Data, Data Storage, Data Collection

Lecture 9: Data Normalization and Denormalization

Romain Pascual

MICS, CentraleSupélec, Université Paris-Saclay

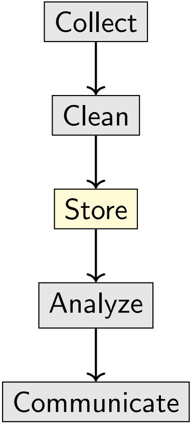
Recap

Recap: NoSQL

1. NoSQL databases address challenges of distributed systems and semi-structured data
2. The **aggregate** is the fundamental unit of organization in NoSQL databases
3. Data modeling is crucial for performance in NoSQL databases
4. Four main families of NoSQL databases: Key-value, Document-oriented, Column-oriented, Graph databases
5. NoSQL databases complement rather than replace relational databases (polyglot persistence)

Introduction

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

Bad Data

Flatmate Expenses

Expense	Category	Amount	Paid by	Year	Month
Pizza	Food	25	Sam	2025	11
Internet	Utilities	40	Alex	2025	11
Coffee	Groceries	12	Mia	2025	11
Time Machine	Utilities	250	Bill	2025	11

Bad Data

Flatmate Expenses

Expense	Category	Amount	Paid by	Year	Month
Pizza	Food	25	Sam	2025	11
Internet	Utilities	40	Alex	2025	11
Coffee	Groceries	12	Mia	2025	11
Time Machine	Utilities	250	Bill	2025	11

⚠ The **Time Machine** entry is likely erroneous.

Bad data can arise from:

- missing data,
- input errors,
- ...

Remember the lectures about data collection.

Data Integrity

Flatmate Expenses

Expense	Category	Amount	Paid by	Year	Month
Pizza	Food	25	Sam	2025	11
Internet	Utilities	40	Alex	2025	11
Coffee	Groceries	12	Mia	2025	11
Dishwasher	Utilities	250	Bill	2025	11

Data Integrity

Flatmate Expenses

Expense	Category	Amount	Paid by	Year	Month
Pizza	Food	25	Sam	2025	11
Internet	Utilities	40	Alex	2025	11
Coffee	Groceries	12	Mia	2025	11
Dishwasher	Utilities	250	Bill	2025	11
Pizza	Food	25	Alex	2025	11
Internet	Utilities	40	Bill	2025	11

Data Integrity

Flatmate Expenses

ExpenseID	Expense	Category	Price	Paid by	Year	Month
1842	Pizza	Food	25	Sam	2025	11
9571	Internet	Utilities	40	Alex	2025	11
6511	Coffee	Groceries	12	Mia	2025	11
3427	Dishwasher	Utilities	250	Bill	2025	11
8689	Pizza	Food	25	Alex	2025	11
9571	Internet	Utilities	40	Bill	2025	11

Good database design can protect against some cases of bad data: when it describes something that cannot be logicaly true.

- The **Internet** expense appears twice paid by different people.
- This is a **failure of data integrity**: the data contradicts itself.

When data disagrees with itself, it is not “just” a problem of bad data, but a problem of bad database design (**normalization**).

Data Normalization

Data Normalization

Definition (Normalization)

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

Flatmate Expenses

ExpenseID	Expense	Category	Price	Paid by	Year	Month
1842	Pizza	Food	25	Sam	2025	11
9571	Internet	Utilities	40	Alex	2025	11
6511	Coffee	Groceries	12	Mia	2025	11
3427	Dishwasher	Utilities	250	Bill	2025	11
8689	Pizza	Food	25	Alex	2025	11
9571	Internet	Utilities	40	Bill	2025	11

Two rows claim to describe the expense (9571) but disagree on **Paid by**.

⚠ The previous table is not allowed as it violates the idea of a **single version of the truth**: the same fact appears twice with conflicting values.

The Normal Forms

- How do we normalize?
- When have we normalized "enough"?

The Normal Forms

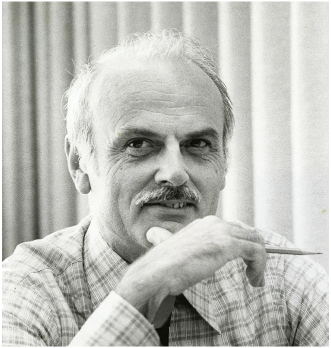
- How do we normalize?
- When have we normalized "enough"?

