Database Normalization

Minimizing Redundancies

Learning Objectives

By the end of this session, you will be able to:

- **Define** normalization and its purpose
- **Identify** functional dependencies in data
- **Apply** normalization steps (1NF, 2NF, 3NF)
- **Evaluate** trade-offs in database design

Why Do We Need Normalization?

Problems with File Systems:

- **Structural/data dependence**
- **Data redundancy** leading to:
  - Poor data integrity
  - Security issues
  - Increased storage costs

Solution (partial): Normalization!

What is Normalization?

> **Normalization** is a process for evaluating and correcting table structures to minimize data redundancies and tease apart entities

The Process:

Unnormalized → **1st Normal Form (1NF)** → **2NF** → **3NF** → ...

Each stage is called a **normal form**

Starting Point: Unnormalized Data

Customer Orders File:

Order ID	Customer ID	Customer Name	All Product IDs	All Product Names	All Quantities
1001	42	John Smith	201, 202	Laptop, Mouse	1, 2
1002	64	Emma Rodriguez	201, 203	Laptop, Monitor	1, 3

- **Problem:** Multi-valued fields (All Product IDs, All Product Names, etc.)

Problems with Unnormalized Data

1. **Non-atomic (multi-valued) data**
    - Hard to query (e.g., "find all orders with laptops")
    - Complex to maintain positional structure
2. **Data redundancy**
    - Customer info repeated for each order
    - Product info repeated across orders

First Normal Form (1NF)

Requirements:

1. **Table format** (rows and columns)
2. **No repeating groups** (atomic values only)
3. **Primary key (PK) chosen**

> A field/attribute (or an irreducible set of fields/attributes) that uniquely identifies each row/record

How to achieve 1NF:

Ungroup multi-valued data into separate rows

Converting to 1NF: Example

Before:

Order ID	Customer ID	Customer Name	All Product IDs	All Product Names	All Quantities
1001	42	John Smith	201, 202	Laptop, Mouse	1, 2
1002	64	Emma Rodriguez	201, 203	Laptop, Monitor	1, 3

After:

Order ID	Customer ID	Customer Name	Product ID	Product Name	Quantity
1001	42	John Smith	201	Laptop	1
1001	42	John Smith	202	Mouse	2

Exercise: Identify the PK

- 1:1 map between PK and row
    - One PK identifies one row
    - One row is identified by 1 PK
- Which irreducible subset of *(Order ID, Customer ID Customer Name, Product ID, Product Name, Quantity)* can be a PK?

Answer: (Order ID, Product ID)

Understanding Functional Dependencies

> Field-set B is **functionally dependent** on field-set A if each value of A determines one and only one value of B

Notation: A → B

- A is the **determinant**
- B is the **dependent**

Examples:

- PK → all other fields
  - (Order ID, Product ID) -> (Customer ID, Customer Name, Product Name, Quantity)

Types of Dependencies

1. Full Functional Dependency

All fields in determinant needed to identify dependent

- *(Order ID, Product ID)* → *Quantity* is **full**
- *(Order ID, Product ID)* → *Customer ID* is **not full**

2. Partial Dependency

Determinant is only part of the primary key e.g. *Order ID* → *Customer ID*

3. Transitive Dependency

A → B where both A and B are non-key fields

- *Customer ID* → *Customer Name* is **transitive**

🎯 Active Learning: Identify Dependencies

Given the following table:

- (Student ID, Course ID) -> (Student Name, Course Name, Professor ID, Professor Name, GPA)

List the following:

1. All partial dependencies
2. All transitive dependencies
3. All full-functional dependencies

Work in pairs

Second Normal Form (2NF)

Requirements:

1. Must be in 1NF
2. **No partial dependencies**

How to achieve 2NF:

1. Create a new table for each partial dependency
2. Keep determinant fields in original table (as link to the dependent table)
3. Move dependent fields to new table

Converting to 2NF: Example

1NF:

Order ID	Product ID	Customer ID	Customer Name	Product Name	Quantity
1001	201	42	John Smith	Laptop	1
1001	202	42	John Smith	Mouse	2
1002	201	64	Emma Rodriguez	Laptop	1
1002	203	64	Emma Rodriguez	Monitor	3

2NF:

New table: Products

Product ID	Product Name
201	Laptop
202	Mouse
203	Monitor

New table: Orders

Order ID	Customer ID	Customer Name
1001	42	John Smith
1002	64	Emma Rodriguez

Modified 1NF table: Order Line Items

Order ID	Product ID	Quantity
1001	201	1
1001	202	2
1002	201	1
1002	203	3

Third Normal Form (3NF)

Requirements:

- Must be in 2NF
- **No transitive dependencies**

How to achieve 3NF:

1. Create new table for each transitive dependency
2. Keep determinant in original table (as link to dependent table)
3. Move dependent fields to new table

Converting to 3NF: Example

2NF:

New table: Products

Product ID	Product Name
201	Laptop
202	Mouse
203	Monitor

New table: Orders

Order ID	Customer ID	Customer Name
1001	42	John Smith
1002	64	Emma Rodriguez

Modified 1NF table: Ord

Order ID	Product ID
1001	201
1001	202
1002	201
1002	203

Do any of the above tables have any transitive dependecies?

Converting to 3NF: Example (contd)

2NF Orders table has:

Order ID → Customer ID → Customer Name

After 3NF:

1. **Customers:** Customer ID → Customer Name
2. **Orders:** Order ID → Customer ID, Order Date

Customer info no longer duplicated!

2NF Orders:

<u>Order ID</u>	Customer ID	Customer Name
1001	42	John Smith
1003	42	John Smith

3NF

Customers:

<u>Customer ID</u>	Customer Name
42	John Smith

Orders:

<u>Order ID</u>	Customer ID
1001	42
1003	42

Final Result: 3NF Database

Unnormalized:

Order ID	Customer ID	Customer Name	All Product IDs	All Product Names	All Quantities
1001	42	John Smith	201, 202	Laptop, Mouse	1, 2
1002	64	Emma Rodriguez	201, 203	Laptop, Monitor	1, 3

3NF:

Customers:

<u>Customer ID</u>	Customer Name
42	John Smith
64	Emma Rodriguez

Orders:

<u>Order ID</u>	Customer ID
1001	42
1002	64

Products:

<u>Product ID</u>	Product Name
201	Laptop
202	Mouse
203	Monitor

Order Line Items:

<u>Order ID</u>	<u>Product I</u>
1001	201
1001	202
1002	201
1002	203

Benefits: Minimal redundancy; Clear entity separation; Maintained relationships

🎯 Active Learning: Normalization

Given the following table for which we identified dependencies earlier:

- (Student ID, Course ID) -> (Student Name, Course Name, Professor ID, Professor Name, GPA)

1. Convert each unnormalized table to a 1NF table
2. Convert each table with partial dependencies to 2NF tables
3. Convert each table with transitive dependencies to 3NF tables

Work in the same pairs as the dependencies exercise

Trade-offs in Normalization

Benefits of Higher Normal Forms:

✅ Less redundancy
✅ Better data integrity
✅ Easier updates

Costs:

❌ More tables to join
❌ Slower query performance
❌ More complex queries

When to Stop Normalizing?

General Rule: Stop at 3NF

Why?

- Good balance of structure and performance
- Most redundancy eliminated
- Acceptable query complexity

Sometimes: Deliberate Denormalization

- For frequently accessed data
- To optimize specific queries

🎯 Active Learning: Normalization Review

Quick Quiz (Think, then discuss):

1. What problem does 1NF solve?
2. What's the difference between partial and transitive dependencies?
3. Why might you denormalize a database?

2 minutes individual thinking, then share

Key Takeaways

1. **Normalization** reduces data redundancy through stages
2. **1NF**: Atomic values and primary key
3. **2NF**: No partial dependencies
4. **3NF**: No transitive dependencies
5. **Balance** structure with performance needs