Casos Prácticos de Estudio para Diseño de Bases de Datos en MySQL

Fase	Contenido	Herramienta
1	**Introducción a MySQL**	MySQL
2	**DDL práctico**: Crear base de datos "universidad"	Editor SQL
3	**DML práctico**: Poblar tablas y consultas básicas	Editor SQL
4	**JOIN simple**: Relacionar 2-3 tablas	Editor SQL
5	**Ejercicios de consolidación**	MySQL Workbench

Caso de Estudio: Sistema de Gestión Universitaria Simple

Contexto: Universidad que necesita gestionar:

- **Facultades** (id, nombre, decano)
- **Carreras** (id, nombre, id_facultad, duración_semestres)
- **Estudiantes** (id, código, nombres, apellidos, id_carrera, fecha_ingreso)
- **Profesores** (id, código, nombres, apellidos, id_facultad, especialidad)
- **Materias** (id, código, nombre, id_carrera, creditos)
- **Inscripciones** (id, id_estudiante, id_materia, semestre, año, nota_final)

Relaciones simples:

- Una facultad tiene muchas carreras
- Una carrera tiene muchos estudiantes y muchas materias
- Un estudiante se inscribe en muchas materias
- Un profesor pertenece a una facultad

Fase 1: Introducción a MySQL

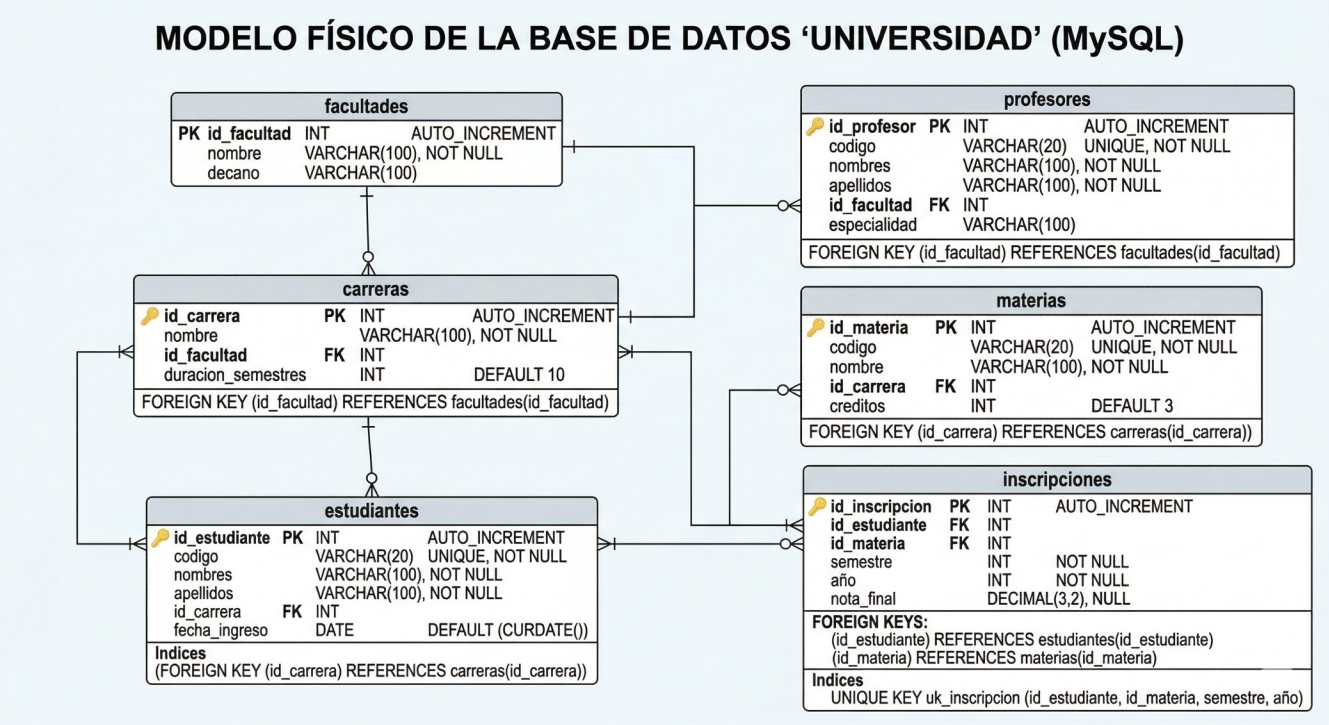
Objetivos:

- Conectar a MySQL
- Crear una conexión al servidor local
- Conocer la interfaz: Navigator, Query Editor, Result Grid

Actividad guiada:

1. Abrir MySQL Workbench
2. Crear conexión "localhost" (root / contraseña)
3. Explorar el esquema "sys" o "mysql" (solo ver, no tocar)
4. Crear un nuevo esquema llamado `universidad_practica`

Fase 2: DDL - Crear la base de datos



Script completo :
sql

```
-- Crear base de datos
CREATE DATABASE IF NOT EXISTS universidad;
USE universidad;

-- Tabla facultades
CREATE TABLE facultades (
    id_facultad INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    decano VARCHAR(100)
);

-- Tabla carreras
CREATE TABLE carreras (
    id_carrera INT AUTO_INCREMENT PRIMARY KEY,
```

```
    nombre VARCHAR(100) NOT NULL,
    id_facultad INT,
    duracion_semestres INT DEFAULT 10,

    FOREIGN KEY (id_facultad) REFERENCES facultades(id_facultad)
);

-- Tabla estudiantes
CREATE TABLE estudiantes (
    id_estudiante INT AUTO_INCREMENT PRIMARY KEY,
    codigo VARCHAR(20) UNIQUE NOT NULL,
    nombres VARCHAR(100) NOT NULL,
    apellidos VARCHAR(100) NOT NULL,
    id_carrera INT,
    fecha_ingreso DATE DEFAULT (CURDATE()),

    FOREIGN KEY (id_carrera) REFERENCES carreras(id_carrera)
);

-- Tabla profesores
CREATE TABLE profesores (
    id_profesor INT AUTO_INCREMENT PRIMARY KEY,
    codigo VARCHAR(20) UNIQUE NOT NULL,
    nombres VARCHAR(100) NOT NULL,
    apellidos VARCHAR(100) NOT NULL,
    id_facultad INT,
    especialidad VARCHAR(100),

    FOREIGN KEY (id_facultad) REFERENCES facultades(id_facultad)
);

-- Tabla materias
CREATE TABLE materias (
    id_materia INT AUTO_INCREMENT PRIMARY KEY,
```

```
    codigo VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    id_carrera INT,
    creditos INT DEFAULT 3,

    FOREIGN KEY (id_carrera) REFERENCES carreras(id_carrera)
);

-- Tabla inscripciones (tabla de relación muchos a muchos)
CREATE TABLE inscripciones (
    id_inscripcion INT AUTO_INCREMENT PRIMARY KEY,
    id_estudiante INT,
    id_materia INT,
    semestre INT NOT NULL,
    año INT NOT NULL,
    nota_final DECIMAL(3,2) NULL, -- 0.00 a 5.00

    FOREIGN KEY (id_estudiante) REFERENCES estudiantes(id_estudiante),
    FOREIGN KEY (id_materia) REFERENCES materias(id_materia),

    -- Evitar inscripción duplicada
    UNIQUE KEY uk_inscripcion (id_estudiante, id_materia, semestre, año)
);
```

Nota sobre DEFAULT (CURDATE())

La función `CURDATE()` devuelve la fecha actual del sistema (sin la hora).
Al usarla como valor por defecto:

- Si insertas un estudiante sin especificar `fecha_ingreso`, MySQL guarda automáticamente la fecha de hoy
- Si insertas un estudiante especificando una fecha, se usa la que tú indiques