Normal Forms

Criteria to assess redundant data.

- Proposed by **Edgar F. Codd**[1,2] (relational model)
- Progressive levels of data integrity

1NF → 2NF → 3NF → 4NF → 5NF



Edgar F. Codd
(1923-2003)

[1] Edgar Codd. "A Relational Model of Data for Large Shared Data Banks". In: **Commun. ACM** 13.6 (1970)

[2] Edgar Codd. **Further Normalization of the Data Base Relational Model**. Tech. rep. IBM Research Report RJ909. 1971

Shared Items in the Flat

Who owns what in the flat?

Flat_Members_Items

Name	**Age**	**Shared Items**
Alex	23	Microwave, 2 Lamp, 2 Chairs
Mia	20	Blender, 2 Chairs, Table
Bill	26	Gaming Console
Sam	21	3 Chairs, TV , Lamp

Shared Items in the Flat

Who owns what in the flat?

Flat_Members_Items

Name	**Age**	**Shared Items**
Alex	23	Microwave, 2 Lamp, 2 Chairs
Mia	20	Blender, 2 Chairs, Table
Bill	26	Gaming Console
Sam	21	3 Chairs, TV , Lamp

Is the **Shared Items** cell describing one thing or several?

Repeating Group

A **repeating group** is a set of values in a single cell. It might create issues beacuse of:

- Varying types
- Number of values

Handle with Strings

Who owns what in the flat?

Can we store repeating groups as a comma-separated string?

Flat_Members_Items

Name	**Age**	**Shared Items**
Alex	23	“Microwave, 2 Lamp, 2 Chairs”
Mia	20	“Blender, 2 Chairs, Table”
Bill	26	“Gaming Console”
Sam	21	“3 Chairs, TV , Lamp”

Handle with Strings

Who owns what in the flat?

Can we store repeating groups as a comma-separated string?

Flat_Members_Items

Name	Age	Shared Items
Alex	23	"Microwave, 2 Lamp, 2 Chairs"
Mia	20	"Blender, 2 Chairs, Table"
Bill	26	"Gaming Console"
Sam	21	"3 Chairs, TV , Lamp"

- ⚠ **Querying is difficult**: How to search for "2 Chairs"?
- ⚠ **Updating is error-prone**: How to modify individual items?

Multiple Columns

Who owns what in the flat?

Can we use multiple columns for each item?

Flat_Members_Items

Name	Age	Item 1	Qty 1	Item 2	Qty 2	Item 3	Qty 3
Alex	23	Microwave	1	Lamp	2	Chairs	2
Mia	20	Blender	1	Chairs	2	Table	1
Bill	26	Gaming Console	1				
Sam	21	Chairs	3	TV	1	Lamp	1

The number of columns depends on the person, not on the data model.

Multiple Columns

Who owns what in the flat?

Can we use multiple columns for each item?

Flat_Members_Items

Name	**Age**	**Item 1**	**Qty 1**	**Item 2**	**Qty 2**	**Item 3**	**Qty 3**
Alex	23	Microwave	1	Lamp	2	Chairs	2
Mia	20	Blender	1	Chairs	2	Table	1
Bill	26	Gaming Console	1				
Sam	21	Chairs	3	TV	1	Lamp	1

The number of columns depends on the person, not on the data model.

- **Disorganized**: How to read and maintain the order?
- **Sparse**: How to handle person having various number of items?
- **Wide**: How well does it scale?

Multiple Columns

Who owns what in the flat?

Can we use multiple columns for each item?

Flat_Members_Items

Name	Age	Item 1	Qty 1	Item 2	Qty 2	Item 3	Qty 3
Alex	23	Microwave	1	Lamp	2	Chairs	2
Mia	20	Blender	1	Chairs	2	Table	1
Bill	26	Gaming Console	1				
Sam	21	Chairs	3	TV	1	Lamp	1

The number of columns depends on the person, not on the data model.

- **Disorganized**: How to read and maintain the order?
- **Sparse**: How to handle person having various number of items?
- **Wide**: How well does it scale?

Storing a repeating group of data in a single row violates 1NF.

