Guía Práctica de MySQL: de Básico a Avanzado

Sintaxis, uso correcto y aclaración de confusiones comunes

Convención en este documento: el símbolo «» denota partes opcionales o nombres de ejemplo, y MAYÚSCULAS indican palabras clave SQL.

0) Recomendaciones clave (mi opción preferida cuando aplica)

- Usa siempre UTF8MB4 y una colación moderna (p. ej., utf8mb4_0900_ai_ci).
- Para dinero, DECIMAL(precision, escala) > FLOAT/DOUBLE (evita errores binarios).
- Evita ENUM/SET para datos vivos; prefiere tablas de referencia + claves foráneas.
- Crea índices donde filtras/ordenas/relacionas; pero evita indexar todo.
- Siempre incluye WHERE en UPDATE/DELETE; si dudas, usa una SELECT previa.
- Prefiere InnoDB, transacciones y claves foráneas. MyISAM es legado.
- Usa nombres explícitos para claves/índices y convención consistente (snake_case).
- Para cadenas largas usa TEXT; para JSON usa tipo JSON (MySQL 5.7+).

1) Tipos de datos esenciales

Enteros: TINYINT/SMALLINT/INT/BIGINT. Autoincrementos para PK pequeñas; BIGINT si esperas muchos registros.

Decimales: DECIMAL(p,s) para cantidades exactas (p.ej., dinero).

Texto: VARCHAR(n) para longitudes variables; CHAR(n) para longitudes fijas; TEXT para bloques grandes.

Fechas: DATE, DATETIME, TIMESTAMP; TIMESTAMP guarda zona UTC interna. Evita guardarlas como texto.

Binarios: BLOB; JSON: tipo nativo con funciones JSON_.

2) DDL — Definición de estructuras

CREATE DATABASE

```
CREATE DATABASE IF NOT EXISTS mi_app
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;
```

CREATE TABLE (columnas, tipos, restricciones).

```
CREATE TABLE usuarios (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  nombre VARCHAR(120) NOT NULL,
  rol_id TINYINT UNSIGNED NOT NULL,
  saldo DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  activo TINYINT(1) NOT NULL DEFAULT 1,
  creado_en DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  actualizado_en DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  CONSTRAINT fk_usuarios_roles FOREIGN KEY (rol_id) REFERENCES roles(id)
```

```
) ENGINE=InnoDB;
```

ALTER TABLE (agregar/modificar/eliminar).

```
-- Agregar columna
ALTER TABLE usuarios ADD telefono VARCHAR(30) NULL AFTER nombre;

-- Modificar tipo (MODIFY) o cambiar nombre (CHANGE)
ALTER TABLE usuarios MODIFY telefono VARCHAR(40) NULL;
ALTER TABLE usuarios CHANGE telefono tel VARCHAR(40) NULL;

-- Agregar índice/único
ALTER TABLE usuarios ADD UNIQUE idx_usuarios_email (email);

-- Eliminar columna/índice
ALTER TABLE usuarios DROP COLUMN tel;
ALTER TABLE usuarios DROP INDEX idx_usuarios_email;
```

DROP TABLE elimina la tabla; TRUNCATE borra filas sin registro histórico (rápido, resetea AUTO_INCREMENT).

CHECK funciona en MySQL 8.0+; para reglas complejas, usa triggers o lógica de aplicación.

3) DML — Inserción de datos

INSERT — inserta nuevas filas.

```
-- Forma explícita (recomendada): especifica columnas
INSERT INTO usuarios (email, nombre, rol_id, saldo)
VALUES ('ana@example.com', 'Ana', 2, 100.00);

-- Inserción múltiple
INSERT INTO usuarios (email, nombre, rol_id, saldo)
VALUES
  ('bob@example.com', 'Bob', 2, 0.00),
  ('caro@example.com', 'Caro', 3, 50.00);

-- INSERT ... SET (menos portátil, pero válido en MySQL)
INSERT INTO usuarios SET email='dani@example.com', nombre='Dani', rol_id=2, saldo=0.00;

-- INSERT ... SELECT (copia/transforma datos)
INSERT INTO usuarios_hist (usuario_id, email, nombre, saldo)
SELECT id, email, nombre, saldo FROM usuarios WHERE saldo > 0;

-- Clave duplicada: actualiza en conflicto (UPSERT)
INSERT INTO usuarios (email, nombre, rol_id, saldo)
VALUES ('ana@example.com', 'Ana', 2, 100.00)
ON DUPLICATE KEY UPDATE
  nombre = VALUES(nombre),
  rol_id = VALUES(rol_id),
  saldo  = VALUES(saldo);
```

Confusiones comunes:

- Especifica columnas: el orden por defecto de la tabla puede cambiar y romper tu inserción.
- NULL ≠ cadena vacía: usa NULL para 'desconocido/no aplica'.
- AUTO_INCREMENT: no lo pases salvo que necesites un valor concreto.
- ON DUPLICATE KEY UPDATE requiere una clave UNIQUE/PK que colisione.

