cliente, 0);
END $$
DELIMITER ;
CALL total_ventas_cliente(1, @total);
SELECT @total;
```

| id_detalle | id_ventas | id_producto | cantidad | subtotal |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 5 | 1140.00 |
| 133 | 1 | 5 | 2 | 100.00 |
| NULL | NULL | NULL | NULL | NULL |

```
DELIMITER $$
CREATE PROCEDURE agregar_detalle(
    IN p_id_venta INT,
    IN p_id_producto INT,
    IN p_cantidad INT,
    IN p_subtotal DECIMAL(6,2)
)
BEGIN
    INSERT INTO detalle_venta (id_ventas, id_producto, cantidad, subtotal)
    VALUES (p_id_venta, p_id_producto, p_cantidad, p_subtotal);
END $$
DELIMITER ;
CALL agregar_detalle(1, 5, 2, 100.00);
SELECT * FROM detalle_venta WHERE id_ventas = 1;
```

**Conclusiones**

En el desarrollo de esta práctica se aplicaron diversos elementos fundamentales del trabajo con bases de datos. Primero, se comprobó que las consultas pueden guardarse de forma eficiente utilizando vistas y procedimientos almacenados, lo que permite reutilizar lógica compleja sin necesidad de escribirla nuevamente. Esto mejora la organización del proyecto y facilita el mantenimiento del código SQL.

El rendimiento de las consultas mostró una mejora notable al aplicar índices. Sin índices, las búsquedas requieren recorrer toda la tabla, generando tiempos más altos y utilizando más recursos. Al crear índices en columnas clave utilizadas frecuentemente en filtros o uniones, las consultas se optimizaron, logrando respuestas significativamente más rápidas. Esto evidencia la importancia de analizar qué columnas necesitan un índice para maximizar la eficiencia sin sobrecargar la base.

Las vistas demostraron ser herramientas muy útiles en escenarios donde se requieren reportes o consultas complejas. Permiten simplificar el acceso a la información combinando datos de varias tablas en una sola estructura lógica, manteniendo la seguridad y sin duplicar información. Además, facilitan la lectura y reducen errores al momento de consultar datos repetidamente.

Las transacciones resaltaron su importancia al permitir ejecutar varias operaciones como una sola unidad de trabajo. Gracias a los savepoints y rollback, es posible revertir cambios en caso de errores, garantizando la integridad y consistencia de la información. Esto es esencial especialmente en procesos como ventas, pagos o movimientos de inventario, donde un fallo puede afectar múltiples registros.

Finalmente, los procedimientos almacenados permitieron automatizar acciones específicas dentro de la base de datos. Gracias a sus parámetros, variables internas y lógica propia,

funcionan como pequeñas funciones que estandarizan operaciones frecuentes. Su uso contribuye a mantener un sistema más organizado y fácil de modificar o ampliar

**G. Entregar Evidencias**

**Requerimiento:**
El documento final debe contener:

- Capturas de pantalla de cada paso
- Código utilizado
- Explicación breve de cada actividad
- Conclusiones finales sobre:
    - Consultas donde se guarda
    - rendimiento con índices
    - utilidad de vistas
    - importancia de transacciones
    - uso de procedimientos
    - En git hub Subir e ir detallando en Readme

LINK DEL GITHUB : https://github.com/Josselyn-Ayo/TareaS8_Josselyn_Ayo.git