First Normal Form (1NF)

1NF – Codd (1970)

Each field must contain **atomic (indivisible)** values:

- No collections (e.g., lists, sets).
- No nested tables.

By construction most relational DBMS (including standard SQL) do not support tables with non-atomic values.

Codd considered 1NF mandatory for relational databases, while the other normal forms were merely guidelines for database design.

Extensions of 1NF

If 1NF is consider the basis for integration in a RDBMS, we might consider other properties to be part of 1NF:

- Unique datatype within a column (compare with spreadsheets)
- Mandatory primary keys

Extensions of 1NF

If 1NF is consider the basis for integration in a RDBMS, we might consider other properties to be part of 1NF:

- Unique datatype within a column (compare with spreadsheets)
- Mandatory primary keys

1NF – Christopher Date (2007)

1. There is no specific top-to-bottom ordering of the rows.
2. There is no specific left-to-right ordering of the columns.
3. There are no duplicate rows.
4. Every intersection of a row and a column contains exactly one value from the applicable domain and nothing else.

Achieving 1NF

We need to store the items in a separate rows instead.

Flat_Items

Name	Item	Qty
Alex	Microwave	1
Alex	Lamp	2
Alex	Chairs	2
Mia	Blender	1
Mia	Chairs	2
Mia	Table	1
Bill	Gaming Console	1
Sam	Chairs	3
Sam	TV	1
Sam	Lamp	1

Achieving 1NF

We need to store the items in a separate rows instead.

Flat_Items

Name	Item	Qty
Alex	Microwave	1
Alex	Lamp	2
Alex	Chairs	2
Mia	Blender	1
Mia	Chairs	2
Mia	Table	1
Bill	Gaming Console	1
Sam	Chairs	3
Sam	TV	1
Sam	Lamp	1

Each person now corresponds to several rows.

Achieving 1NF

We need to store the items in a separate rows instead.

Flat_Items

Name	Item	Qty
Alex	Microwave	1
Alex	Lamp	2
Alex	Chairs	2
Mia	Blender	1
Mia	Chairs	2
Mia	Table	1
Bill	Gaming Console	1
Sam	Chairs	3
Sam	TV	1
Sam	Lamp	1

Each person now corresponds to several rows.

Primary key is now a combination of **Name** and **Item**: it expresses one fact per row.

Adding the Age?

Suppose we want to add the age to the table (new column).

Flat_Info

Name	**Item**	**Qty**	**Age**
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Bill	Gaming Console	1	26
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

- The table is still in 1NF

⚠ **Data duplication**: **Age** is repeated for each item.

Deletion Anomaly

Bill sells his console

Flat_Info

Name	Item	Qty	Age
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Bill	Gaming Console	1	26
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Deletion Anomaly

Bill sells his console, and we delete the entry from the table.

Flat_Info

Name	Item	Qty	Age
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
~~Bill~~	~~Gaming Console~~	~~1~~	~~26~~
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Deletion Anomaly

- Deleting Bill's gaming console **deletes Bill's age as well**.

Update Anomaly

It is Alex's birthday who turns 24, we need to update the table.

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Update Anomaly

It is Alex's birthday who turns 24, we need to update the table.

Flat_Info

Name	Item	Qty	Age
Alex	Microwave	1	24
Alex	Lamp	2	24
Alex	Chairs	2	24
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Update Anomaly

- If we miss any, the database becomes **inconsistent**.
- Now Alex is both 23 and 24 in the database!

Update Anomaly

It is Alex's birthday who turns 24, we need to update the table.

Flat_Info

Name	**Item**	**Qty**	**Age**
Alex	Microwave	1	24
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Insertion Anomaly

Tim joins the flat, but does not bring any new item.

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21

Insertion Anomaly

Tim joins the flat, but does not bring any new item.

Flat_Info

Name	Item	Qty	Age
Alex	Microwave	1	23
Alex	Lamp	2	23
Alex	Chairs	2	23
Mia	Blender	1	20
Mia	Chairs	2	20
Mia	Table	1	20
Sam	Chairs	3	21
Sam	TV	1	21
Sam	Lamp	1	21
Tim			22

Insertion Anomaly

- We can not add Tim without an item (primary key).