Ejemplo:
sql
-- Guarda la fecha de hoy automáticamente, porque no le ingresaste el campo

```
fecha_ingreso

      INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera)
      VALUES ('202410001', 'Luis', 'Torres', 1);

-- Guarda la fecha que tú especifiques

      INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera,
      fecha_ingreso)
      VALUES ('202310006', 'Sofia', 'Ruiz', 2, '2023-02-15');
```

Ejercicios durante la fase:

- Ejecutar el script completo
- Verificar con `SHOW TABLES;`
- Describir una tabla: `DESCRIBE estudiantes;`
- Modificar una tabla: agregar columna email a estudiantes

Fase 3: DML - Poblar y consultar

Inserción de datos:
sql

```
-- Facultades
INSERT INTO facultades (nombre, decano) VALUES
('Ingeniería', 'Dr. Carlos Martínez'),
('Ciencias Básicas', 'Dra. Ana López'),
('Ciencias Sociales', 'Dr. Pedro Gómez');


-- Carreras
INSERT INTO carreras (nombre, id_facultad, duracion_semestres) VALUES
('Ingeniería de Sistemas', 1, 10),
('Tecnología en Desarrollo de Software', 1, 6),
('Matemáticas', 2, 10),
('Psicología', 3, 10);


-- Estudiantes
INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera, fecha_ingreso)
VALUES
```

```
('202310001', 'Juan Carlos', 'Rodríguez Pérez', 2, '2023-02-01'),
('202310002', 'María Fernanda', 'López García', 2, '2023-02-01'),
('202310003', 'Carlos Alberto', 'Martínez Silva', 1, '2022-02-01'),
('202320001', 'Ana Lucía', 'Gómez Torres', 3, '2023-08-01');

-- Profesores
INSERT INTO profesores (codigo, nombres, apellidos, id_facultad, especialidad)
VALUES
('P001', 'Luis Eduardo', 'Fernández Ruiz', 1, 'Bases de Datos'),
('P002', 'Diana Patricia', 'Castro Mendoza', 1, 'Programación Web'),
('P003', 'Roberto', 'Gutiérrez Vargas', 2, 'Estadística');

-- Materias
INSERT INTO materias (codigo, nombre, id_carrera, creditos) VALUES
('BD101', 'Base de Datos I', 2, 3),
('BD201', 'Base de Datos II', 2, 3),
('PG101', 'Programación I', 2, 4),
('PG102', 'Programación II', 1, 4),
('MT101', 'Cálculo I', 3, 4);

-- Inscripciones
INSERT INTO inscripciones (id_estudiante, id_materia, semestre, año, nota_final)
VALUES
(1, 1, 1, 2023, 4.5),  -- Juan Carlos en BD I, nota 4.5
(1, 3, 1, 2023, 4.0),  -- Juan Carlos en Programación I
(2, 1, 1, 2023, 3.8),  -- María en BD I
(2, 3, 1, 2023, 4.2),  -- María en Programación I
(3, 4, 4, 2023, 4.7);  -- Carlos en Programación II
```

Comprobar:

1. Comprobar la estructura de la tabla estudiantes

```
DESCRIBE estudiantes;
```

2. Ver los datos de la tabla estudiantes

*SELECT * FROM estudiantes;*

Consultas básicas:

1. Listar todos los estudiantes

*SELECT * FROM estudiantes;*

2. Obtiene solo apellidos de la tabla estudiantes

SELECT apellidos FROM estudiantes;

3. Obtiene nombre completo (nombre y apellido juntos)

SELECT nombres, apellidos FROM estudiantes;

4. Obtiene todos los identificadores y nombres de la tabla estudiantes

SELECT id_estudiante, name FROM estudiantes;

5. Cambia el nombre de la columna mostrada a "Nombre_Estudiante"

SELECT nombres AS Nombre_Estudiante FROM estudiantes;

ó

SELECT nombres AS ‘Nombre Estudiante’ FROM estudiantes;

6. Múltiples alias

SELECT id_estudiante AS ID, nombres AS Nombre_Estudiante, apellidos AS Apellido_Estudiante FROM estudiantes;

ó

SELECT id_estudiante AS ID, nombres AS ‘Nombre Estudiante’, apellidos AS ‘Apellido Estudiante’ FROM estudiantes;

Ejercicios de modificación de estructura (ALTER TABLE)

1. Agregar una columna `email` de tipo `VARCHAR(150)` a la tabla `estudiantes`.

ALTER TABLE estudiantes ADD correo_electronico *VARCHAR(150);*

2. Cambia el tipo de dato de correo_electronico a VARCHAR(200) para permitir

emails más largos.

