Sesión	***18***
Tema	Introducción a MySQL/MariaDB.
Propósito	Dar a conocer a los participantes los fundamentos de instalación, configuración, uso y administración de los SGBD MySql/MariaDB mediante la correspondiente presentación y demostración del docente en una sesión expositiva-demostrativa.
Fecha	C.06.11.2025
Hora	18:30

Introducción a MySQL y MariaDB

📚 Teoría de los Sistemas de Gestión de Bases de Datos (SGBD)

¿Qué es un SGBD?

Un Sistema de Gestión de Bases de Datos es un software que permite crear, mantener y manipular bases de datos de manera eficiente y segura. Actúa como intermediario entre los usuarios/aplicaciones y los datos almacenados.

Responsabilidades principales de un SGBD

- Gestión de almacenamiento: Administra cómo se almacenan físicamente los datos
- Control de concurrencia: Maneja múltiples usuarios simultáneamente
- Seguridad y permisos: Controla el acceso a los datos
- Integridad de datos: Mantiene la consistencia y validez de la información
- Backup y recuperación: Garantiza la disponibilidad de los datos
- Optimización de consultas: Mejora el rendimiento de las operaciones

Clasificación de SGBD

SGBD Relacionales (SQL)

- Oracle: Empresarial, alto rendimiento
- PostgreSQL: Open source, avanzado, extensible
- MS SQL Server: Microsoft, integración con ecosistema .NET
- MySQL: Popular, open source, amplia adopción
- MariaDB: Fork de MySQL, comunidad-driven

SGBD NoSQL

- MongoDB: Documentos JSON, flexible
- SQLite: Embebido, sin servidor
- Cassandra: Columnar, distribuido
- Redis: Clave-valor, en memoria

SGBD en la Nube

- Firebase: Google, tiempo real, serverless
- Supabase: Open source, alternativa a Firebase
- Amazon RDS: Servicio gestionado de AWS
- Google Cloud SQL: MySQL/PostgreSQL gestionado

🛠️ Procedimientos en MySQL/MariaDB

Lo siguiente, es la presentación de un conjunto de operaciones implementadas con código SQL, para un tema de un hipotético sistema de reservas en hotel.

Creación de Base de Datos

```
CREATE DATABASE sistema_hotelero;

USE sistema_hotelero;
```

La instrucción “CREATE DATABASE”, pertenece al subconjunto SQL – DDL (lenguaje de definición de datos) y es practicamente un estándar en todas las versiones SQL usadas por los diferentes SGBD que lo tienen implementado, se usa el símbolo “;” como carácter delimitador de instrucción (marca el final de la instrucción SQL) en los casos MariaDB/MySQL, siendo otro el mecanismo usado en otros GBD. La instrucción “USE” le indica al SGBD con cual de las bases de datos trabajará a partir de ese momento.

NOTA: Las líneas que comienzan con "-- " (incluido el espacio en blanco) se consideran comentarios en el código SQL y no tendrán ningún efecto al ser procesada por el SGBD.