2NF

2NF – Informal Definition

Every non-key attribute must depend on the entire primary key: there is no part key dependency.

Flat_Info

<u>Name</u>	<u>Item</u>	Qty	Age
⋮	⋮	⋮	⋮

- Does **Qty** depend on the entire primary key?
- Does **Age** depend on the entire primary key?

2NF

2NF – Informal Definition

Every non-key attribute must depend on the entire primary key: there is no part key dependency.

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
⋮	⋮	⋮	⋮

- Does **Qty** depend on the entire primary key? **Yes**
- Does **Age** depend on the entire primary key?

2NF

2NF – Informal Definition

Every non-key attribute must depend on the entire primary key: there is no part key dependency.

Flat_Info

Name	Item	Qty	Age
⋮	⋮	⋮	⋮

- Does **Qty** depend on the entire primary key? **Yes**
- Does **Age** depend on the entire primary key? **No**

Issue appeared when we added **Age** to the table, where is did not really belong.

Attributes, Keys and Dependencies

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
⋮	⋮	⋮	⋮

To (formally) discuss 2NF and 3NF, we need to understand how a table is uniquely identified, so we need some additional notions.

- Superkeys
- Candidate keys
- Prime attribute
- Functional Dependency

Superkey

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
⋮	⋮	⋮	⋮

Superkey

A set of attributes that can uniquely identify each row.

⚠ It may contain unnecessary attributes.

Shared Flatmate Superkeys

- {**Name**, **Item**},
- {**Name**, **Item**, **Qty**},
- {**Name**, **Item**, **Age**},
- {**Name**, **Item**, **Qty**, **Age**}

Candidate Key

Flat_Info

<u>Name</u>	<u>Item</u>	Qty	Age
⋮	⋮	⋮	⋮

Candidate Key

A **minimal** superkey.

Removing any attribute breaks uniqueness.

Shared Flatmate Candidate Keys

Only candidate key: {**Name**, **Item**}.

Prime Attribute

Flat_Info

<u>Name</u>	<u>Item</u>	Qty	Age
⋮	⋮	⋮	⋮

Prime Attribute

An attribute that is part of **at least one** candidate key.

Every other attribute is **non-prime**.

Shared Flatmate Prime Attributes

Non-prime attributes: **Qty** and **Age**

Functional Dependencies

Flat_Info

<u>**Name**</u>	<u>**Item**</u>	**Qty**	**Age**
⋮	⋮	⋮	⋮

Definition (Functional Dependencies (FD) $X \to Y$)

For each value in X, there is a unique of value in Y that depends only on the value in X.

The projection on $X \times Y$ of the relation is a function: knowing the values for X is enough to know the for Y.

Shared Flatmate Functional Dependencies

- $\{$**Name**, **Item**$\} \to \{$**Qty**$\}$
- $\{$**Name**$\} \to \{$**Age**$\}$

2NF, Formally

2NF – Formal Definition

No non-prime attribute depends on part of a candidate key.

Shared Flatmate Dependencies

Only candidate key: $\{$**Name**, **Item**$\}$.

Non-prime attributes: **Qty** and **Age**.

Functional dependencies:

- $\{$**Name**, **Item**$\} \rightarrow \{$**Qty**$\}$
- $\{$**Name**$\} \rightarrow \{$**Age**$\}$

Issue appeared when we added **Age** to the table, where is did not really belong.

Achieving 2NF

Split into two tables:

Flat_Items

Name	**Item**	**Qty**
Alex	Microwave	1
Alex	Lamp	2
Alex	Chairs	2
Mia	Blender	1
Mia	Chairs	2
Mia	Table	1
Bill	Gaming Console	1
Sam	Chairs	3
Sam	TV	1
Sam	Lamp	1

Flat_Members

Name	**Age**
Alex	23
Mia	20
Bill	26
Sam	21

No part-key dependency: 2NF.

Adding Rent Information

We add **RoomSize** and **Rent** to the database.

Flat_Members

<u>**Name**</u>	**Age**	**RoomSize**	**Rent**
Alex	23	Large	500
Mia	20	Small	300
Bill	26	Large	500
Sam	21	Medium	400

Dependencies (✓ 2NF):

- $\{$**Name**$\} \rightarrow \{$**Age**$\}$
- $\{$**Name**$\} \rightarrow \{$**RoomSize**$\}$
- $\{$**Name**$\} \rightarrow \{$**Rent**$\}$

Room Swap

Bill is often out of the flat and swaps room with Mia.

