Database Normalization: 1NF, 2NF and 3NF

WORKED PRACTICE • COMPUTER SCIENCE • SQL DESIGN

Scenario: a fictional university stores student module enrolments, the module title, and the lecturer name. The aim is to avoid duplicated data and anomalies while retaining all relationships.

1. Identify the starting data and dependencies

student_id	student_name	module_id	module_title	lecturer_id	grade
101	Amina	DB1	Databases	7	A
101	Amina	CS2	Networks	8	B
102	Rafi	DB1	Databases	7	B

Assumptions: one student may take many modules; a student has at most one grade per module; each module has a lecturer. The natural composite key of the enrolment rows is (student_id, module_id).

Functional dependencies

student_id → student_name; module_id → module_title, lecturer_id; (student_id, module_id) → grade. If lecturer names are stored as well, lecturer_id → lecturer_name.

2. First normal form (1NF)

Each cell should contain an atomic value for this design, and each enrolment is one row. This table is already in 1NF under these assumptions; repeating student names still causes update anomalies.

3. Second normal form (2NF)

Separate attributes depending only on part of the composite key into their own tables: Student(student_id, student_name), Module(module_id, module_title, lecturer_id), and Enrollment(student_id, module_id, grade). Student and module attributes no longer repeat in every enrolment.

4. Third normal form (3NF)

If lecturer_name is stored alongside lecturer_id in Module, it depends on lecturer_id rather than directly on module_id. Introduce Lecturer(lecturer_id, lecturer_name), retain lecturer_id as a foreign key in Module, and keep the other three tables.

Important modelling note

Database normalization depends on correctly stated business rules and functional dependencies. Different requirements can justify different designs. This is an instructional example, not a student submission.

3NF Schema and a Join Query

STUDY AND ADAPT • ORIGINAL SQL EXAMPLE

The following portable-looking SQL uses familiar relational syntax. Minor type or identifier adjustments may be necessary in your chosen DBMS.

```
CREATE TABLE Student (
  student_id INT PRIMARY KEY,
  student_name VARCHAR(100) NOT NULL
);
CREATE TABLE Lecturer (
  lecturer_id INT PRIMARY KEY,
  lecturer_name VARCHAR(100) NOT NULL
);
CREATE TABLE Module (
  module_id VARCHAR(10) PRIMARY KEY,
  module_title VARCHAR(100) NOT NULL,
  lecturer_id INT NOT NULL,
  FOREIGN KEY (lecturer_id) REFERENCES Lecturer(lecturer_id)
);
CREATE TABLE Enrollment (
  student_id INT NOT NULL,
  module_id VARCHAR(10) NOT NULL,
  grade VARCHAR(3),
  PRIMARY KEY (student_id, module_id),
  FOREIGN KEY (student_id) REFERENCES Student(student_id),
  FOREIGN KEY (module_id) REFERENCES Module(module_id)
);
```

Sample query: list students and the modules they take

```
SELECT s.student_name, m.module_title, e.grade
FROM Enrollment e
JOIN Student s ON s.student_id = e.student_id
JOIN Module m ON m.module_id = e.module_id
ORDER BY s.student_name, m.module_title;
```

Self-check

Q1. Why is (student_id, module_id) an appropriate composite key for Enrollment under the given rules? Q2. Which anomaly appears if lecturer_name is repeated in every enrolment row? Q3. How would the model change if a student can retake a module in different semesters?

Answers to test your reasoning

1. The pair identifies one student-module enrolment; neither column alone is unique. 2. A lecturer name change would need repeated updates, risking inconsistencies. 3. A semester or enrolment identifier may be needed in the key or a separate enrolment entity, depending on the rules.

Useful learning links

PostgreSQL SQL tutorial • Database study guide • Request tutoring and project feedback