Creación de Tablas

```
-- Tabla de hoteles
CREATE TABLE hoteles (
    hotel_id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(200),
    telefono VARCHAR(20),
    email VARCHAR(100),
    estrellas TINYINT CHECK (estrellas BETWEEN 1 AND 5),
    activo BOOLEAN DEFAULT TRUE,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Tabla de habitaciones
CREATE TABLE habitaciones (
    habitacion_id INT AUTO_INCREMENT PRIMARY KEY,
    hotel_id INT NOT NULL,
    numero_habitacion VARCHAR(10) NOT NULL,
    tipo_habitacion ENUM('Individual', 'Doble', 'Suite', 'Familiar') NOT NULL,
    precio_noche DECIMAL(10,2) NOT NULL,
    capacidad TINYINT NOT NULL,
    descripcion TEXT,
    disponible BOOLEAN DEFAULT TRUE,
    FOREIGN KEY (hotel_id) REFERENCES hoteles(hotel_id) ON DELETE CASCADE,
    UNIQUE KEY unique_habitacion_hotel (hotel_id, numero_habitacion)
);
```

```
-- Tabla de clientes
CREATE TABLE clientes (
    cliente_id INT AUTO_INCREMENT PRIMARY KEY,
    dni VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    telefono VARCHAR(20),
    pais VARCHAR(50),
    fecha_registro DATE DEFAULT (CURDATE())
);

-- Tabla de reservas
CREATE TABLE reservas (
    reserva_id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT NOT NULL,
    habitacion_id INT NOT NULL,
    fecha_entrada DATE NOT NULL,
    fecha_salida DATE NOT NULL,
    estado ENUM('Pendiente', 'Confirmada', 'En curso', 'Finalizada', 'Cancelada') DEFAULT 'Pendiente',
    total DECIMAL(10,2),
    observaciones TEXT,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (cliente_id) REFERENCES clientes(cliente_id) ON DELETE CASCADE,
```

```
    FOREIGN KEY (habitacion_id) REFERENCES habitaciones(habitacion_id) ON
DELETE CASCADE,

    CHECK (fecha_salida > fecha_entrada)

);


-- Tabla de pagos

CREATE TABLE pagos (

    pago_id INT AUTO_INCREMENT PRIMARY KEY,

    reserva_id INT NOT NULL,

    monto DECIMAL(10,2) NOT NULL,

    metodo_pago ENUM('Tarjeta', 'Efectivo', 'Transferencia', 'PayPal') NOT NULL,

    estado ENUM('Pendiente', 'Completado', 'Fallido', 'Reembolsado') DEFAULT
'Pendiente',

    fecha_pago TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    transaccion_id VARCHAR(100),

    FOREIGN KEY (reserva_id) REFERENCES reservas(reserva_id) ON DELETE
CASCADE

);
```

🏨 Ejemplo Detallado: Sistema de Reservas Hoteleras

Índices para Optimización

```
-- Índices para búsquedas frecuentes

CREATE INDEX idx_reservas_fechas ON reservas(fecha_entrada, fecha_salida);

CREATE INDEX idx_habitaciones_disponible ON habitaciones(disponible);

CREATE INDEX idx_clientes_email ON clientes(email);

CREATE INDEX idx_reservas_estado ON reservas(estado);

CREATE INDEX idx_habitaciones_tipo ON habitaciones(tipo_habitacion);
```

Tipos de Datos Definidos por Usuario (ENUM)

```
-- Ya definidos en las tablas, pero podemos agregar más restricciones

ALTER TABLE habitaciones

MODIFY tipo_habitacion ENUM('Individual', 'Doble', 'Suite', 'Familiar', 'Presidencial')
NOT NULL;
```

Vistas Útiles

```
-- Vista de reservas activas

CREATE VIEW vista_reservas_activas AS

SELECT

    r.reserva_id,

    CONCAT(c.nombre, ' ', c.apellido) AS cliente,

    h.nombre AS hotel,

    hab.numero_habitacion,

    r.fecha_entrada,

    r.fecha_salida,

    r.estado,

    r.total

FROM reservas r

JOIN clientes c ON r.cliente_id = c.cliente_id

JOIN habitaciones hab ON r.habitacion_id = hab.habitacion_id

JOIN hoteles h ON hab.hotel_id = h.hotel_id

WHERE r.estado IN ('Confirmada', 'En curso')

ORDER BY r.fecha_entrada;
```

```
-- Vista de disponibilidad de habitaciones
CREATE VIEW vista_disponibilidad AS
SELECT
    h.hotel_id,
    h.nombre AS hotel,
    hab.habitacion_id,
    hab.numero_habitacion,
    hab.tipo_habitacion,
    hab.precio_noche,
    hab.capacidad,
    hab.disponible,
    (SELECT COUNT(*) FROM reservas r
     WHERE r.habitacion_id = hab.habitacion_id
     AND r.estado IN ('Confirmada', 'En curso')
     AND CURDATE() BETWEEN r.fecha_entrada AND r.fecha_salida) AS
ocupada_actualmente
FROM habitaciones hab
JOIN hoteles h ON hab.hotel_id = h.hotel_id
WHERE h.activo = TRUE;
```