Flat_Members

Name	Age	RoomSize	Rent
Alex	23	Large	500
Mia	20	Large	300
Bill	26	Small	500
Sam	21	Medium	400

What happened? Dependencies:

- $\{\textbf{Name}\} \rightarrow \{\textbf{Age}\}$
- $\{\textbf{Name}\} \rightarrow \{\textbf{RoomSize}\}$
- $\{\textbf{RoomSize}\} \rightarrow \{\textbf{Rent}\}$

Room Swap

Bill is often out of the flat and swaps room with Mia.

Flat_Members

Name	**Age**	**RoomSize**	**Rent**
Alex	23	Large	500
Mia	20	Large	300
Bill	26	Small	500
Sam	21	Medium	400

The table assumes that **Rent** depends on **Name**, but the data shows it depends on **RoomSize**.

What happened? Dependencies: (TD: Transitive Dependency)

- $\{$**Name**$\} \rightarrow \{$**Age**$\}$
- $\{$**Name**$\} \rightarrow \{$**RoomSize**$\}$
- $\{$**RoomSize**$\} \rightarrow \{$**Rent**$\}$
- $\{$**Name**$\} \rightarrow \{$**RoomSize**$\} \rightarrow \{$**Rent**$\}$: TD

Achieving 3NF

Split the table to eliminate transitive dependencies.

Flat_Members

<u>Name</u>	Age	RoomSize
Alex	23	Large
Mia	20	Small
Bill	26	Large
Sam	21	Medium

Room_Rent

<u>RoomSize</u>	Rent
Small	300
Medium	400
Large	500

Each table represents a single entity type.

3NF

3NF

Every non-prime attribute depends directly on every candidate key, and not on part of a candidate key or other non-prime attributes.

Every non-key attribute depends on the key, the whole key and nothing but the key.

Flat_Members

<u>Name</u>	Age	RoomSize
⋮	⋮	⋮

Room_Rent

<u>RoomSize</u>	Rent
⋮	⋮

Dependencies (✓ 3NF):

- $\{$**Name**$\} \rightarrow \{$**Age**$\}$
- $\{$**Name**$\} \rightarrow \{$**RoomSize**$\}$
- $\{$**RoomSize**$\} \rightarrow \{$**Rent**$\}$

BCNF

Boyce-Codd Normal Form

For every functional dependency $X \rightarrow Y$, X is a superkey.

Every attribute depends on the key,
the whole key and nothing but the key.

Most 3NF tables are also in BCNF.

BCNF fixes a loophole in 3NF and eliminates remaining anomalies caused by functional dependencies when multiple candidate keys overlap.

Member Profiles: Hobbies and Languages

Flat_Profiles

Name	Hobby	Language
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French
Mia	Painting	English
Mia	Reading	English
Mia	Painting	Spanish
Mia	Reading	Spanish
Bill	Gaming	English
Bill	Gaming	German

Member Profiles: Hobbies and Languages

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French
Alex	Cooking	Finnish
Mia	Painting	English
Mia	Reading	English
Mia	Painting	Spanish
Mia	Reading	Spanish
Bill	Gaming	English
Bill	Gaming	German

Update Anomaly

- Alex also speaks Finnish!

Member Profiles: Hobbies and Languages

Flat_Profiles

Name	Hobby	Language
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French
Alex	Cooking	Finnish
Mia	Painting	English
Mia	Reading	English
Mia	Painting	Spanish
Mia	Reading	Spanish
Bill	Gaming	English
Bill	Gaming	German

Update Anomaly

- Alex also speaks Finnish!
- We should also have (Alex, Cycling, Finnish).

Dependencies

Flat_Profiles

Name	Hobby	Language
⋮	⋮	⋮

Something about the table design allows for impossible situations.

Dependencies

Flat_Profiles

<u>Name</u>	Hobby	Language
⋮	⋮	⋮

Something about the table design allows for impossible situations.

What are the non-trivial functional dependencies?

