Ejemplo 2 de Normalización

- Un escenario inicial con datos repetidos y desorganizados.
- Aplicación progresiva de cada forma normal.
- Explicación clara del proceso.
- Resultado final estructurado y optimizado.

Escenario Inicial: Sistema de Matrícula de Cursos

Imagina que estamos diseñando una base de datos para registrar los estudiantes matriculados en cursos de una institución educativa.

Tabla inicial (sin normalizar):

EstudianteID	NombreEstudiante	Curso	NombreProfesor	Nota
1	Ana Gómez	Matemáticas, Ciencias	Juan Pérez	8.5
2	Luis Mendoza	Historia	María López	9.0
3	Carlos Díaz	Lenguaje, Arte, Inglés	Laura Torres	7.0

Esta tabla no está normalizada:

- Tiene grupos de valores repetidos (varios cursos por estudiante).
- No tiene una clave primaria bien definida.
- Contiene redundancias (repetición de nombres de profesores).

Paso 1: Primera Forma Normal (1FN)

Objetivo:
- Eliminar grupos repetidos y asegurar que cada campo tenga un solo valor (valores atómicos).

Acciones:
- Separar los cursos múltiples en filas independientes
- Asignar una clave primaria compuesta si es necesario.

Tabla en 1FN:

EstudianteID	NombreEstudiante	Curso	NombreProfesor	Nota
1	Ana Gómez	Matemáticas	Juan Pérez	8.5
1	Ana Gómez	Ciencias	Juan Pérez	8.5
2	Luis Mendoza	Historia	María López	9.0
3	Carlos Díaz	Lenguaje	Laura Torres	7.0
3	Carlos Díaz	Arte	Laura Torres	7.0

3	Carlos Díaz	Inglés	Laura Torres	7.0

Ahora:
- Todos los campos son atómicos.
- Se eliminaron listas de valores.
- Pero hay redundancia: el profesor se repite varias veces para el mismo curso.

Paso 2: Segunda Forma Normal (2FN)

Objetivo:
- Eliminar dependencias parciales, es decir, que todos los atributos no clave dependan completamente de la clave primaria.

Análisis:
La clave primaria actual es `(EstudianteID, Curso)`, pero:
- `NombreEstudiante` depende solo de `EstudianteID`.
- `Profesor` depende solo de `Curso`.

Solución:
Dividir la tabla en tres tablas relacionadas.

Tabla Estudiantes:

EstudianteID	NombreEstudiante
1	Ana Gómez
2	Luis Mendoza
3	Carlos Díaz

Tabla Cursos:

CursoID	NombreCurso	NombreProfesor
1	Matemáticas	Juan Pérez
2	Ciencias	Juan Pérez
3	Historia	María López
4	Lenguaje	Laura Torres
5	Arte	Laura Torres
6	Inglés	Laura Torres

Tabla Matricula (Relación entre Estudiantes y Cursos):

EstudianteID	CursoID	Nota
1	1	8.5
1	2	8.5
2	3	9.0
3	4	7.0
3	5	7.0
3	6	7.0

Ahora:

- No hay dependencias parciales.
- Cada tabla tiene un propósito claro.
- Redundancia reducida.

Paso 3: Tercera Forma Normal (3FN)

Objetivo:

- Eliminar dependencias transitivas, es decir, que ningún atributo no clave dependa de otro atributo no clave.

Análisis actual:

- En la tabla `Cursos`, todo depende de la clave (`CursoID`) ?→ Profesor depende de CursoID?.
- En la tabla `Estudiantes`, todo depende de la clave (`EstudianteID`) → OK.
- En la tabla `Matricula`, todo depende de la clave compuesta (`EstudianteID`, `CursoID`) → OK.

Tabla Cursos:

CursoID	NombreCurso
1	Matemáticas
2	Ciencias
3	Historia
4	Lenguaje
5	Arte
6	Inglés

Tabla Profesores

ProfesorID	NombreProfesor
1	Juan Pérez
2	María López
3	Laura Torres

Tabla Asignación (Relación entre Cursos y Profesores):

CursoID	ProfesorID
1	1
2	1
3	2
4	3
5	3
6	3

Relaciones

Tabla Estudiantes
- EstudianteID (PK)
- NombreEstudiante

Tabla Matricula
- EstudianteID (FK a Estudiantes)
- CursoID (FK a Cursos)
- Nota

Tabla Cursos
- CursoID (PK)
- NombreCurso

Tabla Profesores
- ProfesorID (PK)
- NombreProfesor

Tabla Asignación
- ProfesorID (FK)
- CursoID (FK)

Beneficios del Proceso de Normalización

Característica	Antes	Después
Repetición de datos	Alta	Baja
Consistencia	Pobre	Alta
Facilidad de mantenimiento	Difícil	Fácil
Complejidad de consultas	Baja	Media
Rendimiento	Bueno para pocos datos	Optimizable con índices

