Proyecto Final: Sistema de Base de Datos Distribuida para la Gestión de una Tienda de Bicicletas

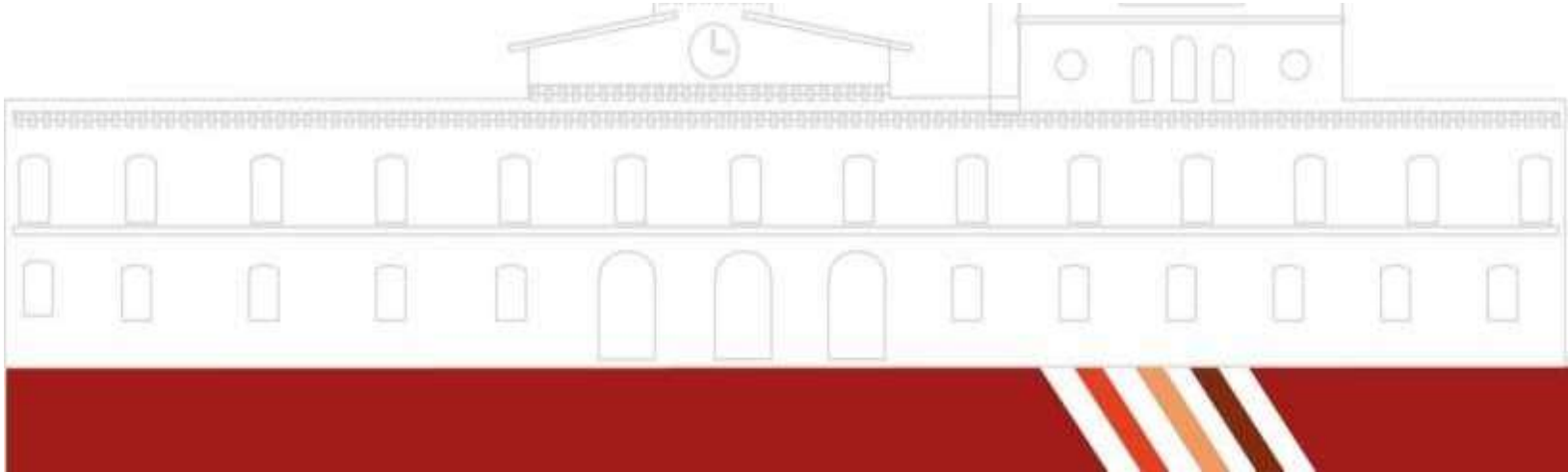
Integrantes:

- Angel Amaya Zumaya
- Cesar Rodríguez García
- Kevin Badillo Olmos
- Rey Vargas Fuentes

Materia: Bases de Datos Distribuidas

Profesor: **EDUARDO CORNEJO VELAZQUEZ**

Fecha: 12 de noviembre de 2025



1. Introducción

1.1. Objetivo del Proyecto

El objetivo de este proyecto es diseñar, implementar y documentar un Sistema de Gestión de Bases de Datos Distribuidas (SGBDD) funcional para un caso de negocio real: una tienda de bicicletas que posee dos áreas de operación principales y físicamente separadas: la Tienda (para ventas) y el Taller (para servicios).

1.2. Descripción del Problema

El negocio presenta una división funcional natural. La Tienda gestiona el inventario de productos (bicicletas, refacciones, accesorios) y la facturación al cliente. El Taller gestiona su propio catálogo de servicios (afinaciones, reparaciones) y el personal técnico (mecánicos).

El desafío principal radica en cómo unificar la facturación. Se debe poder generar un único **Ticket** en la caja de la Tienda que pueda incluir tanto **productos** vendidos del inventario local como **servicios** (mano de obra) realizados en el Taller, consultando ambos catálogos en tiempo real.

1.3. Solución Propuesta

Se propone una arquitectura de base de datos distribuida con dos nodos, implementada en MySQL.

1. **Nodo 1 (Tienda):** Un servidor local (`db_tienda`) que albergará las tablas de `Cliente`, `Producto` y centralizará la facturación en la tabla `Ticket`.
2. **Nodo 2 (Taller):** Un servidor remoto (`db_taller`) que gestionará las tablas de `Mecanico` y `Servicio`.

La conexión entre los nodos se logrará utilizando el motor de almacenamiento **`FEDERATED`** de MySQL. Esto permitirá al Nodo Tienda crear "tablas virtuales" que actúan como un enlace directo al catálogo del Nodo Taller, logrando la **transparencia de ubicación** y permitiendo consultas distribuidas usando SQL estándar.

