NORMALIZATION

Objective

- Normalization presents a set of rules that tables and databases must follow to be well structured.
- Historically presented as a sequence of normal forms

Normalization in DBMS (Database Management System) is the process of organizing the data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller, related tables and defining relationships between them.

Why Do We Need Normalization?

1.Eliminate Redundancy: It reduces duplication of data, which saves storage space and ensures consistency.

2.Improve Data Integrity: It ensures that data is logically stored, making it easier to update, insert, and delete records without causing anomalies.

3.Optimize Queries: Properly normalized databases can lead to more efficient queries and better performance.

Consider a table Students

Student-ID	Student-Name	Course1	Course2	Course3
1	Alice	Math	Physics	NULL
2	Bob	Math	NULL	NULL
3	Carol	Chemistry	Math	Physics

Issues with the Above Table:

- **Data Redundancy:** The course names are repeated for each student.
- **Update Anomalies:** If the course name "Math" needs to be updated, it has to be changed in multiple places.
- **Insertion Anomalies:** Adding a new course requires adding new columns.
- **Deletion Anomalies:** If a student drops all courses, we lose the course information.

Normalization (to 1st Normal Form - 1NF):

We split the table into two related tables:

Student-ID	StudentName
1	Alice
2	Bob
3	Carol

Student-Courses Table:

Student-ID	Course
1	Math
1	Physics
2	Math
3	Chemistry
3	Math
3	Physics

Benefits of Normalization:

- **No Redundancy:** Course names are not duplicated.
- **Easier Updates:** Changing "Math" to "Mathematics" requires updating only one record.
- **Scalable:** Adding new courses doesn’t require altering the table structure.

Real-Life Example of Normalization
Library Book Management:
Imagine a library where you need to keep track of books and the authors who wrote them.

•**Without Normalization:** You might have a list where each book is recorded along with its authors. If a book has multiple authors, you'd list the book multiple times, once for each author. This means the book's title, publication date, and other details are repeated.

•**Problems:**

- **Redundancy:** The same book details are stored multiple times.
- **Updates:** If you need to correct a book's title, you'd have to find and update each occurrence.
- **Inserts:** Adding a new author for an existing book might require copying all book details again.

•**With Normalization:** You separate the information into two lists—one for books and one for authors. Each book is listed only once, and authors are linked to the books they wrote.

- **Benefits:**
  - **Efficiency: The book's details are stored in one place, reducing storage space and the risk of errors.**
  - **Simplicity: Updating a book’s details is easier since you only need to do it once.**
  - **Flexibility: You can add or remove authors without affecting the book's information.**

First Normal From

- A table is in the first normal form iff
  - The domain of each attribute contains only ***atomic values***, and
  - The value of each attribute contains only a ***single value*** from that domain.

In layman's terms. it means every column of your table should only contain *single values*

Example

- For a library

Patron ID	Borrowed books
C45	B33, B44, B55
C12	B56

1-NF Solution

Patron ID	Borrowed book
C45	B33
C45	B44
C45	B33
C12	B56

Example

- For an airline

Flight	Weekdays
UA59	Mo We Fr
UA73	Mo Tu We Th Fr

1NF Solution

Flight	Weekday
UA59	Mo
UA59	We
UA59	Fr
UA73	Mo
UA73	We
...	...

Implication for the ER model

- Watch for entities that can have multiple values for the same attribute
  - Phone numbers, …
- What about course schedules?
  - MW 5:30-7:00pm
    - Can treat them as *atomic time slots*

Functional dependency

Let X and Y be *sets* of attributes in a table *T*

- Y is **functionally dependent** on *X* in *T* **iff** for each set x ∈ R.X there is precisely one corresponding set y∈ R.Y
- Y is **fully functional dependent** on *X* in *T* if *Y* is functional dependent on X and Y is not functional dependent on any proper subset of *X*

Example

- Book table

BookNo	Title	Author	Year
B1	Moby Dick	H. Melville	1851
B2	Lincoln	G. Vidal	1984

Author attribute is:

- ***functionally dependent*** on the pair { BookNo, Title}
- ***fully functionally dependent*** on BookNo

Why it matters

- table BorrowedBooks

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

Address attribute is

- ***functionally dependent*** on the pair { BookNo, Patron}
- ***fully functionally dependent*** on Patron

Problems

- Cannot insert new patrons in the system until they have borrowed books
  - ***Insertion anomaly***
- Must update all rows involving a given patron if he or she moves.
  - ***Update anomaly***
- Will lose information about patrons that have returned all the books they have borrowed
  - ***Deletion anomaly***

Armstrong inference rules (1974)

- ***Axioms:***
  - Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
  - Augmentation: if $X \rightarrow Y$, then $WX \rightarrow WY$
  - Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- ***Derived Rules:***
  - Union: if $X \rightarrow Y$ and $X \rightarrow Z$, the $X \rightarrow YZ$
  - Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity: if $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$

Armstrong inference rules (1974)

