Data, Data Storage, Data Collection

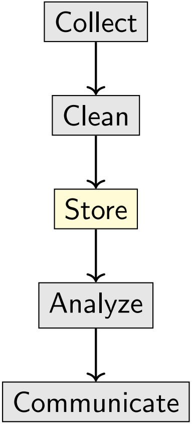
Lecture 10: Data Normalization and Denormalization (Part 2)

Romain Pascual

MICS, CentraleSupélec, Université Paris-Saclay

Recap

Within the lifecycle



Session Objectives

At the end of this session, you should be able to:

- Explain why normalization reduces redundancy and ensures consistency
- Identify and correct design flaws causing data anomalies
- Describe the normal forms up to 5NF and their principles,
- Apply the main normal forms to simple data tables
- Discuss when denormalization may be justified

Normalize?

When data disagrees with itself, it is not “just” a problem of bad data, but a problem of bad design.

Good database design protects against **failure of data integrity** (logical inconsistencies).

Normalize?

When data disagrees with itself, it is not “just” a problem of bad data, but a problem of bad design.

Good database design protects against **failure of data integrity** (logical inconsistencies).

Definition (Normalization)

Process of organizing data into tables and columns in a way that eliminates redundancy and prevents inconsistencies.

Assessed by criteria called **normal forms**:

- 1NF: Atomic values.
- 2NF: No partial dependencies.
- 3NF/BCNF: No transitive dependencies.
- 4NF: No multivalued dependencies.

Keys, and Dependencies: A Refresher

Keys

A superkey is any set of attributes that can **uniquely identify** each row in a table.

Keys, and Dependencies: A Refresher

Keys

A superkey is any set of attributes that can **uniquely identify** each row in a table.

A candidate key is a **minimal** superkey.

Keys, and Dependencies: A Refresher

Keys

A superkey is any set of attributes that can **uniquely identify** each row in a table.

A candidate key is a **minimal** superkey.

A prime attribute is an attribute that is part of **at least one** candidate key.

Keys, and Dependencies: A Refresher

Keys

A superkey is any set of attributes that can **uniquely identify** each row in a table.

A candidate key is a **minimal** superkey.

A prime attribute is an attribute that is part of **at least one** candidate key.

Dependencies

A functional dependency ($X \rightarrow Y$) occurs if for each value in X, there is **exactly one** corresponding value in Y.

Keys, and Dependencies: A Refresher

Keys

A superkey is any set of attributes that can **uniquely identify** each row in a table.

A candidate key is a **minimal** superkey.

A prime attribute is an attribute that is part of **at least one** candidate key.

Dependencies

A functional dependency ($X \rightarrow Y$) occurs if for each value in X, there is **exactly one** corresponding value in Y.

A multivalued dependency ($X \twoheadrightarrow Y$) occurs if for each value in X, there is a **set of values** in Y that depend **only** on X.

Today

- Finish with **5NF**.
- Explore **denormalization**: When and why to break the rules.

(End of) Data Normalization

Flatmate Chores

Alex: I am happy to do Cleaning and Repairs. I know how to use a Brush and a Vacuum.

Mia: I can take care of Decorating and Repairs. I know how to handle a Hammer, a Screw Gun, and a Vacuum.

Sam: I am available for Decorating and Cleaning. I can use a Brush, a Hammer, and a Vacuum.

We also know which tools can be needed for each chore:

- The Brush and the Vacuum are used for Cleaning.
- The Brush and the Screw Gun are used for Decorating.
- The Hammer, the Vacuum and the Screw Gun are used for Repairs.

From this information, we can build the table of chores each flatmate can actually do with the tools they know.

Initial Table Design

Flat_Chores

Name	**Tool**	**Chore**
Alex	Brush	Cleaning
Alex	Vacuum	Cleaning
Alex	Vacuum	Repairs
Mia	Hammer	Repairs
Mia	Screw Gun	Decorating
Mia	Screw Gun	Repairs
Mia	Vacuum	Repairs
Sam	Brush	Cleaning
Sam	Brush	Decorating
Sam	Vacuum	Cleaning

There is no MVD, the table is in 4NF.