ALTER TABLE estudiantes MODIFY correo_electronico VARCHAR(200);

3. Renombra la columna correo_electronico a email.

ALTER TABLE estudiantes RENAME COLUMN correo_electronico TO email;

4. Agregar una columna `correo` de tipo `VARCHAR(150)` a la tabla `estudiantes`, para despues borrarlo.

ALTER TABLE estudiantes ADD correo VARCHAR(150);

ALTER TABLE estudiantes DROP COLUMN correo;

Precaución: DROP COLUMN elimina permanentemente los datos. No tiene "deshacer".

Adicionamos datos en email

UPDATE estudiantes SET email='juancarlos@gmail.com' WHERE id_estudian te=1;

7. Mostrar email, pero si es NULL muestra 'Sin email'.

SELECT nombres, apellidos, IFNULL(email, 'Sin email') FROM estudiantes;

8. Muestra email, pero si es NULL muestra 'Sin email'. Cambiamos el titulo del campo Email a estudiantes sin email.

SELECT nombres, apellidos, IFNULL(email, 'Sin email') AS 'estudiantes sin email' FROM estudiantes;

9. Solo usuarios llamados Juan Carlos

*SELECT * FROM estudiantes WHERE nombres = 'Juan Carlos';*

10. Calcula años desde la fecha de ingreso (si fecha_ingreso existe)

SELECT nombres, fecha_ingreso, YEAR(CURDATE()) - YEAR(fecha_ingreso) AS años_transcurridos FROM estudiantes WHERE fecha_ingreso IS NOT NULL;

Quieres obtener...	**Usa...**
Solo registros que existen en ambas tablas	`INNER JOIN` ó `JOIN`
Todos los registros de la tabla principal, aunque no tengan relación	`LEFT JOIN`
Todos los registros de la tabla secundaria, aunque no tengan relación	`RIGHT JOIN`
Todos los registros de ambas tablas, completando con NULL donde no haya coincidencia	`FULL JOIN` (o `UNION` en MySQL)

11. Ver los estudiantes y su carrera, uniendo con JOIN datos de dos tablas que están relacionadas (tienen una llave foránea), como lo son las tablas estudiantes y carreras

Estructura básica:

*SELECT * FROM tabla1 JOIN tabla2 ON tabla1.llave_primaria=tabla2.llave_foranea ;*

Quedando así:

SELECT * FROM estudiantes JOIN carreras ON estudiantes.id_carrera = carreras.id_carrera;

Compararlo invirtiendo el orden de las tablas:

*SELECT * FROM carreras JOIN estudiantes ON estudiantes.id_carrera = carreras.id_carrera;*

- *¿Que diferencias encuentras?*
- *¿Importa el orden en que ubiquemos las tablas?*

12. Ver el código, nombres, apellidos y carrera de los estudiantes, uniendo con JOIN datos de dos tablas que están relacionadas (tienen una llave foránea), como lo son las tablas estudiantes y carreras

Estructura básica:

```
SELECT tabla1.campo1, tabla1.campo2, tabla1.campo3, tabla2.campo1 FROM tabla1
JOIN tabla2 ON tabla1.llave_primaria=tabla2.llave_foranea ;
```

Quedando asi:

```
SELECT estudiantes.codigo, estudiantes.nombres, estudiantes.apellidos,
carreras.nombre AS carrera
FROM estudiantes
JOIN carreras ON estudiantes.id_carrera = carreras.id_carrera;
```