- Axioms are both
  - ***Sound:***
    when applied to a set of functional dependencies they only produce dependency tables that belong to the transitive closure of that set
  - ***Complete:***
    can produce all dependency tables that belong to the transitive closure of the set

Armstrong inference rules (1974)

- Three last rules can be derived from the first three (the axioms)
- Let us look at the ***union rule***:
  if $X \rightarrow Y$ and $X \rightarrow Z$, the $X \rightarrow YZ$
- Using the first three axioms, we have:
  - if $X \rightarrow Y$, then $XX \rightarrow XY$ same as $X \rightarrow XY$ (2^{nd})
  - if $X \rightarrow Z$, then $YX \rightarrow YZ$ same as $XY \rightarrow YZ$ (2^{nd})
  - if $X \rightarrow XY$ and $XY \rightarrow YZ$, then $X \rightarrow YZ$ (3^{rd})

Second Normal Form

- A table is in 2NF iff
  - It is in 1NF and
  - no non-prime attribute is dependent on any proper subset of any candidate key of the table
- A ***non-prime attribute*** of a table is an attribute that is not a part of any candidate key of the table
- A ***candidate key*** is a minimal superkey

Example

- Library allows patrons to request books that are currently out

BookNo	Patron	PhoneNo
B3	J. Fisher	555-1234
B2	J. Fisher	555-1234
B2	M. Amer	555-4321

Example

- Candidate key is {BookNo, Patron}
- We have
  - Patron → PhoneNo
- Table is not 2NF
  - Potential for
    - Insertion anomalies
    - Update anomalies
    - Deletion anomalies

2NF Solution

- Put telephone number in separate Patron table

BookNo	Patron
B3	J. Fisher
B2	J. Fisher
B2	M. Amer

Patron	PhoneNo
J. Fisher	555-1234
M. Amer	555-4321

Third Normal Form

- A table is in 3NF iff
  - it is in 2NF and
  - all its attributes are determined only by its candidate keys and not by any non-prime attributes

Example

- Table BorrowedBooks

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

- Candidate key is BookNo
- Patron → Address

3NF Solution

- Put address in separate Patron table

BookNo	Patron	Due
B1	J. Fisher	3/2/15
B2	L. Perez	2/28/15

Patron	Address
J. Fisher	101 Main Street
L. Perez	202 Market Street

Another example

- Tournament winners

Tournament	Year	Winner	DOB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 Sept. 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975

- Candidate key is {Tournament, Year}
- Winner →DOB

Boyce-Codd Normal Form

- Stricter form of 3NF
- A table *T* is in BCNF iff
  - for every one of its non-trivial dependencies $X \rightarrow Y$, X is a super key for T
- Most tables that are in 3NF also are in BCNF

Example

Manager	Project	Branch
Alice	Alpha	Austin
Alice	Delta	Austin
Carol	Alpha	Houston
Dean	Delta	Houston

- We can assume
  - Manager → Branch
  - {Project, Branch} → Manager

Example

Manager	Project	Branch
Alice	Alpha	Austin
Bob	Delta	Houston
Carol	Alpha	Houston
Alice	Delta	Austin

- Not in BCNF because Manager → Branch and Manager is not a superkey
- Will decomposition work?

A decomposition (I)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston

- Two-table solution does not preserve the dependency {Project, Branch} → Manager

A decomposition (II)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta
Dean	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston
Dean	Houston

- Cannot have two or more managers managing the same project at the same branch

Multivalued dependencies

- Assume the column headings in a table are divided into three disjoint groupings *X*, *Y*, and *Z*
- For a particular row, we can refer to the data beneath each group of headings as *x*, *y*, and *z* respectively

Multivalued dependencies

- A ***multivalued dependency*** X =>Y occurs if
  - For any x_c actually occurring in the table and the list of all the $x_c yz$ combinations that occur in the table, we will find that x_c is associated with the same y entries regardless of z.
- A ***trivial multivalued dependency*** *X* =>*Y* is one where either
  - *Y* is a subset of *X*, or
  - *Z* is empty (*X* ∪ *Y* has all column headings)

Fourth Normal Form

- A table is in 4NF iff
  - For every one of its non-trivial multivalued dependencies *X* => *Y*, *X* is either:
    - A candidate key or
    - A superset of a candidate key

Example from Wikipedia

Restaurant	Pizza	DeliveryArea
Pizza Milano	Thin crust	SW Houston
Pizza Milano	Thick crust	SW Houston
Pizza Firenze	Thin crust	NW Houston
Pizza Firenze	Thick crust	NW Houston
Pizza Milano	Thin crust	NW Houston
Pizza Milano	Thick crust	NW Houston

Discussion

- The table has no non-key attributes
  - Key is { Restaurant, Pizza, DeliveryArea}
- Two non-trivial multivalued dependencies
  - Restaurant => Pizza
  - Restaurant => DeliveryArea

  since each restaurant delivers the same pizzas to all its delivery areas

4NF Solution

Restaurant	DeliveryArea
Pizza Milano	SW Houston
Pizza Firenze	NW Houston
Pizza Milano	NW Houston

