# BASE DE DATOS ACTIVIDAD A DESARROLLAR

## **Nombre:** Kevin Mendoza

## 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**

Conjunto de símbolos que se pueden almacenar en una columna de texto o en una base de datos

- **Qué es COLLATE**

Compara y ordena los caracteres, es decir, lo hace de forma jerárquica. Estos son algunos de los COLLATE que podemos elegir según nuestra necesidad:

| COLLATE | Comportamiento | Uso recomendado |
|---|---|---|
| utf8mb4_general_ci | Ignora acentos y mayúsculas | Comparaciones rápidas y genéricas |
| utf8mb4_unicode_ci | Más preciso con reglas Unicode | Textos multilingües |
| utf8mb4_bin | Sensible a todo (mayúsculas, acentos) | Comparaciones exactas (hash, claves) |
| utf8mb4_spanish_ci | Reglas específicas del español | Bases de datos en español |

- **Por qué utf8mb4 permite emojis y caracteres universales**

Dado que abarca 4 bytes, si solo usamos el “utf8” lo estaremos usando con su base predeterminada que son 3 bytes y no completa lo necesario para obtener caracteres especiales o emojis, podemos tomar en cuenta la siguiente tabla:

| Característica | utf8 (MySQL) | utf8mb4 |
|---|---|---|
| Máximo de bytes | 3 | 4 |
| Soporta emojis | ❌ No | ✅ Sí |
| Unicode completo | ❌ Parcial | ✅ Completo |
| Recomendado por MySQL | ❌ No | ✅ Sí |

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

Básicamente al comparar un texto no distingue entre mayúsculos y minúsculas, le da igual la forma en la que este escrita un carácter.

- **Cómo afectan estas configuraciones a consultas como:**
    - **LIKE**

      **COLLATE determina si LIKE distingue mayúsculas o acentos.**

      Si usas utf8mb4_general_ci:

      "José" LIKE "jose" → Coincide (ignora acentos y mayúsculas).

      Si usas utf8mb4_bin:

      "José" LIKE "jose" → No coincide (compara byte a byte).

    - **ordenamiento ORDER BY**

      **COLLATE define el orden alfabético.**

      En utf8mb4_spanish_ci, "ñ" viene después de "n", como en el diccionario español.

      En utf8mb4_general_ci, "ñ" puede tratarse como "n" (menos preciso).

    - **agrupamiento GROUP BY**

      **COLLATE decide si dos valores se agrupan como iguales.**

      Si usas utf8mb4_general_ci:

      "Kevin" y "kevin" → se agrupan.

      Si usas utf8mb4_bin:

      "Kevin" y "kevin" → se agrupan por separado.

**2. ENGINE = InnoDB**

Investigar:

- Qué es InnoDB

Es el motor de almacenamiento por defecto en MySQL.

- Por qué se usa

Es el encargado de cómo se guardan, actualizan y recuperan los datos dentro de MySQL, que permite transacciones seguras, integridad referencial y alto rendimiento.

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

"ENGINE = InnoDB" se coloca al final de la sentencia "CREATE TABLE", justo después de definir todas las columnas y restricciones, por ejemplo:

```
-- TABLA CLIENTE
CREATE TABLE cliente (
  id_cliente INT AUTO_INCREMENT PRIMARY KEY,
  nombre_cliente VARCHAR(100) NOT NULL,
  cedula VARCHAR(20),
  telefono VARCHAR(30),
  email VARCHAR(100),
  ciudad VARCHAR(60),
  fecha_registro DATE,
  UNIQUE(email)
) ENGINE=InnoDB;
```

- Si es obligatorio o puede omitirse

ENGINE = InnoDB no es obligatorio, porque es el motor por defecto, pero sí es recomendable incluirlo explícitamente para asegurar que la tabla se cree de forma correcta.

- Ventajas de InnoDB (transacciones, FK, ACID)

1. **Transacciones (ACID)**

InnoDB permite ejecutar múltiples operaciones como una sola unidad lógica:

- **A**tomicidad: todo o nada (si algo falla, se revierte).
- **C**onsistencia: mantiene reglas de integridad (como claves foráneas).
- **I**solación: evita interferencias entre transacciones simultáneas.
- **D**urabilidad: los cambios confirmados sobreviven a fallos del sistema

Si algo falla, puedes usar ROLLBACK y no se pierde la información.

2. **Claves foráneas (FOREIGN KEY)**

Permite definir relaciones entre tablas y mantener integridad referencial automáticamente.

3. **Bloqueo a nivel de fila (Row-level locking)**

Permite que múltiples usuarios trabajen sobre la misma tabla sin bloquearse entre sí, ideal para sistemas concurrentes.