Dependencies

Flat_Profiles

Name	Hobby	Language
⋮	⋮	⋮

Something about the table design allows for impossible situations.

What are the non-trivial functional dependencies? There are none.

Dependencies

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
⋮	⋮	⋮

Something about the table design allows for impossible situations.

What are the non-trivial functional dependencies? There are none.

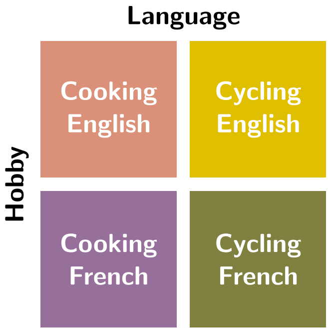
The problem is not a functional dependency, but the opposite: too much independence between attributes.

Multivalued Dependencies

Language

Hobby	**Cooking English**	**Cycling English**
	Cooking French	**Cycling French**

Multivalued Dependencies



Multivalued Dependencies

- Each member has a **set of hobbies**
- Each member has a **set of languages**
- These sets are **independent** of each other

This is called a **multivalued dependency**.

Understanding Multivalued Dependencies

Definition (Multivalued Dependency (MVD) $X \twoheadrightarrow Y$)

For each value in X, there is a set of value in Y that depends only on the value in X: knowing X fixes the possible Ys.

Understanding Multivalued Dependencies

Definition (Multivalued Dependency (MVD) $X \twoheadrightarrow Y$)

For each value in X, there is a set of value in Y that depends only on the value in X: knowing X fixes the possible Ys.

Formal Characterisation of MVDs

A relation $R(X, Y, Z)$ satisfies $X \twoheadrightarrow Y$ if for every x, y_1, z_1, y_2, z_2:

$$R(x, y_1, z_1) \wedge R(x, y_2, z_2) \implies R(x, y_1, z_2) \wedge R(x, y_2, z_1)$$

For each x, the values in Y and Z form a Cartesian product.

Example

Flat_Profiles

<u>Name</u>	Hobby	Language
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French

✓ $\{\textbf{Name}\} \twoheadrightarrow \{\textbf{Hobby}\}$

- (Alex, Cooking, English) and (Alex, Cycling, French) in the table
- (Alex, Cooking, French) and (Alex, Cycling, English) also in the table

Example

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French
Alex	Cooking	Finnish

✓ $\{\textbf{Name}\} \twoheadrightarrow \{\textbf{Hobby}\}$

- (Alex, Cooking, English) and (Alex, Cycling, French) in the table
- (Alex, Cooking, French) and (Alex, Cycling, English) also in the table

Example

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
Alex	Cooking	English
Alex	Cycling	English
Alex	Cooking	French
Alex	Cycling	French
Alex	Cooking	Finnish
Alex	Cycling	Finnish

✓ $\{$**Name**$\} \twoheadrightarrow \{$**Hobby**$\}$

- (Alex, Cooking, English) and (Alex, Cycling, French) in the table
- (Alex, Cooking, French) and (Alex, Cycling, English) also in the table

Once we have established the MVD, (Alex, Cooking, Finnish) and (Alex, Cycling, English) implies (Alex, Cycling, Finnish).

Is the table in 3NF?

What are the candidate keys?

Flat_Profiles

Name	**Hobby**	**Language**
⋮	⋮	⋮

- Knowing {**Name**} alone gives several tuples (different hobbies/languages),

Is the table in 3NF?

What are the candidate keys?

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
⋮	⋮	⋮

- Knowing {**Name**} alone gives several tuples (different hobbies/languages),
- Knowing {**Name**, **Hobby**} still gives several tuples (different languages),

Is the table in 3NF?

What are the candidate keys?

Flat_Profiles

<u>**Name**</u>	**Hobby**	**Language**
⋮	⋮	⋮

- Knowing {**Name**} alone gives several tuples (different hobbies/languages),
- Knowing {**Name**, **Hobby**} still gives several tuples (different languages),
- Knowing {**Name**, **Language**} still gives several tuples (different hobbies).

Is the table in 3NF?

What are the candidate keys?

Flat_Profiles

<u>**Name**</u>	<u>**Hobby**</u>	<u>**Language**</u>
⋮	⋮	⋮

- Knowing {**Name**} alone gives several tuples (different hobbies/languages),
- Knowing {**Name**, **Hobby**} still gives several tuples (different languages),
- Knowing {**Name**, **Language**} still gives several tuples (different hobbies).