2. Marco Teórico

2.1. Conceptos Clave del Sistema Distribuido

- **Nodo (Node):** Se refiere a una de las computadoras físicas individuales dentro de la red que almacena una porción de la base de datos. En este proyecto:
  - **Nodo 1 (Tienda):** Computadora local que alberga el sistema de ventas.
  - **Nodo 2 (Taller):** Computadora remota que alberga el sistema de servicios.
- **Esquema (Schema):** En MySQL, es el sinónimo de una base de datos. Es el contenedor lógico que agrupa todas las tablas, vistas y procedimientos de un nodo.
  - **Esquema Tienda:** `db_tienda`
  - **Esquema Taller:** `db_taller`
- **Tupla (Tuple):** Representa una única fila o registro dentro de una tabla. Por ejemplo, una tupla en la tabla `Producto` sería `(1, 'Bicicleta Montaña R29', 'BIC-001', 5999.99, 15)`.

2.2. Fragmentación y Replicación

- **Fragmentación:** Es la técnica de dividir la base de datos lógica en piezas más pequeñas (fragmentos) que se almacenan en diferentes nodos. Este proyecto utiliza **Fragmentación Vertical por Entidad**, donde las tablas se dividen según su función de negocio:
  - **Fragmento 1 (Ventas):** Las tablas `Producto`, `Cliente` y `Ticket` residen en `db_tienda`.
  - **Fragmento 2 (Servicios):** Las tablas `Mecanico` y `Servicio` residen en `db_taller`.
- **Replicación:** Es el proceso de copiar y mantener datos en múltiples nodos. Aunque en este proyecto no se implementó una replicación completa (ej. copiar la tabla `Cliente` a ambos nodos), es un concepto fundamental en sistemas distribuidos para mejorar la disponibilidad y el rendimiento de las consultas.

2.3. El Motor de Almacenamiento `FEDERATED`

`FEDERATED` es un motor de almacenamiento especial de MySQL. A diferencia de motores como `InnoDB` que almacenan datos en el disco duro local, el motor `FEDERATED` **no guarda datos localmente**.

Su función es actuar como un **enlace directo** o un "acceso directo" a una tabla en un servidor de base de datos remoto. El motor almacena únicamente la definición de la tabla y una **cadena de conexión** (usuario, contraseña, IP y nombre de la tabla remota).

Funcionamiento:

Cuando se ejecuta una consulta SQL sobre una tabla FEDERATED (ej. SELECT * FROM servicios_taller_remoto):

1. El MySQL local (Nodo Tienda) recibe la consulta.
2. El motor `FEDERATED` identifica que es una tabla remota.

3. El motor lee la cadena de conexión almacenada.
4. El MySQL local actúa como un **cliente** y envía la consulta por la red al MySQL remoto (Nodo Taller).
5. El Nodo Taller ejecuta la consulta localmente y devuelve los resultados.
6. El motor `FEDERATED` recibe los resultados y los presenta al usuario como si provinieran de una tabla local.

El uso de `FEDERATED` nos permite alcanzar la **Transparencia de Ubicación**, permitiendo al Nodo Tienda unir tablas locales con tablas remotas usando una sintaxis SQL estándar, sin requerir una capa de aplicación intermedia (como PHP o Python) para gestionar las conexiones.

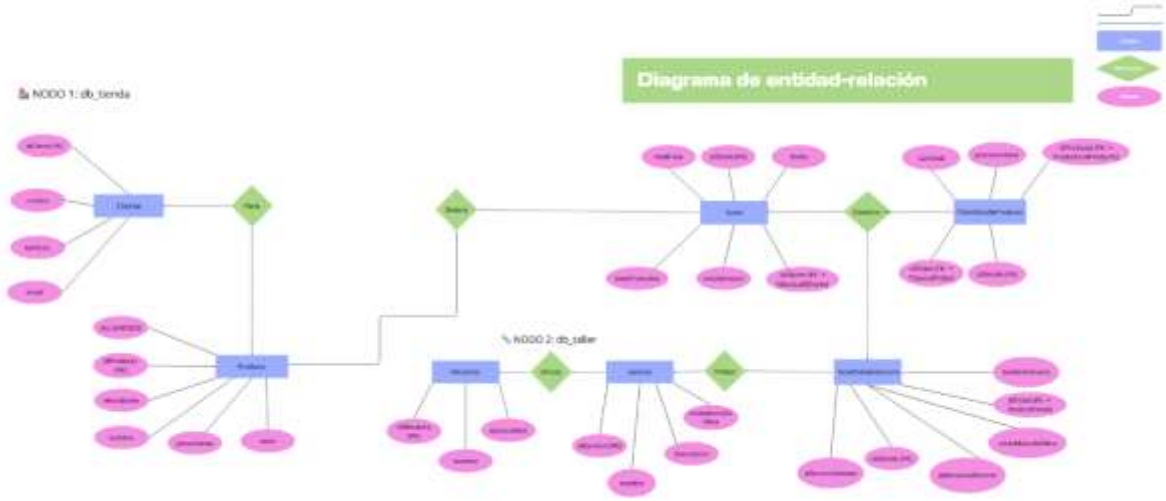
3. Metodología y Diseño de la Arquitectura