- Two separate tables

Restaurant	Pizza
Pizza Milano	Thin crust
Pizza Milano	Thick crust
Pizza Firenze	Thin crust
Pizza Firenze	Thick crust

Join dependency

- A table *T* is subject to a ***join dependency*** if it can always be recreated by ***joining*** multiple tables each having a subset of the attributes of T

- The join dependency is said to be ***trivial*** if one of the tables in the join has all the attributes of the table *T*
- *Notation: *{ A, B, …}* on *T*

Fifth normal form

- A table *T* is said to be 5NF iff
  - Every non-trivial join dependency in it is implied by its candidate keys

- A join dependency *{A, B, … Z} on T is implied by the candidate key(s) of T if and only if each of A, B, …, Z is a superkey for T

An example

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops

- Note that Circuit City sells Apple tablets and phones but only Toshiba laptops

A very bad decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

- Let see what happens when we do a natural join

The result of the join

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops
CompUSA	Toshiba	Laptops

- Introduces two spurious tuples

A different table

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops

- Assume now that any store carrying a given brand and selling a product that is made by that brand will ***always*** carry that product

The same decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

- Let see what happens when we do a natural join

The result of the join

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops
CompUSA	Toshiba	Laptops

- Still one spurious tuple

The right decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

Store	*Brand*
Circuit City	Apple
Circuit City	Toshiba
CompUSA	Apple

Conclusion

- The first "big" table was 5NF
- The second table was decomposable

Lossless Decomposition

General Concept

- If R(A, B, C) satisfies $A \rightarrow B$
  - We can project it on A,B and A,C ***without losing information***
  - Lossless decomposition

- $R = \pi_{AB}(R) \bowtie \pi_{AC}(R)$
  - $\pi_{AB}(R)$ is the projection of R on AB
  - $\bowtie$ is the natural join operator

Example

R

Course	*Instructor*	*Text*
4330	Paris	none
4330	Cheng	none
3330	Hillford	Patterson & Hennessy

- Observe that Course → Text

A lossless decomposition

$\pi_{\text{Course, Text}}(R)$

Course	*Text*
4330	none
3330	Patterson & Hennessy

$\pi_{\text{Course, Instructor}}(R)$

Course	*Instructor*
4330	Paris
4330	Cheng
3330	Hillford

A different case

R

Course	*Instructor*	*Text*
4330	Paris	Silberschatz and Peterson
4330	Cheng	none
3330	Hillford	Patterson & Hennessy

- Now Course → Text
- R cannot be decomposed

A lossy decomposition

$\pi_{\text{Course, Text}}(R)$

Course	*Text*
4330	none
4330	Silberschatz & Peterson
3330	Patterson & Hennessy

$\pi_{\text{Course, Instructor}}(R)$

Course	*Instructor*
4330	Paris
4330	Cheng
3330	Hillford

An Example

Normalisation Example

- We have a table representing orders in an online store
- Each row represents an item on a particular order
- Primary key is {Order, Product}

- Columns
  - Order
  - Product
  - Quantity
  - UnitPrice
  - Customer
  - Address

Functional Dependencies

- Each order is for a single customer:
  - Order → Customer
- Each customer has a single address
  - Customer → Address
- Each product has a single price
  - Product → UnitPrice
- As Order → Customer and Customer → Address
  - Order → Address

2NF Solution (I)

- ***First decomposition***
  - First table

Order	Product	Quantity	UnitPrice

  - Second table

Order	Customer	Address

2NF Solution (II)

- ***Second decomposition***
  - First table

Order	Product	Quantity

  - Second table

Order	Customer	Address

  - Third table

Product	UnitPrice

3NF

- In second table

Order	Customer	Address

  - Customer → Address
- Split second table into

Order	Customer

Customer	Address

Normalisation to 2NF

- Second normal form means no partial dependencies on candidate keys
  - {Order} → {Customer, Address}
  - {Product} → {UnitPrice}

- To remove the first FD we project over

  {Order, Customer, Address} (R1)

and

{Order, Product, Quantity, UnitPrice} (R2)

Normalisation to 2NF

- R1 is now in 2NF, but there is still a partial FD in R2

  {Product} → {UnitPrice}

- To remove this we project over

{Product, UnitPrice} (R3)

and

{Order, Product, Quantity} (R4)

Normalisation to 3NF

- R has now been split into 3 relations - R1, R3, and R4
  - R3 and R4 are in 3NF
  - R1 has a transitive FD on its key
- To remove

  {Order} → {Customer} → {Address}
- we project R1 over
  - {Order, Customer}
  - {Customer, Address}

Normalisation

- 1NF:
  - {Order, Product, Customer, Address, Quantity, UnitPrice}
- 2NF:
  - {Order, Customer, Address}, {Product, UnitPrice}, and {Order, Product, Quantity}
- 3NF:
  - {Product, UnitPrice}, {Order, Product, Quantity}, {Order, Customer}, and {Customer, Address}