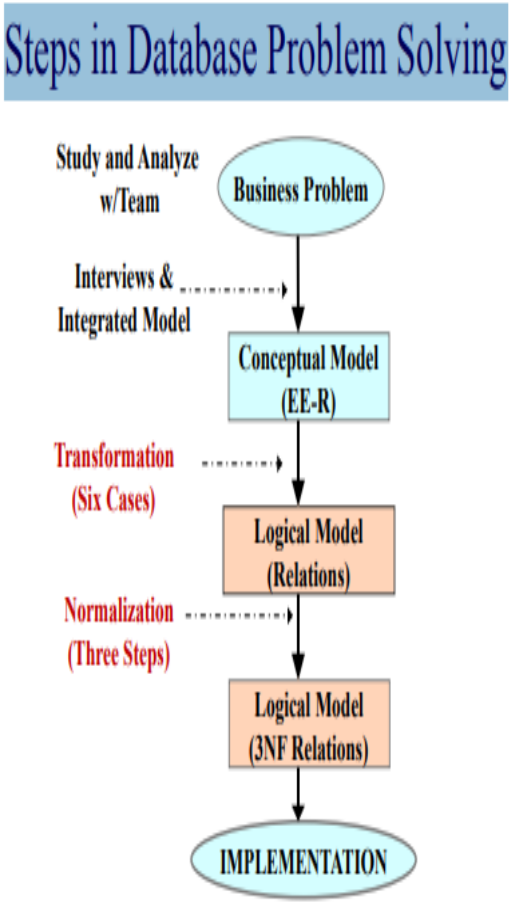
Lebanese University **Faculty of Media and Documentation** **- Section 1 -** **Computer Science** **First Year - Semester 1** **Database 1**		**الجامعة اللبنانية** **كلية الإعلام والتوثيق** **- الفرع الأول -** **Chapter 6: Normalization in DBMS**

1. Normalization.



Normalization in DBMS is a technique using which you can organize the data in the database tables so that:

- There is less repetition of data,
- A large set of data is structured into a bunch of smaller tables,
- and the tables have a proper relationship between them.

DBMS Normalization is a systematic approach to **decompose (break down) tables** to eliminate data redundancy(repetition) and undesirable characteristics like Insertion anomaly in DBMS, Update anomaly in DBMS, and Delete anomaly in DBMS.

It is a **multi-step process** that puts data into tabular form, removes duplicate data, and set up the relationship between tables.

Normalization is the process of efficiently organizing data in a database.

2. Benefits of using Normal forms.

Normalization is required for:

- Eliminating redundant(useless) data, therefore handling **data integrity**, because if data is repeated it increases the chances of inconsistent data.
- Normalization helps in keeping **data consistent** by storing the data in one table and referencing it everywhere else.
- Storage optimization although that is not an issue these days because Database storage is cheap.
- Breaking down large tables into smaller tables with relationships, so it makes the database structure more scalable and adaptable.
- Ensuring data dependencies make sense i.e. data is logically stored.

Normalization serves several key purposes:

- *Remove redundant data → Avoid duplicates and save storage.*
- *Ensure logical data dependencies → Keep relationships correct.*
- *Simplify schema maintenance → make the schema easier to update and evolve.*
- *Improve consistency and integrity → Data stays accurate and reliable.*
- *Prevent anomalies → Avoid issues during insert, update, or delete operations*

3. Problems without normalization

If a table is not properly normalized and has data redundancy(repetition) then it will not only **eat up extra memory space** but will also make it difficult for you to handle and update the data in the database, without losing data.

Insertion, Updation, and Deletion Anomalies are very frequent if the database is not normalized.

To understand these anomalies let us take an example of a **Student** table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

In the table above, we have data for four Computer Sci. students.

As we can see, data for the **fields branch, hod(Head of Department), and office_tel are repeated for the students who are in the same branch** in the college, this is **Data Redundancy.**

3.1. Insertion Anomaly in DBMS : adding new users forces user to create duplicate data

- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.
- Also, if we have to insert data for 100 students of the same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but Insertion anomalies.
- **If you have to repeat the same data in every row of data, it's better to keep the data separately and reference that data in each row (solution).**
- So in the above table, we can keep the branch information separately, and just use the branch_id in the student table, where branch_id can be used to get the branch information.

3.2. Updating Anomaly in DBMS

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

- What if Mr. X leaves the college? or Mr. X is no longer the HOD of the computer science department? In that case, all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.
- This is an Updation anomaly because you need to update all the records in your table just because one piece of information got changed.

3.3. Deletion Anomaly in DBMS

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

- In our Student table, two different pieces of information are kept together, the Student information and the Branch information.
- So if only a single student is enrolled in a branch, and that student leaves the college, or for some reason, the entry for the student is deleted, we will lose the branch information too.
- So never in DBMS, we should keep two different entities together, which in the above example is Student and branch,

The solution for all the three anomalies described above is to keep the student information and the branch information in two different tables. And use the branch_id in the student table to reference the branch.