- Un ejemplo simple indicando su utilidad

```
169     -- Crear tabla de usuarios
170 •   CREATE TABLE usuarios (
171       id INT PRIMARY KEY,
172       nombre VARCHAR(100)
173     ) ENGINE=InnoDB;
174
175     -- Crear tabla de publicaciones con clave foránea
176 •   CREATE TABLE publicaciones (
177       id INT PRIMARY KEY,
178       usuario_id INT,
179       contenido TEXT,
180       FOREIGN KEY (usuario_id) REFERENCES usuarios(id)
181     ) ENGINE=InnoDB;
182
183     -- Transacción segura
184 •   START TRANSACTION;
185 •   INSERT INTO usuarios VALUES (1, 'Kevin');
186 •   INSERT INTO publicaciones VALUES (1, 1, '¡Hola mundo!');
187 •   COMMIT;
```

Si falla la segunda inserción, la primera también se revierte. Esto garantiza consistencia y seguridad.

**3. Datos Huérfanos**

Investigar:

- Qué significa un dato huérfano

Un dato huérfano es un registro en una tabla hija que hace referencia a un registro inexistente en la tabla padre.

- Es decir, tiene una clave foránea (foreign key) que apunta a un valor que ya no existe en la tabla relacionada.

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

Sucede cuando:

- Se elimina un registro padre (por ejemplo, un paciente),

- Pero no se eliminan o actualizan los registros hijos (por ejemplo, sus citas),
- Y no hay restricciones que lo impidan o controlen.

- Por qué es un problema en bases de datos relacionales

Rompe la integridad referencial: los datos ya no tienen sentido lógico.

- Causa errores en consultas: joins, búsquedas o reportes pueden fallar o devolver resultados incompletos.
- Dificulta el mantenimiento: aparecen registros inútiles o inconsistentes.
- Afecta la lógica de negocio: por ejemplo, citas sin paciente, pedidos sin cliente, etc.

- Cómo se evita

1. Usando claves foráneas (FOREIGN KEY) con restricciones

- ON DELETE CASCADE: elimina automáticamente los hijos.
- ON DELETE SET NULL: pone la clave foránea en NULL.
- ON DELETE RESTRICT o NO ACTION: impide eliminar el padre si hay hijos.

2. Validando manualmente en la aplicación

- Antes de eliminar un registro padre, verificar si tiene hijos relacionados.
- Eliminar o reasignar los hijos primero.

3. Usando motores como InnoDB

- Solo motores como InnoDB permiten claves foráneas y restricciones automáticas.
- Otros como MyISAM no las aplican, lo que facilita que ocurran datos huérfanos.

**4. Integridad Referencial**

- Qué es

Es un principio de las bases de datos relacionales que asegura que los valores de una clave foránea en una tabla hija coincidan con valores existentes en la tabla padre.

- Ejemplo: Si Pedidos.id_cliente apunta a Clientes.id_cliente, la integridad referencial garantiza que ese cliente exista.

- Por qué es importante

Evita datos huérfanos: registros que hacen referencia a datos inexistentes.

- Mantiene la coherencia lógica entre tablas relacionadas.
- Facilita la navegación y consulta entre entidades conectadas.
- Permite automatizar reglas de negocio como eliminar o actualizar en cascada.

Sin integridad referencial, podrías tener pedidos sin cliente, matrículas sin estudiante, o citas sin paciente.

- Acciones referenciales:

| Acción referencial | Comportamiento al eliminar o actualizar el padre | Requiere configuración especial |
|---|---|---|
| `CASCADE` | Propaga la eliminación o actualización al hijo | ✅ Ideal para limpieza automática |
| `SET NULL` | Pone la clave foránea en `NULL` | ✅ La columna debe permitir `NULL` |
| `RESTRICT` | Impide la acción si hay hijos relacionados | ✅ Protege la integridad manualmente |
| `NO ACTION` | Igual que `RESTRICT` en MySQL | ✅ Verifica al final de la transacción |

  - **CASCADE**

```
19      -- Tabla dependiente (Hija)
20      CREATE TABLE Pedidos (
21          id_pedido INT PRIMARY KEY,
22          id_cliente INT,
23          producto VARCHAR(50),
24          cantidad INT,
25          FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
26              ON DELETE CASCADE
27              ON UPDATE CASCADE
28      );
29      INSERT INTO Pedidos (id_pedido, id_cliente, producto, cantidad)
30      VALUES
31      (101, 1, 'Laptop', 1),
32      (102, 1, 'Mouse', 2),
33      (103, 2, 'Teclado', 1),
34      (104, 3, 'Monitor', 2),
35      (105, 4, 'Impresora', 1),
36      (106, 5, 'Tablet', 3),
37      (107, 5, 'Auriculares', 2);
38      select * from Pedidos;
```

  - **RESTRICT**

```
--
CREATE TABLE Pedidos2 (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
        ON DELETE RESTRICT
);
-- Verifica existencia de clientes
SELECT * FROM Clientes;
-- Inserta sin especificar id_pedido (deja que AUTO_INCREMENT lo maneje)
INSERT INTO Pedidos2 (id_cliente) VALUES
(1), (1), (2), (3), (4), (5);
-- Verifica los datos
SELECT * FROM Pedidos2;
```