Initial Table Design

Sam: I am available for Decorating and Cleaning. I can use a Brush, a Hammer, and a Vacuum.

Flat_Chores

Name	Tool	Chore
Alex	Brush	Cleaning
Alex	Vacuum	Cleaning
Alex	Vacuum	Repairs
Mia	Hammer	Repairs
Mia	Screw Gun	Decorating
Mia	Screw Gun	Repairs
Mia	Vacuum	Repairs
Sam	Brush	Cleaning
Sam	Brush	Decorating
Sam	Vacuum	Cleaning

There is no MVD, the table is in 4NF. But the table does not state that Sam can use the Hammer.

Decomposition into Pairwise Relations

At the beginning, we were given three pieces of information, so we should have built 3 tables:

Prefered_Chores

Name	Chore
Alex	Cleaning
Alex	Repairs
Mia	Decorating
Mia	Repairs
Sam	Cleaning
Sam	Decorating

Can_Use

Name	Tool
Alex	Brush
Alex	Vacuum
Mia	Hammer
Mia	Screw Gun
Mia	Vacuum
Sam	Brush
Sam	Hammer
Sam	Vacuum

Can_Be_Used_For

Tool	Chore
Brush	Cleaning
Brush	Decorating
Hammer	Repairs
Screw Gun	Repairs
Screw Gun	Decorating
Vacuum	Cleaning
Vacuum	Repairs

When needed, we retrieve the information by joining the tables.

5NF

Join Dependency

A table can be losslessly reconstructed from several smaller tables.

Some tables represent facts that come from **several independent relationships**. All pairwise joins look correct, but the full table may still contain redundancy or missing combinations.

5NF – Projection-Join Normal Form

Every non-trivial join dependency is a superkey.

It is the final normal form as far as removing redundancy is concerned.

Ternary Relations

⚠ **Ternary relationships** may still exist

Decomposing such a table into pairwise relations can create **spurious rows**.

Flat_Chores

Name	Tool	Chore
Alex	Brush	Cleaning
Alex	Vacuum	Repairs
Mia	Hammer	Repairs
Mia	Screw Gun	Decorating
Mia	Screw Gun	Repairs
Mia	Vacuum	Repairs
Sam	Brush	Decorating
Sam	Vacuum	Cleaning

Ternary Relations

⚠ **Ternary relationships** may still exist

Decomposing such a table into pairwise relations can create **spurious rows**.

Prefered_Chores

Name	Chore
Alex	Cleaning
Alex	Repairs
Mia	Decorating
Mia	Repairs
Sam	Decorating
Sam	Cleaning

Can_Use

Name	Tool
Alex	Brush
Alex	Vacuum
Mia	Hammer
Mia	Screw Gun
Mia	Vacuum
Sam	Brush
Sam	Vacuum

Can_Be_Used_For

Tool	Chore
Brush	Decorating
Brush	Cleaning
Hammer	Repairs
Vacuum	Cleaning
Vacuum	Repairs
Screw Gun	Repairs
Screw Gun	Decorating

Ternary Relations

⚠ **Ternary relationships** may still exist

Decomposing such a table into pairwise relations can create **spurious rows**.

Flat_Chores

Name	**Tool**	**Chore**
Alex	Brush	Cleaning
Alex	Vacuum	Cleaning
Alex	Vacuum	Repairs
Mia	Hammer	Repairs
Mia	Screw Gun	Decorating
Mia	Screw Gun	Repairs
Mia	Vacuum	Repairs
Sam	Brush	Decorating
Sam	Brush	Cleaning
Sam	Vacuum	Cleaning

Summary (Normal Forms)

Edgar Codd (inventor of the relational model) proposed the theory of data normalization, through **normal forms**:

- 1NF: Atomic values.
- 2NF: No partial dependencies.
- 3NF/BCNF: No transitive dependencies.
- 4NF: No multivalued dependencies.
- 5NF: No join dependencies.
- And more (no longer about redundancy)

Core Idea

Decompose tables into **smaller, related tables**, such that each table represent **one specific topic**.

Reduces database modification anomalies (appropriate for OLTP systems).

Data Denormalization

Data Modeling for a Commercial Platform

A normalized schema: **Customers**, **Orders**, **Products**.