4) SELECT — Lectura y consulta

```
-- Selección básica
SELECT id, email, nombre FROM usuarios;

-- Filtros
SELECT * FROM usuarios
WHERE activo = 1 AND saldo >= 100
  AND email LIKE '%.com'    -- usa índice si no hay comodín inicial
ORDER BY saldo DESC
LIMIT 10 OFFSET 0;

-- DISTINCT, agregaciones y agrupación
SELECT rol_id, COUNT(*) AS total, SUM(saldo) AS saldo_total
FROM usuarios
GROUP BY rol_id
HAVING COUNT(*) > 1;

-- JOINs (relaciones)
SELECT u.id, u.email, r.nombre AS rol
FROM usuarios u
JOIN roles r ON r.id = u.rol_id;          -- INNER JOIN
-- LEFT JOIN conserva usuarios sin rol
SELECT u.id, r.nombre
FROM usuarios u LEFT JOIN roles r ON r.id = u.rol_id;

-- Subconsultas y CTEs (MySQL 8+)
WITH top_activos AS (
  SELECT id FROM usuarios WHERE activo=1 ORDER BY saldo DESC LIMIT 5
)
SELECT * FROM usuarios WHERE id IN (SELECT id FROM top_activos);

-- Ventanas (window functions)
SELECT id, nombre, saldo,
       RANK() OVER (ORDER BY saldo DESC) AS rank_saldo
FROM usuarios;
```

Notas:

- WHERE filtra antes de agrupar; HAVING filtra el resultado de GROUP BY.
- LIKE 'abc%' usa índice; LIKE '%abc' no (requiere escaneo).
- Evita SELECT * en producción: define columnas para estabilidad y rendimiento.

5) UPDATE — Modificación de filas

```
-- Siempre especifica WHERE
UPDATE usuarios
SET saldo = saldo + 25, actualizado_en = NOW()
WHERE id = 10;

-- UPDATE con JOIN
UPDATE usuarios u
JOIN roles r ON r.id = u.rol_id
SET u.activo = 0
WHERE r.nombre = 'suspendido';
```

Buenas prácticas:

- Prueba primero con SELECT para confirmar las filas a afectar.
- Considera ORDER BY + LIMIT en lotes para cambios masivos.

■ Peligro: sin WHERE actualizarás toda la tabla. Activa SQL_SAFE_UPDATES en clientes interactivos si te ayuda.

6) DELETE / TRUNCATE — Eliminación

```
-- DELETE (respeta transacciones y puede tener WHERE)
DELETE FROM usuarios WHERE activo = 0;

-- DELETE con JOIN
DELETE u FROM usuarios u
JOIN roles r ON r.id = u.rol_id
WHERE r.nombre = 'borrado';

-- TRUNCATE: borra todo, rápido, reinicia AUTO_INCREMENT, no tiene WHERE
TRUNCATE TABLE usuarios_tmp;
```

Usa TRUNCATE solo para tablas temporales o que puedes reconstruir fácilmente.

7) Transacciones (TCL) y aislamiento

```
START TRANSACTION;
  UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;
  UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;
COMMIT;  -- o ROLLBACK;

-- Puntos de guardado
START TRANSACTION;
  UPDATE ...;
  SAVEPOINT paso1;
  UPDATE ...;
  ROLLBACK TO SAVEPOINT paso1;
COMMIT;

-- Nivel de aislamiento (elige según caso de uso; por defecto REPEATABLE READ en InnoDB)
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Mi opción recomendada: READ COMMITTED para apps OLTP típicas; SERIALIZABLE solo si realmente lo necesitas (más bloqueos).

8) DCL — Usuarios y privilegios

```
-- Crear usuario (MySQL 8+)
CREATE USER 'app'@'%' IDENTIFIED BY 'contraseña_segura';
GRANT SELECT, INSERT, UPDATE, DELETE ON mi_app.* TO 'app'@'%';
FLUSH PRIVILEGES;
-- Ver permisos
SHOW GRANTS FOR 'app'@'%';
```

Principio de mínimo privilegio: otorga solo lo necesario; evita usar root en apps.

9) Índices y rendimiento

```
-- Índice simple y compuesto
CREATE INDEX idx_usuarios_activo ON usuarios (activo);
```

```
CREATE INDEX idx_usuarios_rol_saldo ON usuarios (rol_id, saldo);

-- Único y texto completo
CREATE UNIQUE INDEX idx_unq_email ON usuarios (email);
CREATE FULLTEXT INDEX idx_ft_nombre ON usuarios (nombre);  -- MATCH() AGAINST()

