LESSON 8

Normalization

Ihor Liutak

Objectives

1. Understand the purpose of normalization.
2. Learn rules and processes for achieving 1NF, 2NF, and 3NF.
3. Practice identifying and fixing normalization issues.

Introduction to Normalization

Definition

Normalization is the process of structuring a relational database in accordance with a series of normal forms.

Main Goals

Reduce data redundancy

Improve data integrity and consistency

Why Normalize?

Easier maintenance

Clearer data relationships

Efficient data updates

Tables with repeating groups → Separated, smaller tables
Reduced redundancy and fewer anomalies

Additional Example for Introduction

-- Common problem: Repeated customer info in multiple tables
-- After normalization: Single customer table, references used elsewhere

Characteristics of an Unnormalized Dataset

What is an Unnormalized Dataset?

Contains redundancy.

May store multiple values in a single column.

Difficult to query and maintain.

Issues:

Multiple products stored in a single column.

Difficult to search or aggregate specific product data.

Orders Table:

OrderID	Customer	Products
1	John Smith	Apple, Banana, Orange
2	Jane Doe	Mango, Apple

First Normal Form (1NF)

Rule Each column holds *atomic* (indivisible) values.
No repeating groups or arrays.

Key Criteria No multi-valued attributes
Each column contains a single value

```sql
-- Unnormalized Table
CREATE TABLE unnormalized (
  student_id INT,
  name       VARCHAR(50),
  courses    VARCHAR(100) -- e.g., "Math, Science"
);

INSERT INTO unnormalized VALUES
(1, 'Alice', 'Math, Science'),
(2, 'Bob',   'History, Math');
```

```sql
-- Converting to 1NF:
-- Split courses into separate rows

CREATE TABLE students_1nf (
  student_id INT,
  name       VARCHAR(50),
  course     VARCHAR(50)
);

INSERT INTO students_1nf VALUES
(1, 'Alice', 'Math'),
(1, 'Alice', 'Science'),
(2, 'Bob',   'History'),
(2, 'Bob',   'Math');
```

Second Normal Form (2NF)

Rule

Table must be in 1NF.
Full functional dependency on the
entire primary key (no partial dependency).

When It Matters

Typically applies when you have a composite primary key.
student_name & course_title may cause partial dependencies.

```sql
-- Example table in 1NF, but not in 2NF
CREATE TABLE enrollments_1nf (
  student_id INT,
  course_id  INT,
  student_name VARCHAR(50),
  course_title VARCHAR(100),
  PRIMARY KEY (student_id, course_id)
);
```

```sql
-- Split into two tables for 2NF

CREATE TABLE students (
  student_id   INT PRIMARY KEY,
  student_name VARCHAR(50)
);

CREATE TABLE courses (
  course_id    INT PRIMARY KEY,
  course_title VARCHAR(100)
);

CREATE TABLE enrollments_2nf (
  student_id INT,
  course_id  INT,
  PRIMARY KEY (student_id, course_id),
  FOREIGN KEY (student_id) REFERENCES students(student_id),
  FOREIGN KEY (course_id)  REFERENCES courses(course_id)
);
```

Third Normal Form (3NF)

Rule

Table must be in 2NF.

No *transitive dependencies* (non-key columns depending on other non-key columns).

Key Example

student_id → department_id → department_name

In 3NF, separate *department* details into another table.

This separation removes the transitive dependency on department_name

```sql
-- Insert sample data into departments
INSERT INTO departments VALUES
(1, 'Computer Science'),
(2, 'Mathematics');

-- Insert related data in students_3nf
INSERT INTO students_3nf VALUES
(101, 'Alice', 1),
(102, 'Bob',   2);
```

```sql
-- 2NF table with transitive dependency:
CREATE TABLE students_2nf (
  student_id       INT PRIMARY KEY,
  student_name     VARCHAR(50),
  department_id    INT,
  department_name  VARCHAR(100)
);

-- 3NF approach:
CREATE TABLE students_3nf (
  student_id    INT PRIMARY KEY,
  student_name  VARCHAR(50),
  department_id INT
);

CREATE TABLE departments (
  department_id   INT PRIMARY KEY,
  department_name VARCHAR(100)
);
```