3.1. Definición de Nodos y Esquemas Físicos

Basado en el marco teórico, el sistema se divide en los dos nodos físicos y esquemas previamente definidos: `db_tienda` (Nodo 1) y `db_taller` (Nodo 2).

3.2. Modelo Conceptual (Entidad-Relación)

El siguiente diagrama muestra la relación lógica entre las entidades del sistema, identificando los límites de cada nodo.



3.3. Modelo Lógico y Relacional

El modelo conceptual se traduce en los siguientes esquemas relacionales para cada nodo físico. La tabla `TicketDetalleServicio` en el Nodo Tienda actúa como el puente principal, almacenando claves foráneas remotas (`idServicioRemoto`, `idMecanicoRemoto`) que referencian a las tablas en el Nodo Taller.

4. Implementación y Configuración del Sistema

4.1. Creación de Esquemas Locales (Scripts DDL)

El primer paso fue crear las bases de datos y tablas locales en cada nodo respectivo, ejecutando los scripts DDL correspondientes.

- **En el Nodo 1 (Tienda):** Se ejecutó el script para crear el esquema `db_tienda` y sus tablas (`Cliente`, `Producto`, `Ticket`, `TicketDetalleProducto`, `TicketDetalleServicio`).
- **En el Nodo 2 (Taller):** Se ejecutó el script para crear el esquema `db_taller` y sus tablas (`Mecanico`, `Servicio`), incluyendo datos de ejemplo.

4.2. Configuración de Red y Seguridad

Para que los nodos pudieran comunicarse, se realizó la siguiente configuración de red y seguridad.

1. Configuración de Red (IPs):

Ambos nodos se conectaron a la misma red local (WiFi) y se identificaron sus direcciones IP privadas mediante el comando ipconfig en la terminal.

- IP Nodo 1 (Tienda): `10.10.87.38`
- IP Nodo 2 (Taller): `10.10.82.164`

2. Configuración del Firewall:

En ambas computadoras, se configuró el Firewall de Windows Defender para permitir la comunicación entrante en el puerto estándar de MySQL.

- **Acción:** Se creó una nueva "Regla de entrada".
- **Tipo:** Puerto
- **Protocolo:** TCP
- **Puerto:** 3306
- **Acción:** Permitir la conexión
- **Perfil:** Privado
- **Nombre:** `MySQL-Allow-3306`

3. Creación de Usuarios de Red:

Se crearon usuarios específicos en cada nodo para permitir la conexión remota únicamente desde la IP del otro nodo, aplicando el principio de mínimo privilegio.

- En Nodo 1 (Tienda), se ejecutó:

  CREATE USER 'usuario_taller'@'10.10.82.164' IDENTIFIED BY '1234';

  GRANT SELECT ON db_tienda.* TO 'usuario_taller'@'10.10.82.164';

  FLUSH PRIVILEGES;

- En Nodo 2 (Taller), se ejecutó:

  (Este usuario es crítico. Permite a la Tienda leer los catálogos de servicios y mecánicos).

  CREATE USER 'usuario_tienda'@'10.10.87.38' IDENTIFIED BY '1234';

  GRANT SELECT ON db_taller.Servicio TO 'usuario_tienda'@'10.10.87.38';

  GRANT SELECT ON db_taller.Mecanico TO 'usuario_tienda'@'10.10.87.38';

  FLUSH PRIVILEGES;

4.3. Configuración del Motor `FEDERATED`

Para establecer la conexión a nivel de motor de base de datos, se habilitó el Storage Engine `FEDERATED` en el Nodo 1 (Tienda).

- **Acción:** Se editó el archivo de configuración `my.ini` (ubicado en `C:\ProgramData\MySQL\MySQL Server 8.0\`) como administrador.
- **Modificación:** Se añadió la palabra `federated` bajo la sección `[mysqld]`.
- **Finalización:** Se reinició el servicio de Windows `MySQL80` para aplicar los cambios.