Funciones Almacenadas

-- Función para calcular el total de una reserva

```
DELIMITER //
CREATE FUNCTION calcular_total_reserva(
    p_habitacion_id INT,
    p_fecha_entrada DATE,
    p_fecha_salida DATE
)
RETURNS DECIMAL(10,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE v_precio_noche DECIMAL(10,2);
    DECLARE v_num_noches INT;
    DECLARE v_total DECIMAL(10,2);
    -- Obtener precio por noche
    SELECT precio_noche INTO v_precio_noche
    FROM habitaciones
    WHERE habitacion_id = p_habitacion_id;
    -- Calcular número de noches
    SET v_num_noches = DATEDIFF(p_fecha_salida, p_fecha_entrada);
    -- Calcular total
    SET v_total = v_precio_noche * v_num_noches;
    RETURN v_total;
END //
DELIMITER ;
```

```
-- Función para verificar disponibilidad
DELIMITER //
CREATE FUNCTION verificar_disponibilidad(
    p_habitacion_id INT,
    p_fecha_entrada DATE,
    p_fecha_salida DATE
)
RETURNS BOOLEAN
READS SQL DATA
BEGIN
    DECLARE v_count INT;

    SELECT COUNT(*) INTO v_count
    FROM reservas
    WHERE habitacion_id = p_habitacion_id
    AND estado IN ('Confirmada', 'En curso')
    AND (
        (p_fecha_entrada BETWEEN fecha_entrada AND fecha_salida) OR
        (p_fecha_salida BETWEEN fecha_entrada AND fecha_salida) OR
        (fecha_entrada BETWEEN p_fecha_entrada AND p_fecha_salida)
    );

    RETURN (v_count = 0);
END //
DELIMITER ;
```

Procedimientos Almacenados

-- Procedimiento para crear una reserva

```
DELIMITER //
CREATE PROCEDURE crear_reserva(
    IN p_cliente_id INT,
    IN p_habitacion_id INT,
    IN p_fecha_entrada DATE,
    IN p_fecha_salida DATE,
    IN p_observaciones TEXT,
    OUT p_reserva_id INT,
    OUT p_resultado VARCHAR(100)
)
BEGIN
    DECLARE v_disponible BOOLEAN;
    DECLARE v_total DECIMAL(10,2);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET p_resultado = 'Error en la reserva';
    END;
    START TRANSACTION;
    -- Verificar disponibilidad
    SET v_disponible = verificar_disponibilidad(p_habitacion_id, p_fecha_entrada, p_fecha_salida);
    IF v_disponible THEN
        -- Calcular total
        SET v_total = calcular_total_reserva(p_habitacion_id, p_fecha_entrada, p_fecha_salida);
        -- Insertar reserva
        INSERT INTO reservas (cliente_id, habitacion_id, fecha_entrada, fecha_salida, total, observaciones)
        VALUES (p_cliente_id, p_habitacion_id, p_fecha_entrada, p_fecha_salida, v_total, p_observaciones);
        SET p_reserva_id = LAST_INSERT_ID();
```

```
        SET p_resultado = 'Reserva creada exitosamente';
        COMMIT;
    ELSE
        SET p_reserva_id = NULL;
        SET p_resultado = 'Habitación no disponible en las fechas seleccionadas';
        ROLLBACK;
    END IF;
END //
DELIMITER ;
-- Procedimiento para check-in
DELIMITER //
CREATE PROCEDURE realizar_checkin(
    IN p_reserva_id INT,
    OUT p_resultado VARCHAR(100)
)
BEGIN
    DECLARE v_estado VARCHAR(20);
    DECLARE v_fecha_entrada DATE;
    SELECT estado, fecha_entrada INTO v_estado, v_fecha_entrada
    FROM reservas
    WHERE reserva_id = p_reserva_id;
    IF v_estado = 'Confirmada' AND v_fecha_entrada = CURDATE() THEN
        UPDATE reservas
        SET estado = 'En curso'
        WHERE reserva_id = p_reserva_id;
        SET p_resultado = 'Check-in realizado exitosamente';
    ELSE
        SET p_resultado = 'No se puede realizar check-in: reserva no confirmada o fecha incorrecta';
    END IF;
END //
DELIMITER ;
```