13. Ver el código, nombres, apellidos y carrera de los estudiantes de la carrera Tecnología en Desarrollo de Software. Usando alias en el nombre de la tabla.

```
SELECT e.codigo, e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
JOIN carreras c ON e.id_carrera = c.id_carrera
WHERE c.nombre = 'Tecnología en Desarrollo de Software';
```

14. INNER JOIN con tres tablas, estudiantes, carreras y facultades.

```
SELECT * FROM estudiantes e
INNER JOIN carreras c ON e.id_carrera = c.id_carrera
INNER JOIN facultades f ON c.id_facultad = f.id_facultad;
```

15. INNER JOIN con tres tablas, estudiantes, carreras y facultades. Mostrando nombres, apellidos, carrera y facultad.

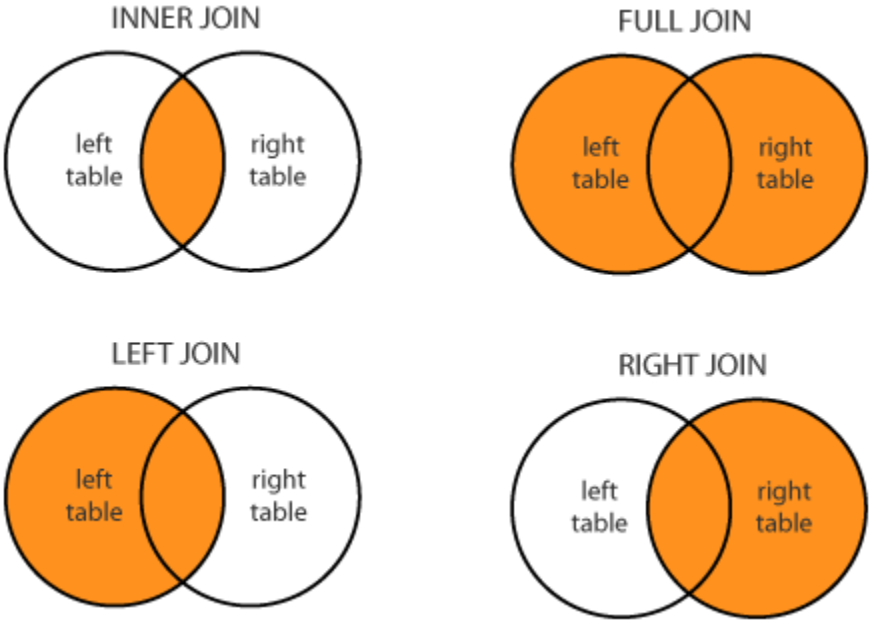
```
SELECT
    e.nombres,
    e.apellidos,
```

```
    c.nombre AS carrera,
    f.nombre AS facultad
FROM estudiantes e
INNER JOIN carreras c ON e.id_carrera = c.id_carrera
INNER JOIN facultades f ON c.id_facultad = f.id_facultad;
```

Fase 4: JOINs - Uniendo tablas

Adicionamos el siguiente registro o tupla:

```
INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera,
fecha_ingreso) VALUES ('202310005', 'Pedro', 'Gómez', NULL, '2023-02-01');
```



1. INNER JOIN

1.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e INNER JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM carreras c INNER JOIN estudiantes e ON e.id_carrera =
c.id_carrera;
```

1.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
INNER JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM carreras c
INNER JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

2. LEFT JOIN

2.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e LEFT JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM  carreras c LEFT JOIN estudiantes e ON e.id_carrera =
c.id_carrera;
```

2.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM  carreras c
LEFT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

3. RIGHT JOIN

3.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e RIGHT JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM carreras c RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

3.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

- El resultado es igual a alguno de los dos ejemplos del LEFT JOIN?
- Comparar el orden en nombres y carrera en el left y right join.
- Cual es el patrón del orden?

¿Por qué MySQL no tiene FULL JOIN?

⚠ Nota importante: MySQL no implementa la sintaxis `FULL OUTER JOIN` nativamente.

¿Por qué? Es una decisión de diseño del motor. Otros motores como PostgreSQL o SQL Server sí lo soportan.