Customers

CustomerID	Name	City
C01	Alex	Paris
C02	Sam	Lyon
C03	Mia	Lille

Products

ProductID	Product	UnitPrice
P01	Coffee Machine	50
P02	Mug	5
P03	Tea Box	6

Orders

OrderID	CustomerID	Date
O01	C01	2025-11-01
O02	C02	2025-11-02
O03	C03	2025-11-03

Order_Products

OrderID	ProductID	Quantity
O01	P01	1
O01	P02	2
O02	P03	3
O03	P01	1

Each fact is stored once and the structure prevents contradictions.

Why Would We Ever *Not* Normalize?

Normalization protects against inconsistencies and modification anomalies. In principle, we would like every table to be normalized.

Why Would We Ever *Not* Normalize?

Normalization protects against inconsistencies and modification anomalies. In principle, we would like every table to be normalized.

But... in practice, some databases are **denormalized**.

Denormalization

Intentional modification of a normalized database in a way that violates previously maintained normal forms.

Why Would We Ever *Not* Normalize?

Normalization protects against inconsistencies and modification anomalies. In principle, we would like every table to be normalized.

But... in practice, some databases are **denormalized**.

Denormalization

Intentional modification of a normalized database in a way that violates previously maintained normal forms.

Why would a designer deliberately accept redundancy and inconsistency risk?

Denormalization due to External Integration

When integrating data from other platforms (secondary data), the source may provide a single flat table. We have no control over it.

Orders

Customer	City	Order Date	Product	Unit Price	Quantity
Alex	Paris	2025-11-01	Coffee Machine	50	1
Alex	Paris	2025-11-01	Mug	5	2
Sam	Lyon	2025-11-02	Tea Box	6	3
Mia	Lille	2025-11-03	Coffee Machine	50	1

Denormalization due to External Integration

When integrating data from other platforms (secondary data), the source may provide a single flat table. We have no control over it.

Orders

Customer	City	Order Date	Product	Unit Price	Quantity
Alex	Paris	2025-11-01	Coffee Machine	50	1
Alex	Paris	2025-11-01	Mug	5	2
Sam	Lyon	2025-11-02	Tea Box	6	3
Mia	Lille	2025-11-03	Coffee Machine	50	1

⚠ If the source contains inconsistencies, loading it into the normalized schema may fail.

Sometimes denormalization is **not a choice**, but a constraint of data integration.

Denormalization from Evolving Rules

The source database could have been normalized. Why is it not?

Denormalization from Evolving Rules

The source database could have been normalized. Why is it not?

A normalized schema assumes certain rules:

- Each product has a single official price.

Denormalization from Evolving Rules

The source database could have been normalized. Why is it not?

A normalized schema assumes certain rules:

- Each product has a single official price.
- What about discounts, promotions or dynamic pricing?

Orders

Customer	City	Order Date	Product	Unit Price	Quantity
Alex	Paris	2025-11-01	Coffee Machine	50	1
Alex	Paris	2025-11-01	Mug	5	2
Sam	Lyon	2025-11-02	Tea Box	6	3
Mia	Lille	2025-11-03	Coffee Machine	40	1

⚠ Rule changes make the strictly normalized design obsolete.

Sometimes denormalization is **a design choice**, for anticipating changes.

Purposes of Denormalization

Why denormalize a database?

Denormalization for Practical Considerations

1. Data arrives from a system that is not normalized.
2. Business rules evolve and invalidate a normalized design.

Purposes of Denormalization

Why denormalize a database?

Denormalization for Practical Considerations

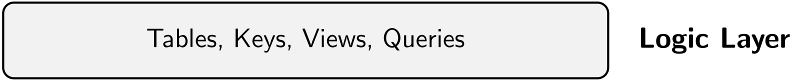
1. Data arrives from a system that is not normalized.
2. Business rules evolve and invalidate a normalized design.

Denormalization for Performance

3. To make queries faster.

To understand it, we need some knowledge of how relational database technology works at a high level.