- **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
);
INSERT INTO Pedidos3 (id_pedido, id_cliente) VALUES
(301, 1),
(302, 1),
(303, 2),
(304, 3),
(305, 4),
(306, 5),
(307, 5);
SELECT * FROM Pedidos3;
```

- **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
);
INSERT INTO Pedidos4 (id_pedido, id_cliente) VALUES
(401, 1),
(402, 1),
(403, 2),
(404, 3),
(405, 4),
(406, 5),
(407, 5);
select * from Pedidos4;
```

Sale un error dado que no permite insertar default

- **NO ACTION**

```sql
CREATE TABLE Pedidos5 (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
        ON DELETE NO ACTION
);
INSERT INTO Pedidos5 (id_pedido, id_cliente) VALUES
(501, 1),
(502, 1),
(503, 2),
(504, 3),
(505, 4),
(506, 5),
(507, 5);
select * from Pedidos5;
```

**5. INDEX**

Investigar:

- Para qué se utiliza un índice

Un índice en una base de datos es una estructura auxiliar que permite acceder rápidamente a los datos sin tener que recorrer toda la tabla.

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

**Consultas lentas**: el motor debe hacer un escaneo completo de la tabla (full table scan)

**Carga excesiva del CPU y disco**: especialmente en operaciones frecuentes o concurrentes

**Bloqueos y cuellos de botella**: en sistemas multiusuario o con alta concurrencia

**Mal rendimiento en joins**: especialmente si se unen tablas sin índices en las claves

- Qué tipos de índices existen

Los tipos de índices lo resumiremos en la siguiente tabla para mejor entendimiento, la cual va de la siguiente manera:

| Tipo de índice | Descripción breve |
|---|---|
| **B-Tree** | Árbol balanceado, ideal para búsquedas por rango y ordenadas ( `<` , `>` , `BETWEEN` ) |
| **Hash** | Rápido en búsquedas exactas ( `=` ), no sirve para rangos |
| **Clustered** | Ordena físicamente los datos de la tabla según el índice |
| **Non-clustered** | Índice separado que apunta a las filas, útil para múltiples búsquedas |
| **Full-text** | Optimizado para búsquedas de texto en columnas grandes |
| **Composite** | Índice sobre varias columnas, útil en filtros combinados |

- Cómo ayuda a mejorar tiempos de consulta

Reduce el número de registros que se deben escanear

Optimiza el plan de ejecución del motor SQL

Evita operaciones costosas como TABLE SCAN

Permite usar estructuras como árboles o hashes para localizar datos en milisegundos

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

Usa herramientas como EXPLAIN o EXPLAIN ANALYZE para ver el plan de ejecución

Observa si hay scans completos en columnas que se usan en filtros o joins

Considera índices en columnas usadas en WHERE, JOIN, GROUP BY, ORDER BY

**PRACTICA**

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

```
3   CREATE DATABASE prueba_indices;
4   USE prueba_indices;
5
6   CREATE TABLE clientes (
7       id INT AUTO_INCREMENT PRIMARY KEY,
8       nombre VARCHAR(50),
9       email VARCHAR(100),
10      ciudad VARCHAR(50)
11  );
12  INSERT INTO clientes (nombre, email, ciudad)
13  VALUES
14  ("Cliente_1", "cliente1@correo.com", "Ciudad_1"),
15  ("Cliente_2", "cliente2@correo.com", "Ciudad_2"),
16  ("Cliente_3", "cliente3@correo.com", "Ciudad_3"),
17  ("Cliente_7890", "cliente7890@correo.com", "Ciudad_90"),
18  ("Cliente_9999", "cliente9999@correo.com", "Ciudad_99");
```

Result Grid | Filter Rows: | Edit:

| id | nombre | email | ciudad |
|---|---|---|---|
| 1 | Cliente_1 | cliente1@correo.com | Ciudad_1 |
| 2 | Cliente_2 | cliente2@correo.com | Ciudad_2 |
| 3 | Cliente_3 | cliente3@correo.com | Ciudad_3 |
| 4 | Cliente_7890 | cliente7890@correo.com | Ciudad_90 |
| 5 | Cliente_9999 | cliente9999@correo.com | Ciudad_99 |
| NULL | NULL | NULL | NULL |

Paso 2: Insertar muchos registros de prueba

```
24  CREATE PROCEDURE insertar_clientes()
25  BEGIN
26      DECLARE i INT DEFAULT 1;
27      WHILE i <= 1000 DO
28          INSERT INTO clientes (nombre, email, ciudad)
29          VALUES (
30              CONCAT("Cliente_", i),
31              CONCAT("cliente", i, "@correo.com"),
32              CONCAT("Ciudad_", MOD(i, 100))
33          );
34          SET i = i + 1;
35      END WHILE;
36  END$$
37
38  DELIMITER ;
39
40  CALL insertar_clientes();
```

Result Grid | Filter Rows: | Edit:

| id | nombre | email | ciudad |
|---|---|---|---|
| 1 | Cliente_1 | cliente1@correo.com | Ciudad_1 |
| 2 | Cliente_2 | cliente2@correo.com | Ciudad_2 |
| 3 | Cliente_3 | cliente3@correo.com | Ciudad_3 |
| 4 | Cliente_7890 | cliente7890@correo.com | Ciudad_90 |
| 5 | Cliente_9999 | cliente9999@correo.com | Ciudad_99 |
| 6 | Cliente_1 | cliente1@correo.com | Ciudad_1 |
| 7 | Cliente_2 | cliente2@correo.com | Ciudad_2 |