4.4. Creación del Enlace Distribuido

Una vez habilitado el motor `FEDERATED` y con los permisos de red establecidos, se crearon las tablas de enlace (federadas) en el Nodo 1.

- **Script (Ejecutado en `db_tienda`):**

```
USE db_tienda;

-- Enlace a la tabla Servicio de César
CREATE TABLE servicios_taller_remoto (
    idServicio INT NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT,
    costoManoDeObra DECIMAL(10, 2) NOT NULL
)
ENGINE=FEDERATED
CONNECTION='mysql://usuario_tienda:1234@10.10.82.164:3306/db_taller/Servicio';

-- Enlace a la tabla Mecanico de César
CREATE TABLE mecanicos_taller_remoto (
    idMecanico INT NOT NULL,
    nombre VARCHAR(150) NOT NULL
)
ENGINE=FEDERATED
CONNECTION='mysql://usuario_tienda:1234@10.10.82.164:3306/db_taller/Mecanico';
```

5. Pruebas y Resultados

5.1. Prueba de Conexión Distribuida

Se realizó una consulta simple de tipo `UNION` para verificar que el Nodo 1 podía consultar simultáneamente su base de datos local y la base de datos remota a través del enlace federado.

- **Consulta:**

```
SELECT nombre AS NombreLocal, precioVenta AS PrecioLocal
```

- `FROM Producto WHERE idProducto = 1`
- `UNION ALL`
- `SELECT nombre AS NombreRemoto, costoManoDeObra AS PrecioRemoto`
- `FROM servicios_taller_remoto WHERE idServicio = 1;`

- **Resultado:** La consulta devolvió exitosamente un producto local y un servicio remoto, validando la conexión distribuida.

5.2. Simulación de Transacción Distribuida (Generación de Ticket)

Esta es la prueba principal del proyecto. Se simuló una venta completa que incluye productos locales y servicios remotos, asegurando la integridad de los datos mediante una transacción.

- **Lógica de la Transacción:**

1. **`START TRANSACTION`**: Inicia el modo seguro.
2. **`INSERT INTO Ticket`**: Se crea la cabecera del ticket.
3. **`SET @ID_TICKET_NUEVO = LAST_INSERT_ID()`**: Se captura el ID del nuevo ticket en una variable.
4. **`INSERT INTO TicketDetalleProducto`**: Se añade un producto local (un casco) al ticket.
5. **`INSERT INTO TicketDetalleServicio (...) SELECT ...`**: **Consulta distribuida.** Se inserta un detalle de servicio en la tabla local, pero los datos (`nombreServicio`, `costoManoDeObra`) se leen en tiempo real desde la tabla remota `servicios_taller_remoto`.
6. **`UPDATE Ticket SET ...`**: Se calculan y actualizan los subtotales y el total final en la cabecera del ticket.
7. **`COMMIT`**: Se confirman todos los cambios de forma permanente.

- **Resultado:** La transacción se completó con éxito. Las consultas finales (`SELECT * FROM Ticket...`) demostraron que todos los datos se registraron correctamente.

5.3. Implementación del Fragmento de Reporte (TICKET_FRAG)

Para cumplir con el requerimiento de generar un fragmento de reporte unificado (como se vio en el diagrama `TICKET_FRAG`), se creó una `VIEW` (Vista) en el Nodo Tienda.

- **Definición de Vista:** Una `VIEW` es una tabla virtual que almacena una consulta `SELECT`. Se utilizó para ocultar la complejidad de las uniones (locales y distribuidas) necesarias para generar el ticket.
- **Acción:** Se creó la vista `VISTA_TICKET_COMPLETO` (ver script en la conversación) que une `Ticket`, `Cliente`, `Producto`, `TicketDetalleProducto`, y crucialmente, las tablas federadas `servicios_taller_remoto` y `mecanicos_taller_remoto`.
- **Consulta de la Vista (Ejecutada en `db_tienda`):**
- `SELECT * FROM VISTA_TICKET_COMPLETO WHERE idTicket = 1;`

- **Resultado:** Esta simple consulta devuelve el ticket completo, uniendo datos de productos locales y servicios remotos, incluyendo `descripcionServicio` y `nombreMecanico` obtenidos directamente del Nodo Taller.

6. Conclusiones

El proyecto se implementó con éxito, logrando construir un sistema de base de datos distribuida funcional que resuelve el problema de negocio planteado.