Two Layers in a Relational DBMS



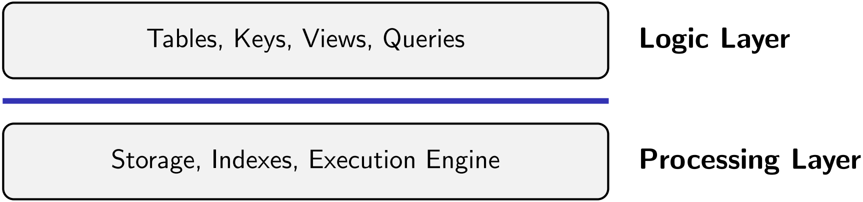
Tables, Keys, Views, Queries

Logic Layer

Logic Layer

- What you **see and interact with**.
- Guarantees correctness of answers.

Two Layers in a Relational DBMS



Tables, Keys, Views, Queries

Logic Layer

Storage, Indexes, Execution Engine

Processing Layer

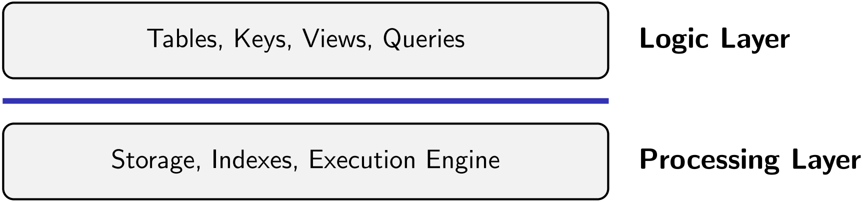
Logic Layer

- What you **see and interact with**.
- Guarantees correctness of answers.

Processing Layer

- Controls **physical organization** of data.
- Handles processing of operations

Two Layers in a Relational DBMS



Tables, Keys, Views, Queries

Logic Layer

Storage, Indexes, Execution Engine

Processing Layer

Logic Layer

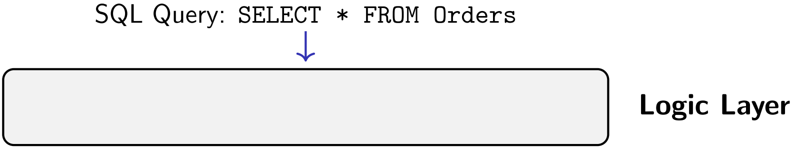
- What you **see and interact with**.
- Guarantees correctness of answers.

Processing Layer

- Controls **physical organization** of data.
- Handles processing of operations

Performance issues should be addressed in the **processing layer**.