Result Grid | Filter Rows: | Edit:

| id | nombre | email | ciudad |
|---|---|---|---|
| 994 | Cliente_989 | cliente989@correo.com | Ciudad_89 |
| 995 | Cliente_990 | cliente990@correo.com | Ciudad_90 |
| 996 | Cliente_991 | cliente991@correo.com | Ciudad_91 |
| 997 | Cliente_992 | cliente992@correo.com | Ciudad_92 |
| 998 | Cliente_993 | cliente993@correo.com | Ciudad_93 |
| 999 | Cliente_994 | cliente994@correo.com | Ciudad_94 |
| 1000 | Cliente_995 | cliente995@correo.com | Ciudad_95 |

Paso 3: Medir tiempo de consulta sin índice

```
42      #Activar medicion de tiempo
43  SET PROFILING = 1;
44
45      #Consulta sin indice
46  SELECT * FROM clientes WHERE email = "cliente7890@correo.com";
47
48      #Ver tiempo de ejecucion
49  SHOW PROFILES;
50
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Query_ID | Duration | Query |
|---|---|---|
| 1 | 0.00488000 | SHOW WARNINGS |
| 2 | 0.00013250 | SET PROFILING = 1 |
| 3 | 0.00007700 | SHOW WARNINGS |
| 4 | 0.01560025 | SELECT * FROM clientes WHERE email = "client... |

Paso 4: Crear un índice en la columna

```
51 ●    CREATE INDEX idx_email ON clientes(email);
```

Paso 5: Medir tiempo de consulta con índice

```
53      #Repetir la misma consulta
54 ●    SELECT * FROM clientes WHERE email = "cliente7890@correo.com";
55
56      #Ver nuevo tiempo de ejecucion
57 ●    SHOW PROFILES;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Query_ID | Duration | Query |
|---|---|---|
| 1 | 0.00488000 | SHOW WARNINGS |
| 2 | 0.00013250 | SET PROFILING = 1 |
| 3 | 0.00007700 | SHOW WARNINGS |
| 4 | 0.01560025 | SELECT * FROM clientes WHERE email = "client... |
| 5 | 0.17560875 | CREATE INDEX idx_email ON clientes(email) |
| 6 | 0.00186400 | SHOW FULL COLUMNS FROM `prueba_indices`... |

**6. Declaraciones en procedimientos**

Investigar:

- Qué es DECLARE i INT DEFAULT 1;

Esta instrucción declara una variable local llamada i de tipo entero (INT) y le asigna un valor inicial de 1. Es común en bucles como WHILE o LOOP.

- DECLARE: palabra clave para definir variables locales
- i: nombre de la variable
- INT: tipo de dato entero
- DEFAULT 1: valor inicial asignado automáticamente

Ejemplo de estructura den MySQL:

```
DECLARE i INT DEFAULT 1;
```

- Qué es DECLARE c INT;

Esta instrucción también declara una variable local llamada c, pero sin valor inicial. Por defecto, su valor será NULL hasta que se le asigne uno con SET.

- En qué parte del procedimiento almacenado se usan

Las variables declaradas con DECLARE solo pueden usarse dentro del cuerpo del procedimiento, y deben colocarse al inicio del bloque BEGIN...END, antes de cualquier otra instrucción como SET, SELECT, IF, WHILE, etc.

Ejemplo de estructura en MySQL:

```
CREATE PROCEDURE ejemplo()
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE total INT;

    SET total = i + 5;
    SELECT total;
END;
```

- Función del SET dentro de un PROCEDURE

SET se utiliza para asignar o actualizar el valor de una variable local dentro del procedimiento.

Por ejemplo: SET i = i + 1;

También se puede usar para asignar valores desde parámetros o resultados de expresiones:

Por ejemplo: SET total = precio * cantidad;

## REQUERIMIENTOS PRÁCTICOS – ACTIVIDAD COMPLETA

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

### PROPONER ENUNCIADOS O PROBLEMAS INDICAR Y REALIZAR

**Enunciado propuesto:**

"Una empresa de venta de llantas desea gestionar sus clientes, productos, ventas y el detalle de cada venta. Se requiere un sistema relacional que permita registrar las transacciones, consultar estadísticas y optimizar el rendimiento de búsqueda."