Implementación de Triggers

```
-- Trigger para actualizar disponibilidad automáticamente
DELIMITER //
CREATE TRIGGER after_update_reserva
AFTER UPDATE ON reservas
FOR EACH ROW
BEGIN
    -- Cuando una reserva se cancela, liberar la habitación
    IF NEW.estado = 'Cancelada' AND OLD.estado != 'Cancelada' THEN
        UPDATE habitaciones
        SET disponible = TRUE
        WHERE habitacion_id = NEW.habitacion_id;
    END IF;
    -- Cuando una reserva se confirma, marcar habitación como no disponible
    IF NEW.estado = 'Confirmada' AND OLD.estado != 'Confirmada' THEN
        UPDATE habitaciones
        SET disponible = FALSE
        WHERE habitacion_id = NEW.habitacion_id;
    END IF;
    -- Cuando una reserva finaliza, liberar la habitación
    IF NEW.estado = 'Finalizada' AND OLD.estado != 'Finalizada' THEN
        UPDATE habitaciones
        SET disponible = TRUE
        WHERE habitacion_id = NEW.habitacion_id;
    END IF;
END //
DELIMITER ;
```

```
-- Trigger para validar fechas antes de insertar
DELIMITER //
CREATE TRIGGER before_insert_reserva
BEFORE INSERT ON reservas
FOR EACH ROW
BEGIN
    IF NEW.fecha_entrada >= NEW.fecha_salida THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fecha de entrada debe ser anterior a la fecha de salida';
    END IF;

    IF NEW.fecha_entrada < CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No se pueden hacer reservas con fechas pasadas';
    END IF;
END //
DELIMITER ;
```

🔍 Ejemplos de Consultas

Consultas Básicas

```
-- 1. Obtener todas las habitaciones disponibles en un hotel específico
SELECT numero_habitacion, tipo_habitacion, precio_noche, capacidad
FROM habitaciones
WHERE hotel_id = 1
AND disponible = TRUE
ORDER BY precio_noche;

-- 2. Buscar reservas de un cliente
SELECT r.reserva_id, r.fecha_entrada, r.fecha_salida, r.total, r.estado,
       h.nombre AS hotel, hab.numero_habitacion
FROM reservas r
JOIN habitaciones hab ON r.habitacion_id = hab.habitacion_id
JOIN hoteles h ON hab.hotel_id = h.hotel_id
WHERE r.cliente_id = 5
ORDER BY r.fecha_entrada DESC;

-- 3. Calcular ingresos por mes
SELECT
    YEAR(fecha_entrada) AS año,
    MONTH(fecha_entrada) AS mes,
    SUM(total) AS ingresos_totales,
    COUNT(*) AS numero_reservas
FROM reservas
WHERE estado IN ('Finalizada', 'En curso')
```

```
GROUP BY YEAR(fecha_entrada), MONTH(fecha_entrada)
ORDER BY año DESC, mes DESC;
```

Consultas Avanzadas

-- 4. Habitaciones más populares

```
SELECT
    hab.tipo_habitacion,
    COUNT(*) AS total_reservas,
    AVG(r.total) AS promedio_ingresos,
    SUM(r.total) AS ingresos_totales
FROM reservas r
JOIN habitaciones hab ON r.habitacion_id = hab.habitacion_id
WHERE r.estado != 'Cancelada'
GROUP BY hab.tipo_habitacion
ORDER BY total_reservas DESC;
```