1. **Arquitectura Exitosa:** La fragmentación vertical por función de negocio (Tienda y Taller) demostró ser una solución lógica y eficiente.
2. **Tecnología Clave:** El motor de almacenamiento `FEDERATED` de MySQL fue la tecnología central que permitió la transparencia de ubicación. Permitió al nodo principal (`db_tienda`) consultar y unificar datos del nodo remoto (`db_taller`) usando comandos SQL estándar, sin necesidad de una capa de aplicación intermedia.
3. **Integridad de Datos:** El uso de `TRANSACTION` en la simulación del ticket garantizó que una venta que involucra múltiples nodos se registre de manera atómica (o todo o nada), previniendo la inconsistencia de datos.
4. **Aplicación Real:** Si bien en un entorno de producción a gran escala se utilizaría una capa de aplicación (API) para gestionar las conexiones, esta implementación a nivel de motor de base de datos es una demostración pura y poderosa de los principios de las bases de datos distribuidas.

El sistema es escalable: se podrían añadir más nodos (ej. un Nodo de Bodega o un Nodo de Contabilidad) utilizando principios similares de fragmentación y federación.

7. Anexos

Anexo A: Script de Creación `db_tienda`

```
CREATE DATABASE IF NOT EXISTS db_tienda DEFAULT CHARACTER SET utf8mb4;
USE db_tienda;

CREATE TABLE Cliente (
  idCliente INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(150) NOT NULL,
  telefono VARCHAR(15),
  email VARCHAR(100)
);

CREATE TABLE Producto (
  idProducto INT PRIMARY KEY AUTO_INCREMENT,
  sku VARCHAR(50) UNIQUE NOT NULL,
  nombre VARCHAR(150) NOT NULL,
  descripcion TEXT,
  precioVenta DECIMAL(10, 2) NOT NULL,
  stock INT NOT NULL DEFAULT 0
);

CREATE TABLE Ticket (
```

```
  idTicket INT PRIMARY KEY AUTO_INCREMENT,
  idCliente INT,
  fecha DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  totalProductos DECIMAL(10, 2) DEFAULT 0,
  totalServicios DECIMAL(10, 2) DEFAULT 0,
  totalFinal DECIMAL(10, 2) DEFAULT 0,
  FOREIGN KEY (idCliente) REFERENCES Cliente(idCliente)
);

CREATE TABLE TicketDetalleProducto (
  idDetalle INT PRIMARY KEY AUTO_INCREMENT,
  idTicket INT NOT NULL,
  idProducto INT NOT NULL,
  cantidad INT NOT NULL,
  precioUnitario DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (idTicket) REFERENCES Ticket(idTicket),
  FOREIGN KEY (idProducto) REFERENCES Producto(idProducto)
);

CREATE TABLE TicketDetalleServicio (
  idDetalle INT PRIMARY KEY AUTO_INCREMENT,
  idTicket INT NOT NULL,
  idServicioRemoto INT NOT NULL,
  idMecanicoRemoto INT NOT NULL,
  nombreServicio VARCHAR(100),
  costoManoDeObra DECIMAL(10, 2),
  FOREIGN KEY (idTicket) REFERENCES Ticket(idTicket)
);

-- Datos de ejemplo
INSERT INTO Cliente (nombre, telefono) VALUES ('Cliente General',
'99999999');
INSERT INTO Producto (sku, nombre, precioVenta, stock) VALUES
('BIC-M-R29', 'Bicicleta de Montaña Rodada 29', 8500.00, 10),
('CAS-NEG-M', 'Casco Negro Mediano', 750.00, 30),
('LLN-R29-X', 'Llanta R29 Todo Terreno', 450.00, 50);
```

Anexo B: Script de Creación `db_taller`

```
CREATE DATABASE IF NOT EXISTS db_taller DEFAULT CHARACTER SET utf8mb4;
USE db_taller;

CREATE TABLE Mecanico (
  idMecanico INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(150) NOT NULL,
  especialidad VARCHAR(100)
);

CREATE TABLE Servicio (
  idServicio INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(100) NOT NULL,
  descripcion TEXT,
  costoManoDeObra DECIMAL(10, 2) NOT NULL
);

-- Datos de ejemplo
```

```
INSERT INTO Mecanico (nombre, especialidad) VALUES
('Raúl Gómez', 'Suspensiones y Frenos'),
('Ana Torres', 'Transmisiones');

INSERT INTO Servicio (nombre, descripcion, costoManoDeObra) VALUES
('Afinación Completa (Mano de Obra)', 'Limpieza y ajuste de componentes,
lubricación.', 450.00),
('Ajuste de Frenos (Mano de Obra)', 'Ajuste de chicotes y balatas.',
150.00),
('Servicio a Suspensión Delantera', 'Cambio de aceite y retenes de
suspensión.', 600.00);
```