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

```sql
#1. Crear Base de Datos Relacional

CREATE DATABASE llantasSD;
USE llantasSD;

CREATE TABLE clientes (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    correo VARCHAR(100),
    telefono VARCHAR(20)
);
```

Result Grid | Filter Rows: | Edit:

| id_cliente | nombre | correo | telefono |
|---|---|---|---|
| 1 | Kevin Torres | kevin@mail.com | 0999999999 |
| 2 | Laura Pérez | laura@mail.com | 0988888888 |
| 3 | Carlos Ruiz | carlos@mail.com | 0977777777 |

```sql
CREATE TABLE productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    marca VARCHAR(50),
    precio DECIMAL(10,2),
    stock INT
);
```

Result Grid | Filter Rows: | Edit:

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 1 | Llantas Aro 15 | Michelin | 120.00 | 50 |
| 2 | Llantas Aro 16 | Pirelli | 135.00 | 40 |
| 3 | Llantas Aro 17 | Goodyear | 150.00 | 30 |

```sql
CREATE TABLE ventas (
    id_venta INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    fecha DATE,
    total DECIMAL(10,2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| id_venta | id_cliente | fecha | total |
|---|---|---|---|
| 1 | 1 | 2025-11-01 | 270.00 |

```sql
CREATE TABLE detalle_venta (
    id_detalle INT AUTO_INCREMENT PRIMARY KEY,
    id_venta INT,
    id_producto INT,
    cantidad INT,
    subtotal DECIMAL(10,2),
    FOREIGN KEY (id_venta) REFERENCES ventas(id_venta),
    FOREIGN KEY (id_producto) REFERENCES productos(id_producto)
);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| id_detalle | id_venta | id_producto | cantidad | subtotal |
|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 240.00 |
| 2 | 1 | 2 | 1 | 135.00 |

**2. Realizar 5 Consultas SQL Básicas**

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

1. Consulta con **WHERE**

```sql
# 2. Consultas SQL Básica

#1. WHERE
SELECT * FROM productos WHERE precio > 130;
```

Result Grid | Filter Rows: | Edit:

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 2 | Llantas Aro 16 | Pirelli | 135.00 | 40 |
| 3 | Llantas Aro 17 | Goodyear | 150.00 | 30 |

2. Consulta con **LIKE**

```
65      #2. LIKE
66 •    SELECT * FROM clientes WHERE nombre LIKE "K%";
67      #3. ORDER BY
```

Result Grid | Filter Rows: | Edit:

| id_cliente | nombre | correo | telefono |
|---|---|---|---|
| 1 | Kevin Torres | kevin@mail.com | 0999999999 |

3. Consulta con **ORDER BY**

```
67      #3. ORDER BY
68 •    SELECT * FROM ventas ORDER BY fecha DESC;
69      #4. GROUP BY
70 •    SELECT id_cliente, COUNT(*) AS total_vent
```

Result Grid | Filter Rows: | Edit:

| id_venta | id_cliente | fecha | total |
|---|---|---|---|
| 1 | 1 | 2025-11-01 | 270.00 |

4. Consulta con **GROUP BY**

```
69      #4. GROUP BY
70 •    SELECT id_cliente, COUNT(*) AS total_ventas FROM ventas GROUP BY id_cliente;
71      #5. Función agregada
72 •    SELECT AVG(precio) AS promedio_precio FROM productos;
73
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| id_cliente | total_ventas |
|---|---|
| 1 | 1 |

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

```
71      #5. Función agregada
72 •    SELECT AVG(precio) AS promedio_precio FROM productos;
73
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

| promedio_precio |
|---|
| 135.000000 |

**3. Realizar 3 Subconsultas**

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

1. Subconsulta en **WHERE**

```
74      #3. Subconsulta
75
76      #1. WHERE
77      SELECT * FROM productos WHERE id_producto IN (
78          SELECT id_producto FROM detalle_venta WHERE cantidad >= 2
79      );
80
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| id_producto | nombre | marca | precio | stock |
|---|---|---|---|---|
| 1 | Llantas Aro 15 | Michelin | 120.00 | 50 |

2. Subconsulta en **FROM**

```
81      #2. FROM
82      SELECT AVG(subtotal) AS promedio_subtotal FROM (
83          SELECT subtotal FROM detalle_venta WHERE cantidad >= 1
84      ) AS sub;
85
86      #3. Anidada con función agregada
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| promedio_subtotal |
|---|
| 187.500000 |

3. Subconsulta anidada con funciones agregadas

```
88      SELECT nombre FROM productos WHERE precio = (SELECT MAX(precio) FROM productos
89      );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| nombre |
|---|
| Llantas Aro 17 |

## 4. Crear 3 Índices

```
94    #indice 1: en correo de clientes
95 •  CREATE INDEX idx_correo ON clientes(correo);
96 •  SHOW INDEX FROM clientes;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| clientes | 0 | PRIMARY | 1 | id_cliente | A | 3 | NULL | NULL | | BTREE | | | YES | NULL |
| clientes | 1 | idx_correo | 1 | correo | A | 3 | NULL | NULL | YES | BTREE | | | YES | NULL |

```
98    #indice 2: en fecha de ventas
99 •  CREATE INDEX idx_fecha ON ventas(fecha);
100 •  SHOW INDEX FROM ventas;
101
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ventas | 0 | PRIMARY | 1 | id_venta | A | 1 | NULL | NULL | | BTREE | | | YES | NULL |
| ventas | 1 | id_cliente | 1 | id_cliente | A | 1 | NULL | NULL | YES | BTREE | | | YES | NULL |
| ventas | 1 | idx_fecha | 1 | fecha | A | 0 | NULL | NULL | YES | BTREE | | | YES | NULL |