-- 5. **Clientes frecuentes**

```
SELECT
    c.cliente_id,
    CONCAT(c.nombre, ' ', c.apellido) AS cliente,
    COUNT(*) AS total_reservas,
    SUM(r.total) AS total_gastado,
    MAX(r.fecha_entrada) AS ultima_reserva
FROM clientes c
JOIN reservas r ON c.cliente_id = r.cliente_id
WHERE r.estado != 'Cancelada'
GROUP BY c.cliente_id
```

```
HAVING total_reservas > 2
ORDER BY total_gastado DESC;


-- 6. Disponibilidad para fechas específicas
SELECT
    h.nombre AS hotel,
    hab.numero_habitacion,
    hab.tipo_habitacion,
    hab.precio_noche,
    hab.capacidad,
    CASE
        WHEN verificar_disponibilidad(hab.habitacion_id, '2024-12-20', '2024-12-25') THEN 'Disponible'
        ELSE 'No disponible'
    END AS estado
FROM habitaciones hab
JOIN hoteles h ON hab.hotel_id = h.hotel_id
WHERE h.activo = TRUE
AND hab.capacidad >= 2
ORDER BY h.nombre, hab.precio_noche;
```

Uso de Procedimientos y Funciones

-- Llamar procedimiento para crear reserva

CALL crear_reserva(1, 5, '2024-12-20', '2024-12-25', 'Cama extra necesaria', @reserva_id, @resultado);

SELECT @reserva_id, @resultado;

-- Verificar disponibilidad usando función

SELECT

habitacion_id,

numero_habitacion,

verificar_disponibilidad(habitacion_id, '2024-12-20', '2024-12-25') AS disponible

FROM habitaciones

WHERE hotel_id = 1;

💡 Consideraciones Finales

Mejores Prácticas

- Índices estratégicos en campos de búsqueda frecuente
- Normalización adecuada para evitar redundancias
- Backups regulares de la base de datos
- Monitorización del rendimiento con EXPLAIN en consultas complejas
- Uso de transacciones para operaciones críticas

Diferencias MySQL vs MariaDB

- MariaDB incluye más motores de almacenamiento
- Mejor optimización de consultas en MariaDB
- Compatibilidad casi total, pero con mejoras en MariaDB
- Comunidad más activa en MariaDB

ANEXOS

📜 **Evolución Histórica de MySQL**
Orígenes y Crecimiento (1995-2008)

1995 - MySQL creado por Michael "Monty" Widenius y David Axmark
→ Objetivo: Base de datos rápida y ligera para aplicaciones web
→ Lema: "MySQL - The world's most popular open source database"

2000 - Se adopta la licencia dual (GPL + comercial)
→ Estrategia que permitió crecimiento rápido

2001 - Versión 3.23 con soporte para transacciones
→ Se convierte en seria alternativa a bases de datos comerciales

2003 - Versión 4.0 con uniones, subconsultas y replicación
→ Adopción masiva en aplicaciones web

2005 - Versión 5.0 con procedimientos almacenados, triggers y vistas
→ Se acerca a la compatibilidad SQL estándar

2008 - MySQL AB es adquirido por Sun Microsystems por $1 billón
→ Momento crucial en la historia del proyecto

La Era Oracle (2010-Presente)

2010 - Oracle adquiere Sun Microsystems (y por ende, MySQL)
→ Preocupación en la comunidad sobre el futuro open source

2010 - Aparece MariaDB como fork liderado por el creador original
→ Monty Widenius crea MariaDB por temor al control de Oracle

2013 - MySQL 5.6 con mejoras en replicación y optimización
→ Oracle continúa desarrollando MySQL activamente

2015 - MySQL 5.7 con JSON, GIS y mejor rendimiento
→ Oracle demuestra compromiso con el producto

2018 - MySQL 8.0 con SQL window functions, CTEs, roles
→ Gran actualización con características modernas

🚀 **Nacimiento de MariaDB**
El "Fork" como Protesta

-- Contexto histórico clave:
-- 2008: Sun compra MySQL AB → Comunidad preocupada pero optimista
-- 2010: Oracle compra Sun → Alarmas en la comunidad open source
-- 2010: Monty Widenius abandona Oracle y crea MariaDB

-- Filosofía de MariaDB:
-- "Ser un reemplazo directo de MySQL, 100% compatible"
-- "Mantener siempre el código abierto y libre"
-- "Desarrollo dirigido por la comunidad"

Evolución de MariaDB

2012 - MariaDB 5.5 → Compatibilidad total con MySQL 5.5
+ Motores de almacenamiento adicionales

2014 - MariaDB 10.0 → Supera a MySQL 5.6
+ Dynamic columns, Cassandra storage engine