SQL procedure to achieve 1NF

```
DELIMITER $$

CREATE PROCEDURE SplitSupplierInfo()
BEGIN
    -- Add new columns for supplier_name and supplier_address
    ALTER TABLE goods
    ADD COLUMN supplier_name VARCHAR(255) AFTER price,
    ADD COLUMN supplier_address VARCHAR(255) AFTER supplier_name;

    -- Update the new columns with data split from supplier_info
    UPDATE goods
    SET
        supplier_name = SUBSTRING_INDEX(supplier_info, ' - ', 1),
        supplier_address = SUBSTRING_INDEX(supplier_info, ' - ', -1);

    -- Drop the old supplier_info column
    ALTER TABLE goods DROP COLUMN supplier_info;
END$$

DELIMITER ;
```

SQL procedure that splits the `supplier_info` column in the `goods` table into two separate columns, `supplier_name` and `supplier_address`, to achieve 1NF

Explanation

Adding Columns: The procedure first adds two new columns, supplier_name and supplier_address, after the price column.

Updating Columns: It then splits the supplier_info using SUBSTRING_INDEX(), which extracts text before and after the delimiter (' - ').

Removing supplier_info: Finally, it drops the old supplier_info column.

Using VSCode Plugin for MySQL Shell

Open VSCode and connect to your MySQL server using the MySQL plugin.

Make sure you are connected to the correct database.

Paste the entire SQL procedure into the query editor.

Run the query to create the procedure.

Then execute: `CALL SplitSupplierInfo();`

SQL procedure to achieve 2NF - 1

```
DELIMITER $$

CREATE PROCEDURE CreateCategoriesTable()
BEGIN
    -- Create the categories table
    CREATE TABLE IF NOT EXISTS categories (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) UNIQUE NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    );

    -- Insert distinct categories from goods table into categories table
    INSERT IGNORE INTO categories (name)
    SELECT DISTINCT category
    FROM goods;

    -- Add a category_id column to the goods table
    ALTER TABLE goods
    ADD COLUMN category_id INT AFTER price;
```

heidenheim.site/2nf.sql

SQL procedure to achieve 2NF - 2

```
    -- Update goods table to link to categories table by category_id
    UPDATE goods
    INNER JOIN categories ON goods.category = categories.name
    SET goods.category_id = categories.id;

    -- Drop the old category column from the goods table
    ALTER TABLE goods DROP COLUMN category;

    -- Add foreign key constraint to goods table for category_id
    ALTER TABLE goods
    ADD CONSTRAINT fk_category
    FOREIGN KEY (category_id)
    REFERENCES categories(id)
    ON DELETE SET NULL;
END$$

DELIMITER ;
```

SQL procedure to achieve 2NF - Explanation

Create `categories` Table:

The procedure creates a new table, `categories`, which contains an `id` (primary key) and `name` (unique category name).

Insert Distinct Categories:

Insert all distinct category names from the `goods` table into the `categories` table to establish a unique list of categories.

Add `category_id` Column:

Add a new column `category_id` to the `goods` table to link goods with categories.

Update Goods Table:

Populate the `category_id` field in `goods` based on the `categories` table by matching the category names.

Remove Old `category` Column:

Drop the original `category` column from the `goods` table as it's now replaced by `category_id`.

Add Foreign Key Constraint:

Add a foreign key to link `category_id` in the `goods` table to the `id` in the `categories` table.

SQL procedure to achieve 3NF

```
DELIMITER $$

CREATE PROCEDURE DecomposeLocation()
BEGIN
    -- Add new columns for shelf and warehouse
    ALTER TABLE goods
    ADD COLUMN shelf VARCHAR(255) AFTER category_id,
    ADD COLUMN warehouse VARCHAR(255) AFTER shelf;

    -- Update the new columns with data split from location
    UPDATE goods
    SET
        shelf = SUBSTRING_INDEX(location, ', ', 1),
        warehouse = SUBSTRING_INDEX(location, ', ', -1);

    -- Drop the old location column
    ALTER TABLE goods DROP COLUMN location;
END$$

DELIMITER ;
```