```
102    #indice 3: en id_producto de detalle_venta
103 •  CREATE INDEX idx_id_producto ON detalle_venta(id_producto);
104 •  SHOW INDEX FROM detalle_venta;
105
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| detalle_venta | 0 | PRIMARY | 1 | id_detalle | A | 2 | NULL | NULL | | BTREE | | | YES | NULL |
| detalle_venta | 1 | id_venta | 1 | id_venta | A | 1 | NULL | NULL | YES | BTREE | | | YES | NULL |
| detalle_venta | 1 | idx_id_producto | 1 | id_producto | A | 2 | NULL | NULL | YES | BTREE | | | YES | NULL |

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

1. Crear el índice en una tabla

   Listo

2. Justificar por qué es necesario

– idx_correo: mejora búsquedas por correo electrónico, útil para validaciones y contacto directo.

– idx_fecha: optimiza consultas por fecha de venta, común en reportes y filtros temporales.

– idx_id_producto: acelera joins y filtros en detalle_venta, especialmente al analizar productos vendidos.

3. Ejecutar una consulta SIN índice (medir tiempo)

```
106 •  SET PROFILING = 1;
107
108 •  SELECT * FROM clientes WHERE correo = "kevin@mail.com";
109 •  SHOW PROFILES;
110
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Query_ID | Duration | Query |
|---|---|---|
| 57 | 0.00103175 | SHOW FULL COLUMNS FROM `llantassd`.`clien... |
| 58 | 0.00089475 | SHOW FULL COLUMNS FROM `llantassd`.`deta... |
| 59 | 0.00078700 | SHOW FULL COLUMNS FROM `llantassd`.`prod... |
| 60 | 0.00080600 | SHOW FULL COLUMNS FROM `llantassd`.`vent... |
| 61 | 0.00092125 | CREATE INDEX idx_correo ON clientes(correo) |
| 62 | 0.00761300 | SHOW INDEX FROM clientes |
| 63 | 0.00123600 | SHOW FULL COLUMNS FROM `llantassd`.`clien... |
| 64 | 0.00119300 | SHOW FULL COLUMNS FROM `llantassd`.`deta... |
| 65 | 0.00083175 | SHOW FULL COLUMNS FROM `llantassd`.`prod... |
| 66 | 0.00076800 | SHOW FULL COLUMNS FROM `llantassd`.`vent... |
| 67 | 0.04424550 | CREATE INDEX idx_fecha ON ventas(fecha) |
| 68 | 0.00701225 | SHOW INDEX FROM ventas |
| 69 | 0.03821100 | CREATE INDEX idx_id_producto ON detalle_ven... |
| 70 | 0.00676700 | SHOW INDEX FROM detalle_venta |

Result 17

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

```
113 •  SELECT * FROM clientes WHERE correo = "kevin@mail.com";
114 •  SHOW PROFILES;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Query_ID | Duration | Query |
|---|---|---|
| 58 | 0.00089475 | SHOW FULL COLUMNS FROM `llantassd`.`deta... |
| 59 | 0.00078700 | SHOW FULL COLUMNS FROM `llantassd`.`prod... |
| 60 | 0.00080600 | SHOW FULL COLUMNS FROM `llantassd`.`vent... |
| 61 | 0.00092125 | CREATE INDEX idx_correo ON clientes(correo) |
| 62 | 0.00761300 | SHOW INDEX FROM clientes |
| 63 | 0.00123600 | SHOW FULL COLUMNS FROM `llantassd`.`clien... |
| 64 | 0.00119300 | SHOW FULL COLUMNS FROM `llantassd`.`deta... |
| 65 | 0.00083175 | SHOW FULL COLUMNS FROM `llantassd`.`prod... |
| 66 | 0.00076800 | SHOW FULL COLUMNS FROM `llantassd`.`vent... |
| 67 | 0.04424550 | CREATE INDEX idx_fecha ON ventas(fecha) |
| 68 | 0.00701225 | SHOW INDEX FROM ventas |
| 69 | 0.03821100 | CREATE INDEX idx_id_producto ON detalle_ven... |
| 70 | 0.00676700 | SHOW INDEX FROM detalle_venta |

lientes 18 Result 19

5. Comparar y explicar la diferencia

- Sin índice: el motor hace un escaneo completo de la tabla (), lo que es lento en tablas grandes.
- Con índice: el motor usa el índice (), accediendo directamente al registro deseado.