Solución en MySQL: Simulamos el comportamiento usando UNION entre un LEFT JOIN y un RIGHT JOIN.

UNION elimina automáticamente las filas duplicadas (las que coinciden en ambas tablas). Si quisieras mantener duplicados, usarías `UNION ALL`.

4. FULL JOIN (simulado con UNION en MySQL, ya que MySQL no tiene FULL JOIN)

4.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e
```

```
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION
SELECT * FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM carreras c
LEFT JOIN estudiantes e ON e.id_carrera = c.id_carrera
UNION
SELECT * FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

4.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Comparar usando UNION ALL

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION ALL
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

UNION elimina los registros duplicados del resultado final, mientras que UNION ALL incluye todas las filas, duplicados incluidos

Pregunta:

Cual es la diferencia entres estas dos consultas:

```
SELECT * FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

y

```
SELECT * FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

Comparación visual con datos reales

Usando las tablas *estudiantes* y *carreras*:

JOIN	**Filas resultado**	**¿Qué muestra?**
INNER JOIN	4 filas	Solo estudiantes que SÍ tienen carrera
LEFT JOIN	5 filas	Todos los estudiantes.
RIGHT JOIN	5 filas	Todas las carreras. Los sin estudiantes muestran `NULL`
FULL JOIN	6 filas	Todos los estudiantes + todas las carreras, completando con `NULL`

Otros ejemplos

5. Materias sin estudiantes inscritos (LEFT JOIN)

```
SELECT m.codigo, m.nombre, c.nombre AS carrera
FROM materias m
JOIN carreras c ON m.id_carrera = c.id_carrera
LEFT JOIN inscripciones i ON m.id_materia = i.id_materia
WHERE i.id_inscripcion IS NULL;
```

6. Actualizar nota

```
UPDATE inscripciones
SET nota_final = 4.8
WHERE id_estudiante = 1 AND id_materia = 1;
```

7. Eliminar inscripción (ejemplo académico)

```
DELETE FROM inscripciones
WHERE id_estudiante = 2 AND id_materia = 3;
```

DELETE vs TRUNCATE: ¿Cuál usar?

Característica	DELETE	TRUNCATE
Velocidad	Lento (registro por registro)	Rápido (elimina la tabla y la recrea)
WHERE	Sí, puedes filtrar	No, elimina TODO
Reinicia AUTO_INCREMENT	No	Sí, vuelve a 1

Ejemplo comparativo:
```sql
-- Elimina solo las inscripciones del año 2023

      DELETE FROM inscripciones WHERE año = 2023;

-- Elimina TODAS las inscripciones y reinicia el contador de id_inscripcion


      TRUNCATE TABLE inscripciones;

-- Sintaxis básica para eliminar una tabla

      DROP TABLE nombre_tabla;

-- Sintaxis recomendada para evitar errores si la tabla no existe

      DROP TABLE IF EXISTS nombre_tabla;
```

Regla práctica: Usa DELETE cuando necesites condiciones. Usa TRUNCATE solo cuando quieras vaciar completamente una tabla y no te importe perder todo el historial.

Cuando defines una columna como `PRIMARY KEY` o `UNIQUE`, MySQL crea automáticamente un índice sobre esa columna. Esto hace que las búsquedas por `id_estudiante` o `codigo` sean muy rápidas, incluso con millones de registros.

Puedes ver los índices de una tabla con:
sql

```
      SHOW INDEX FROM estudiantes;
```

Fase 5: Ejercicios de consolidación

Ejercicios para resolver en clase:

1. Agregar 2 nuevas materias a la carrera de Tecnología en Desarrollo de Software
2. Inscribir al estudiante Carlos Alberto en Base de Datos I con nota 4.2
3. Listar todos los profesores de la facultad de Ingeniería
4. Calcular cuántos estudiantes ingresaron en 2023
5. Mostrar la carrera con más estudiantes

Preguntas de cierre:

- ¿Por qué usamos `DECIMAL(3,2)` para notas y no `FLOAT`?
- ¿Qué pasa si intentas borrar una facultad que tiene carreras asociadas?