-- Prefijos (en textos largos)
CREATE INDEX idx_email_pref ON usuarios (email(20));
```

- Orden en índice compuesto importa: (a,b) sirve para consultas por a o por a+b, no para solo b.
- EXPLAIN planifica tu consulta; ANALYZE TABLE/EXPLAIN ANALYZE (8.0.18+) para tiempos reales.
- Evita funciones en columnas indexadas en el WHERE (p.ej., LOWER(col)).

10) Vistas

```
CREATE VIEW v_usuarios_activos AS
SELECT id, email, nombre, saldo FROM usuarios WHERE activo = 1;

-- Actualizables si derivan de una sola tabla sin agregaciones/UNION/etc.
UPDATE v_usuarios_activos SET saldo = saldo + 10 WHERE id = 5;
```

11) Procedimientos, funciones y triggers

```
DELIMITER //
CREATE PROCEDURE subir_saldo(IN p_id BIGINT, IN p_monto DECIMAL(12,2))
BEGIN
  UPDATE usuarios SET saldo = saldo + p_monto WHERE id = p_id;
END //
DELIMITER ;

-- Función
DELIMITER //
CREATE FUNCTION iva(monto DECIMAL(12,2)) RETURNS DECIMAL(12,2)
DETERMINISTIC
BEGIN
  RETURN monto * 0.16;
END //
DELIMITER ;

-- Trigger
DELIMITER //
CREATE TRIGGER usuarios_bu BEFORE UPDATE ON usuarios
FOR EACH ROW
BEGIN
  IF NEW.saldo < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo negativo no permitido';
  END IF;
END //
DELIMITER ;
```

Recuerda cambiar el DELIMITER en clientes como mysql para definir cuerpos de rutina sin conflictos con ';'.

12) Importación y exportación

```
-- Rápido (requiere permisos y configuración)
LOAD DATA INFILE '/ruta/datos.csv'
```

```
INTO TABLE usuarios
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES (email, nombre, rol_id, saldo);

-- Exportar
SELECT * FROM usuarios
INTO OUTFILE '/ruta/usuarios.csv'
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';

-- Copias de seguridad
mysqldump -u usuario -p --routines --triggers mi_app > backup.sql
mysql -u usuario -p mi_app < backup.sql
```

13) Cheats de sintaxis rápida (por verbo y forma)

```
INSERT: INSERT INTO <tabla> (<cols>) VALUES (<valores>);
INSERT múltiples: INSERT INTO <tabla> (<cols>) VALUES (...), (...);
INSERT SET (MySQL): INSERT INTO <tabla> SET col=val, ...;
UPSERT: INSERT ... ON DUPLICATE KEY UPDATE col=VALUES(col), ...;

SELECT básico: SELECT <cols> FROM <tabla> [WHERE ...] [ORDER BY ...] [LIMIT n OFFSET m];
DISTINCT: SELECT DISTINCT <cols> FROM <tabla>;
JOIN: SELECT ... FROM a JOIN b ON b.id=a.b_id;
LEFT JOIN: SELECT ... FROM a LEFT JOIN b ON ...;
GROUP BY/HAVING: SELECT ... FROM ... GROUP BY ... HAVING ...;

UPDATE: UPDATE <tabla> SET col=expr [, col2=expr2] WHERE condición;
UPDATE JOIN: UPDATE a JOIN b ON ... SET a.col=... WHERE ...;

DELETE: DELETE FROM <tabla> WHERE condición;
DELETE JOIN: DELETE a FROM a JOIN b ON ... WHERE ...;
TRUNCATE: TRUNCATE TABLE <tabla>;

CREATE TABLE: CREATE TABLE <t> (col tipo [NOT NULL] [DEFAULT ...], ..., PRIMARY KEY(...), [FORE
ALTER TABLE ADD: ALTER TABLE <t> ADD <definición>;
ALTER TABLE MODIFY/CHANGE: ALTER TABLE <t> MODIFY/CHANGE <definición>;
CREATE INDEX: CREATE [UNIQUE|FULLTEXT] INDEX <idx> ON <t>(col[, col2]);
CREATE VIEW: CREATE VIEW <v> AS SELECT ...;
PROCEDURE: CREATE PROCEDURE <p>(IN p1 tipo, OUT p2 tipo) BEGIN ... END;
TRIGGER: CREATE TRIGGER <nombre> {BEFORE|AFTER} {INSERT|UPDATE|DELETE} ON <t> FOR EACH ROW BEGI
```

14) Errores y confusiones típicas

- Usar INT para dinero → mejor DECIMAL.
- LIKE '%texto%' esperando índice → usa FULLTEXT o busca por prefijo.
- Olvidar zona horaria → guarda en UTC y convierte en la app.
- ENUM para catálogos cambiantes → usa tabla + FK.
- UPDATE/DELETE sin WHERE → desastre (haz backup/usa transacciones).

Fin de la guía. Sugerencia: imprime esta página o guárdala cerca de tu editor. ¡Feliz consulta SQL!