**5. Evaluación de Rendimiento con Índices**

**Requerimiento:**

- Seleccionar **2 consultas lentas**
- Ejecutarlas SIN índice y registrar tiempo
- Crear un índice adecuado
- Ejecutar nuevamente y registrar tiempo
- Escribir una comparación final

```
118     #5. Evaluación de Rendimient
119
120     #Consulta lenta sin índice
121 •   SELECT * FROM ventas WHERE fecha = "2025-11-01";
```

Result Grid | Filter Rows: | Edit:

| id_venta | id_cliente | fecha | total |
|---|---|---|---|
| 1 | 1 | 2025-11-01 | 270.00 |

**6. Crear 3 Vistas**

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

1. CREATE VIEW
2. Consulta que la utiliza
3. Explicación del por qué esa vista es útil y en qué escenario se usaría

```
127     #6. Crear 3 Vista
128
129     #Vista 1: ventas por cliente
130 •   CREATE VIEW vista_ventas_cliente AS
131     SELECT c.nombre, v.fecha, v.total
132     FROM clientes c JOIN ventas v ON c.id_cliente = v.id_cliente;
```

```
5 •   SELECT * FROM llantassd.vista_ventas_cliente;
```

Result Grid | Filter Rows: | Export: | Wra

| nombre | fecha | total |
|---|---|---|
| Kevin Torres | 2025-11-01 | 270.00 |

```
134     #Vista 2: stock bajo
135 •   CREATE VIEW vista_stock_bajo AS
136     SELECT * FROM productos WHERE stock < 20;
```

```
10 •    SELECT * FROM llantassd.vista_ventas_cliente;
```

Result Grid | Filter Rows: | Export: | Wrap

| nombre | fecha | total |
|---|---|---|
| Kevin Torres | 2025-11-01 | 270.00 |

```
138     #Vista 3: resumen de ventas
139 •   CREATE VIEW vista_resumen_ventas AS
140     SELECT v.id_venta, COUNT(d.id_producto) AS productos_vendidos, SUM(d.subtotal) AS total
141     FROM ventas v JOIN detalle_venta d ON v.id_venta = d.id_venta
142     GROUP BY v.id_venta;
```

```
10 •    SELECT * FROM llantassd.vista_ventas_cliente;
```

Result Grid | Filter Rows: | Export: | Wrap

| nombre | fecha | total |
|---|---|---|
| Kevin Torres | 2025-11-01 | 270.00 |

**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**
- Ejemplo de salida final

```
146 •  START TRANSACTION;
147
148 •  INSERT INTO ventas (id_cliente, fecha, total) VALUES (2, "2025-11-20", 300.00);
149 •  SAVEPOINT punto1;
150
151 •  UPDATE productos SET stock = stock - 2 WHERE id_producto = 1;
152
153    #ROLLBACK TO punto1; si ocurre error
154 •  COMMIT;
155
156    #Resultado esperado: nueva venta registrada, stock actualizado
157 •  SELECT * FROM ventas WHERE id_cliente = 2;
158 •  SELECT stock FROM productos WHERE id_producto = 1;
159
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content

| id_venta | id_cliente | fecha | total |
|---|---|---|---|
| 2 | 2 | 2025-11-20 | 300.00 |

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| stock |
|---|
| 48 |

**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**

```sql
#8. Crear 3 Procedimientos Almacenado

DELIMITER $$

CREATE PROCEDURE registrar_venta(
    IN p_id_cliente INT,
    IN p_fecha DATE,
    IN p_total DECIMAL(10,2)
)
BEGIN
    DECLARE nueva_venta INT;
    INSERT INTO ventas (id_cliente, fecha, total) VALUES (p_id_cliente, p_fecha, p_total);
    SET nueva_venta = LAST_INSERT_ID();
    SELECT nueva_venta AS id_generado;
END$$

DELIMITER ;

#Ejecutar
CALL registrar_venta(3, "2025-11-20", 180.00);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| id_generado |
|---|
| 3 |

**9. 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 guardan

Las consultas deben organizarse y almacenarse en archivos o scripts bien estructurados, facilitando su reutilización y mantenimiento.

- rendimiento con índices

El uso adecuado de índices mejora significativamente el rendimiento de las consultas, especialmente en búsquedas y filtrados sobre grandes volúmenes de datos.

- utilidad de vistas

Las vistas permiten simplificar el acceso a datos complejos, encapsular lógica de negocio y mejorar la seguridad al limitar el acceso directo a tablas.