So the key is {**Name**, **Hobby**, **Language**}.

✓ As a result, the tabel is trivially in 3NF.

4NF

4NF

For every (non-trivial) multivalued dependencies $X \twoheadrightarrow Y$, X is a superkey.

4NF

4NF

For every (non-trivial) multivalued dependencies $X \twoheadrightarrow Y$, X is a superkey.

The key is {**Name**, **Hobby**, **Language**}.

The MVDs are:

- {**Name**} $\twoheadrightarrow$ {**Hobby**}
- {**Name**} $\twoheadrightarrow$ {**Language**}

{**Name**} is not the key, so the multivalued dependencies are not on the key.

Achieving 4NF

As always, the fix it the split the table into multuple tables.

Member_Hobbies

Name	Hobby
Alex	Cooking
Alex	Cycling
Mia	Painting
Mia	Reading
Bill	Gaming
Sam	Swimming

Member_Language

Name	Language
Alex	English
Alex	French
Mia	English
Mia	Spanish
Bill	English
Bill	German
Sam	English

We still have MVD, but they become trivial since they cover all attributes.

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**.

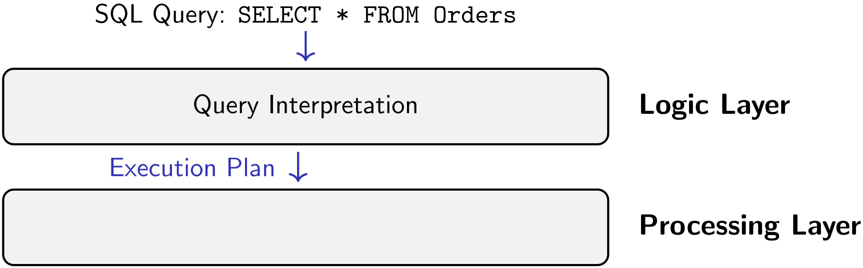
Query Handling

SQL Query: `SELECT * FROM Orders`

↓

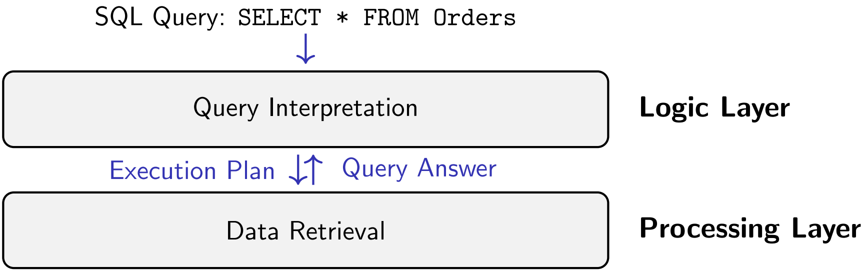
Logic Layer

Query Handling



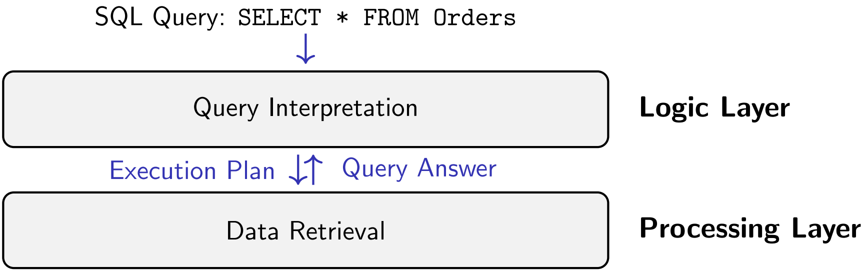
- The **logic layer** receives and interprets the query.

Query Handling



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

Query Handling



- 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**.

Query with JOIN

The JOIN Problem

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

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

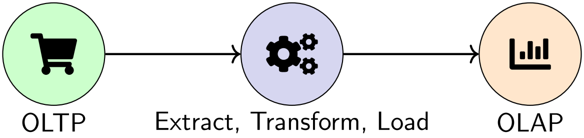
⚠ Inconsistencies become possible, and updates become slower.

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**.