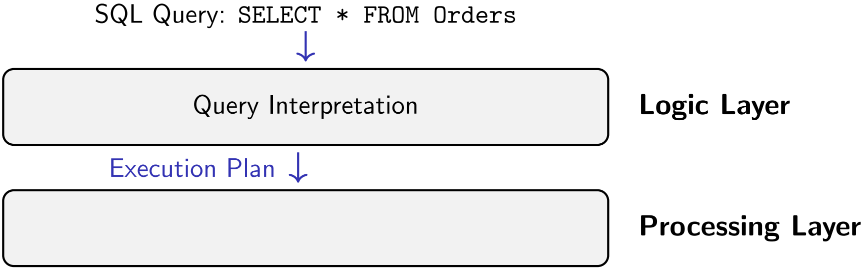
Query Handling



SQL Query: `SELECT * FROM Orders`

Logic Layer

Query Handling



SQL Query: `SELECT * FROM Orders`

Query Interpretation

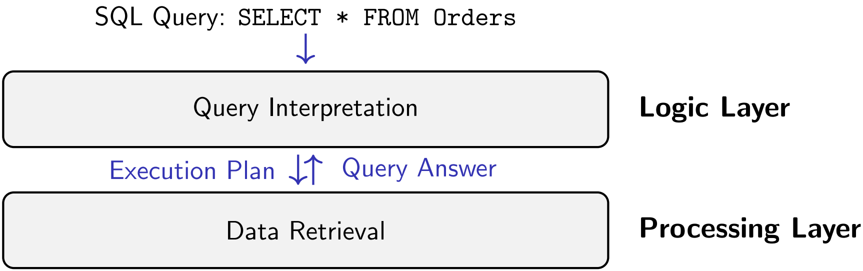
Logic Layer

Execution Plan

Processing Layer

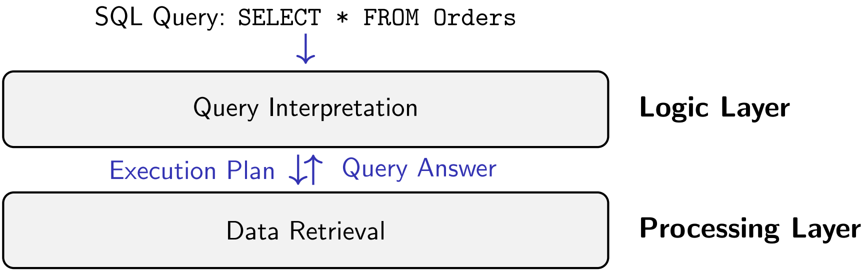
- The **logic layer** receives and interprets the query.

Query Handling



- The **logic layer** receives and interprets the query.
- The **processing layer** optimizes and executes it.

Query Handling



SQL Query: `SELECT * FROM Orders`

Query Interpretation

Logic Layer

Execution Plan ↓↑ Query Answer

Data Retrieval

Processing Layer

- The **logic layer** receives and interprets the query.
- The **processing layer** optimizes and executes it.
- Storage change (indexes, caching) can happen **without changing the logical design**.

Denormalization is **not** the first solution for performance!

Query with JOIN

The JOIN Problem

Joins combine records from multiple tables, which can be **slow**.

Query with JOIN

The JOIN Problem

Joins combine records from multiple tables, which can be **slow**.

Table A
Joined Result
Table B

Query with JOIN

The JOIN Problem

Joins combine records from multiple tables, which can be **slow**.

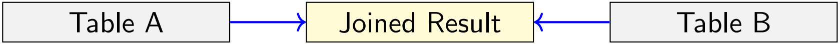


Table A

Joined Result

Table B

Solution: Physical Optimization

- Optimize the **processing layer**:
  - Indexes
  - Statistics for the query optimizer
  - Caching
  - Parallelization
- Data is stored in a **pre-joined form** under the hood.
- The **logic layer** remains unchanged: tables stay normalized.

When Physical Fixes Are Not Possible

What if the DBMS only provides **limited control** over the processing layer?

Denormalization may be the **only option** for:

- Acceptable performance
- Simplified reads

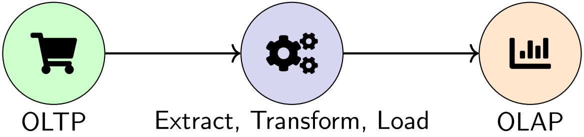
⚠ Inconsistencies become possible, and updates become slower.

Read-Only and Analytical Databases

- Data is **read often, updated rarely**.
- Denormalized structures (e.g., wider tables, precomputed summaries) can improve performance.

Read-Only and Analytical Databases

- Data is **read often, updated rarely**.
- Denormalized structures (e.g., wider tables, precomputed summaries) can improve performance.



- Behind the scenes, the operational source may remain normalized (OLTP).
- A loading process transfers the data into a denormalized reporting structure (OLAP).

Summary (Denormalization)

There are three recurring situations:

1. Data arrives from a **non-normalized system**.
2. Business rules evolve, invalidating a **normalized design**.
3. Performance concerns lead to **pre-joined or duplicated data**.

Denormalization can violate **any normal form** (5NF, 4NF, 3NF, 2NF, 1NF).

- Reduces query complexity **but sacrifices consistency**.
- Best suited for **read-heavy contexts** (e.g., OLAP systems).

Conclusion

Takeaways: Normalization and Denormalization

Normalization

1. Eliminates redundancy and ensures **data integrity**.
2. **1NF → 2NF → 3NF → BCNF → 4NF → 5NF**.
3. Solves update, deletion, and insertion anomalies.
4. Best for **OLTP systems** (frequent writes).

Denormalization

1. **Intentional violation** of normal forms for practical or performance reasons.
2. Best for **OLAP systems** (frequent reads).

Normalize for **consistency**, denormalize for **performance**.