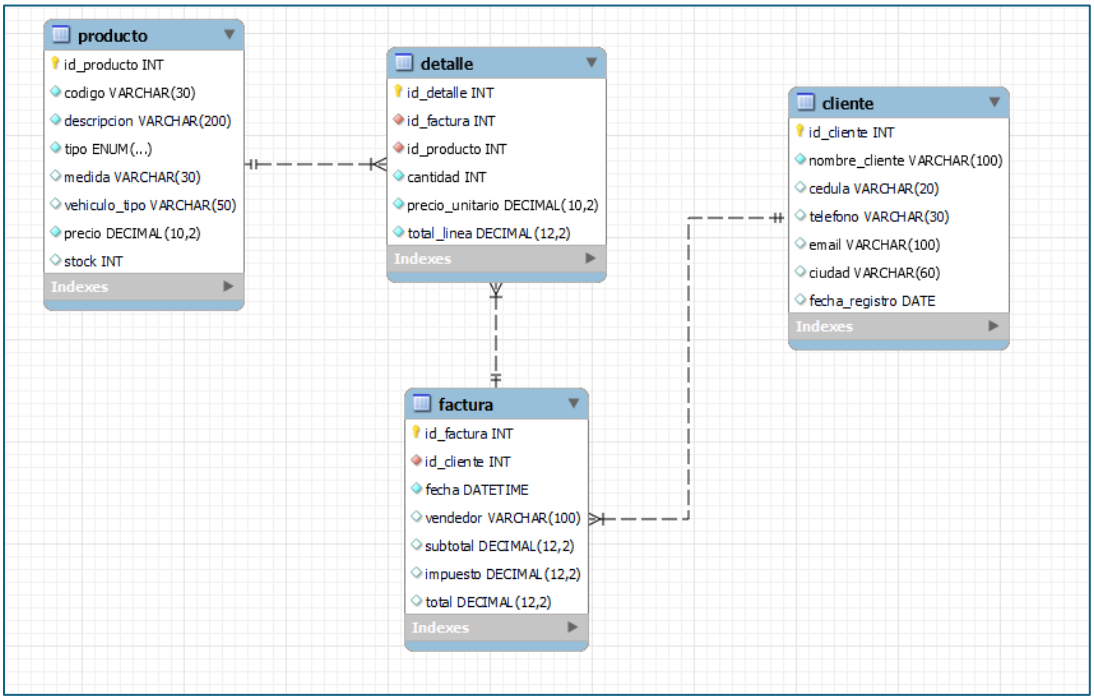
- importancia de transacciones

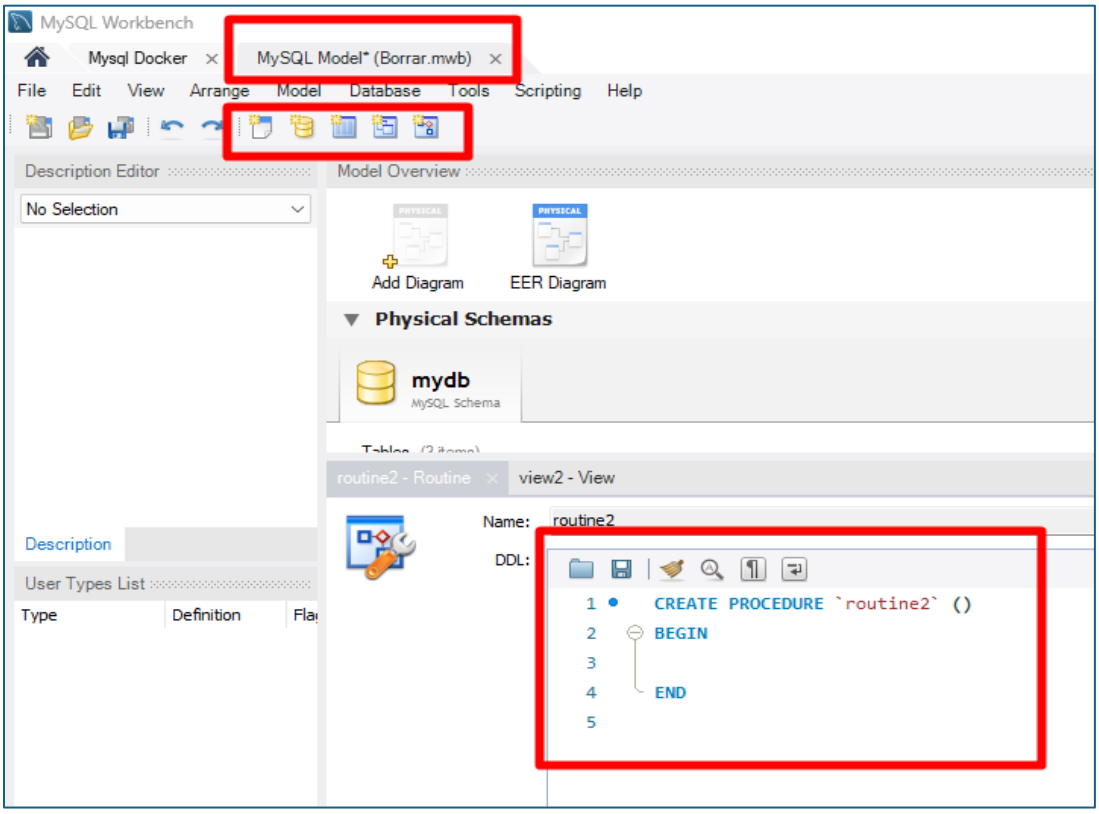
Las transacciones aseguran la integridad de los datos, permitiendo operaciones atómicas y controlando errores mediante commit y rollback

- uso de procedimientos

Los procedimientos encapsulan lógica repetitiva, mejoran la eficiencia del sistema y facilitan la modularidad del código SQL.

- En git hub Subir e ir detallando en Readme




MySQL Workbench
Mysql Docker
MySQL Model* (Borrar.mwb)
File Edit View Arrange Model Database Tools Scripting Help
Description Editor
No Selection
Model Overview
Add Diagram
EER Diagram
Physical Schemas
mydb
MySQL Schema
routine2 - Routine
view2 - View
Name: routine2
DDL:
Description
User Types List
Type
Definition
1 CREATE PROCEDURE `routine2` ()
2 BEGIN
3
4 END
5