2017 - MariaDB 10.3 → Compatibilidad con Oracle PL/SQL
+ Temporal tables, sequences

2020 - MariaDB 10.5 → Mejoras en seguridad y performance
+ System versioned tables mejoradas

2023 - MariaDB 11.0 → Características empresariales avanzadas
+ Mejor optimización de consultas

⚖️ Comparación Detallada: MySQL vs MariaDB

Compatibilidad y Migración

```
-- ✅ Compatibilidad casi total en sintaxis básica
SELECT * FROM usuarios WHERE activo = 1; -- Funciona en ambos

-- 🔄 Diferencias en características avanzadas
-- MySQL 8.0 tiene mejor soporte para Window Functions
SELECT
   nombre,
   salario,
   AVG(salario) OVER (PARTITION BY departamento) as promedio_depto
FROM empleados;

-- MariaDB tiene motores de almacenamiento únicos
CREATE TABLE logs (
   id INT PRIMARY KEY,
   mensaje TEXT
) ENGINE=Aria; -- Solo en MariaDB
```

Tabla Comparativa Completa

Característica	MySQL	MariaDB
Licencia	GPL + Oracle Commercial	Siempre GPL, más permisiva
Desarrollo	Oracle Corporation	Fundación MariaDB + Comunidad
Motores	Almacenamiento InnoDB, MyISAM	InnoDB, Aria, ColumnStore, MyRocks
Rendimiento	Excelente en cargas mixtas	Mejor en lectura intensiva
Replicación	Clásica y GTID	Multi-source, parallel replication
Seguridad	Enterprise firewall (pago)	Firewall incluido en versión community
JSON	Nativo desde 5.7	Compatible pero menos optimizado
GIS	Muy robusto	Bueno pero menos características
Comunidad	Grande pero corporativa	Muy activa y comprometida
Documentación	Excelente y completa	Buena pero menos detallada

Rendimiento y Escalabilidad

```
-- MySQL: Optimizado para cargas mixtas lectura/escritura
-- Ventaja en aplicaciones transaccionales intensivas

-- MariaDB: Excelente en operaciones de lectura
-- Mejor rendimiento en consultas complejas y replicación

-- Ejemplo: Consulta que muestra diferencias de optimización

EXPLAIN
SELECT c.nombre, COUNT(r.reserva_id) as total_reservas
FROM clientes c
LEFT JOIN reservas r ON c.cliente_id = r.cliente_id
WHERE r.fecha_entrada BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY c.cliente_id
HAVING total_reservas > 5;
-- MariaDB suele tener mejor optimización en JOINs complejos
```

Características Únicas por Plataforma
MySQL Exclusivo

```
-- MySQL Enterprise features (pago)
-- Audit Plugin avanzado
-- Thread pooling enterprise
-- Backup en caliente empresarial

-- MySQL 8.0 improvements
WITH RECURSIVE category_path AS (
    SELECT category_id, name, parent_id, name as path
    FROM categories
    WHERE parent_id IS NULL
    UNION ALL
    SELECT c.category_id, c.name, c.parent_id,
           CONCAT(cp.path, ' > ', c.name)
    FROM categories c
    JOIN category_path cp ON c.parent_id = cp.category_id
)
```

```
SELECT * FROM category_path;
```

MariaDB Exclusivo

```
-- Motores de almacenamiento únicos
CREATE TABLE analytics (
    id INT PRIMARY KEY,
    data JSON
) ENGINE=ColumnStore; -- Optimizado para data warehousing

-- Dynamic columns
INSERT INTO productos (nombre, attributes)
VALUES ('Laptop', COLUMN_CREATE(
    'cpu', 'i7',
    'ram', '16GB',
    'storage', '1TB SSD'
));

-- Flashback (recuperación de datos eliminados)
SELECT * FROM tabla
FOR SYSTEM_TIME BETWEEN '2024-01-01 10:00:00' AND '2024-01-01 11:00:00';
```

🎯 Recomendación: ¿Cuál Adoptar y Por Qué?