Script para crear las tablas

`sql
-- Crear la base de datos
CREATE DATABASE IF NOT EXISTS SistemaMatricula;
USE SistemaMatricula;

Tabla Estudiantes
CREATE TABLE IF NOT EXISTS Estudiantes (
EstudianteID INT AUTO_INCREMENT PRIMARY KEY,
NombreEstudiante VARCHAR(100) NOT NULL
);

-- Tabla Cursos
CREATE TABLE IF NOT EXISTS Cursos (
CursoID INT AUTO_INCREMENT PRIMARY KEY,
NombreCurso VARCHAR(100) NOT NULL,
Profesor VARCHAR(100) NOT NULL
);

-- Tabla Matricula
CREATE TABLE IF NOT EXISTS Matricula (
EstudianteID INT NOT NULL,
CursoID INT NOT NULL,
Nota DECIMAL(4,2),
PRIMARY KEY (EstudianteID, CursoID),
FOREIGN KEY (EstudianteID) REFERENCES Estudiantes(EstudianteID),
FOREIGN KEY (CursoID) REFERENCES Cursos(CursoID)
);

-- Tabla Profesores
CREATE TABLE IF NOT EXISTS Profesores (
ProfesorID INT AUTO_INCREMENT PRIMARY KEY,
NombreProfesor VARCHAR(100) NOT NULL
);

-- Tabla Asignacion
```
CREATE TABLE IF NOT EXISTS Asignacion (
    ProfesorID INT NOT NULL,
    CursoID INT NOT NULL,
    PRIMARY KEY (ProfesorID, CursoID),
    FOREIGN KEY (ProfesorID) REFERENCES Profesores(ProfesorID),
    FOREIGN KEY (CursoID) REFERENCES Cursos(CursoID)
);
```

Script para insertar los registros

sql
-- Insertar estudiantes
```
INSERT INTO Estudiantes (NombreEstudiante) VALUES
('Ana Gómez'),
('Luis Mendoza'),
('Carlos Díaz');
```

-- Insertar cursos
```
INSERT INTO Cursos (NombreCurso, Profesor) VALUES
('Matemáticas', 'Juan Pérez'),
('Ciencias', 'Juan Pérez'),
('Historia', 'María López'),
('Lenguaje', 'Laura Torres'),
('Arte', 'Laura Torres'),
('Inglés', 'Laura Torres');
```

-- Insertar matrículas
```
INSERT INTO Matricula (EstudianteID, CursoID, Nota) VALUES
(1, 1, 8.5),
(1, 2, 8.5),
(2, 3, 9.0),
(3, 4, 7.0),
(3, 5, 7.0),
(3, 6, 7.0);
```

Consulta final con JOIN para ver los datos combinados

sql
```
SELECT
    e.NombreEstudiante,
    c.NombreCurso,
    c.Profesor,
    m.Nota
FROM Matricula m
JOIN Estudiantes e ON m.EstudianteID = e.EstudianteID
JOIN Cursos c ON m.CursoID = c.CursoID
ORDER BY e.NombreEstudiante, c.NombreCurso;
```

ó

sql
```
SELECT
    estudiantes.NombreEstudiante,
    cursos.NombreCurso,
    cursos.Profesor,
    matricula.Nota
FROM matricula JOIN estudiantes ON matricula.EstudianteID = estudiantes.EstudianteID JOIN
cursos ON matricula.CursoID = cursos.CursoID
ORDER BY estudiantes.NombreEstudiante, cursos.NombreCurso;
```

ó
```
SELECT
        estudiantes.NombreEstudiante,
        cursos.NombreCurso,
        cursos.Profesor,
        matricula.Nota
FROM cursos JOIN matricula ON matricula.CursoID = cursos.CursoID JOIN estudiantes ON
matricula.EstudianteID = estudiantes.EstudianteID;
```

Probar :
sql
```
SELECT
    estudiantes.NombreEstudiante as Estudiante,
    cursos.NombreCurso as Curso,
    cursos.Profesor,
    matricula.Nota
FROM matricula JOIN estudiantes ON matricula.EstudianteID = estudiantes.EstudianteID
JOIN cursos ON matricula.CursoID = cursos.CursoID
ORDER BY estudiantes.NombreEstudiante, cursos.NombreCurso;
```
Resultado esperado de la consulta:

NombreEstudiante	**NombreCurso**	**Profesor**	**Nota**
Ana Gómez	Ciencias	Juan Pérez	8.50
Ana Gómez	Matemáticas	Juan Pérez	8.50
Carlos Díaz	Arte	Laura Torres	7.00
Carlos Díaz	Inglés	Laura Torres	7.00
Carlos Díaz	Lenguaje	Laura Torres	7.00
Luis Mendoza	Historia	María López	9.00

Ejemplo: Promedio de notas por estudiante

sql
SELECT
e.NombreEstudiante,
AVG(m.Nota) AS Promedio
FROM Matricula m
JOIN Estudiantes e ON m.EstudianteID = e.EstudianteID
GROUP BY e.NombreEstudiante;
```
```