Escenarios y Recomendaciones Específicas

✅ Elegir MySQL cuando:

-- 1. Entornos corporativos grandes
-- Justificación: Soporte empresarial de Oracle
-- Ejemplo: Bancos, grandes e-commerce

-- 2. Aplicaciones que usan intensivamente JSON
-- Justificación: Mejor implementación y rendimiento
SELECT * FROM ordenes
WHERE JSON_EXTRACT(datos, '$.cliente.pais') = 'México';

-- 3. Sistemas que requieren GIS avanzado
-- Justificación: Funcionalidades espaciales más completas
SELECT * FROM sucursales
WHERE ST_Distance(ubicacion, POINT(-99.1332, 19.4326)) < 10000;

-- 4. Cuando ya existe expertise en MySQL
-- Justificación: Curva de aprendizaje cero

✅ Elegir MariaDB cuando:

-- 1. Proyectos open source puros
-- Justificación: Filosofía 100% open source
-- Ejemplo: Startups, proyectos comunitarios

-- 2. Necesidad de replicación avanzada
-- Justificación: Multi-source replication
CREATE SLAVE conn1, conn2, conn3;
SET GLOBAL master_connection = 'conn1';

-- 3. Entornos con muchas lecturas
-- Justificación: Mejor rendimiento en consultas SELECT
-- Ejemplo: Sitios de contenido, blogs, CMS

-- 4. **Cuando se quiere evitar vendor lock-in**
-- Justificación: Desarrollo comunitario independiente

🏆 Recomendación General

Para la mayoría de casos nuevos: MariaDB

Razones principales:

- Filosofía open source pura - Menor riesgo de cambios restrictivos
- Innovación más rápida - Características nuevas llegan antes
- Comunidad vibrante - Desarrollo orientado a usuarios reales
- Compatibilidad garantizada - Puedes migrar desde MySQL fácilmente
- Rendimiento generalmente mejor - Especialmente en operaciones de lectura

Casos específicos donde MySQL sigue siendo mejor opción:

- Grandes corporaciones que necesitan soporte empresarial de Oracle
- Aplicaciones que dependen fuertemente de JSON nativo
- Proyectos existentes con inversión significativa en MySQL
- Necesidad de características GIS avanzadas

📊 Tendencias del Mercado

2024 - Estadísticas de adopción:
- MySQL: 45% del mercado (gracias a legacy y corporativo)
- MariaDB: 25% y creciendo rápidamente
- PostgreSQL: 30% (otra alternativa fuerte)

Tendencia:

- MariaDB gana terreno en nuevos proyectos
- MySQL mantiene dominio en instalaciones existentes

🔄 Migración entre Sistemas

Migrar de MySQL a MariaDB es generalmente directo:
1. Backup de la base de datos

```
mysqldump -u root -p database > backup.sql
```

```
# 2. Instalar MariaDB
    sudo apt install mariadb-server

# 3. Restaurar backup
    mysql -u root -p database < backup.sql

# La mayoría de aplicaciones funcionan sin cambios
```

🎯 Conclusión Final

- MariaDB representa la evolución natural de MySQL, manteniendo su esencia mientras avanza con innovación abierta. Para proyectos nuevos, especialmente aquellos con fuertes valores open source o necesidades de rendimiento en lectura, MariaDB es la elección recomendada.
- MySQL sigue siendo sólido para entornos empresariales tradicionales y donde las características específicas de Oracle son necesarias.

La belleza de esta dualidad es que puedes comenzar con uno y migrar al otro si tus necesidades cambian, gracias a la alta compatibilidad entre ambos sistemas.

BIBLIOGRAFÍA

- "Fundamentos de Bases de Datos" de Abraham Silberschatz, Henry F. Korth y S. Sudarshan
- "Sistemas de Bases de Datos: un enfoque práctico" de Thomas M. Connolly y Carolyn Begg
- "Desarrollo de Bases de Datos: casos prácticos desde el análisis a la implementación" de Dolores Cuadra, Elena Castro, Ana M. Iglesias.
- "Tecnología y Diseño de Bases de Datos" de Marcos, C. Calero y B. Vela
- https://docs.oracle.com/en/database/