Primary Key and Non-key attributes

Before we move on to learn different Normal Forms in DBMS, let's first understand what is a primary key and what are non-key attributes.

As you can see in the table above, the student_id column is a primary key because using the student_id value we can uniquely identify each row of data, hence the remaining columns then become the non-key attributes.

4. Normalization rules

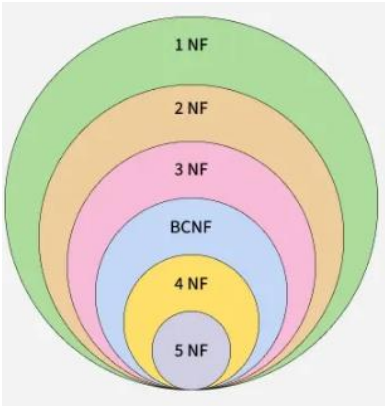
The Diagram below shows the hierarchy of database normal forms. Each inner circle represents a stricter level of normalization, starting from 1NF (basic structure) to 5NF (most refined). As you move inward, data redundancy reduces and data integrity improves.

Normalization rules divides larger tables into smaller tables and links them using relationships.

Normal forms are database normalization levels which determine the "goodness" of a table.

Normal forms

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)*
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)



➢The Theory of Data Normalization is still being developed further. For example, there are discussions even on 6th Normal Form. However, in most practical applications, normalization achieves its best in 3rd Normal Form.

3. 1. First Normal Form (1NF): Eliminating Duplicate Records = No multivalued attributes.

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. And the order in which data is stored should not matter.

- Each row is unique (i.e., no duplicate rows).

Example 1:

If we have an Employee table in which we store the employee information along with the employee skillset, the table will look like this:

emp_id	emp_name	emp_mobile	emp_skills
1	John Tick	9999957773	Python, JavaScript
2	Darth Trader	8888853337	HTML, CSS, JavaScript
3	Rony Shark	7777720008	Java, Linux, C++

The above table has 4 columns:

- All the columns have different names.
- All the columns hold values of the same type like emp_name has all the names, emp_mobile has all the contact numbers, etc.
- The order in which we save data doesn't matter
- **But the emp_skills column holds multiple comma-separated values, while as per the First Normal form, each column should have a single value.**

Hence the above table fails to pass the First Normal form.

So how do you fix the above table? There are two ways to do this:

1. Remove the emp_skills column from the Employee table and keep it in some other table.
2. Or add multiple rows for the employee and each row is linked with one skill.

1. Create Separate tables for Employee and Employee Skills

So the Employee table will look like this,

And the new Employee_Skill table:

emp_id	emp_name	emp_mobile
1	John Tick	9999957773
2	Darth Trader	8888853337
3	Rony Shark	7777720008

emp_id	emp_skill
1	Python
1	JavaScript
2	HTML
2	CSS
2	JavaScript
3	Java
3	Linux
3	C++

2. **Add Multiple rows for Multiple skills**

You can also simply add multiple rows to add multiple skills. This will lead to repetition of the data, but that can be handled as you further Normalize your data using the Second Normal form and the Third Normal form.

emp_id	emp_name	emp_mobile	emp_skill
1	John Tick	9999957773	Python
1	John Tick	9999957773	JavaScript
2	Darth Trader	8888853337	HTML
2	Darth Trader	8888853337	CSS
2	Darth Trader	8888853337	JavaScript
3	Rony Shark	7777720008	Java
3	Rony Shark	7777720008	Linux
3	Rony Shark	7777720008	C++

Example 2:

- A relation is in 1NF (first normal form) if :
  - it does not contain any composite attribute.
  - it does not contain any multi-valued attribute.

EmployeeID	FirstName	LastName	Children
10	Sami	Khalil	Leila, Karim 1/10/10, 5/9/12
15	Fadi	Salim	Leila. Rim, Chadi 12/12/15, 9/8/14, 7/4/17
17	Raed	Rami	Sarah 11/10/14

Problem: Composite and multi-valued attribute

EmployeeID	FirstName	LastName	Children-Name	Children-BirthDate
10	Sami	Khalil	Leila, Karim	1/10/10, 5/9/12
15	Fadi	Salim	Leila. Rim, Chadi	12/12/15, 9/8/14, 7/4/17
17	Raed	Rami	Sarah	11/10/14

Problem: multi-valued attribute

4. 2. Second Normal Form (2NF): Eliminating Partial Dependency

For a table to be in the Second Normal Form,

1. It should be in the **First Normal form.**

2. And, it should not have **Partial Dependency.**

What is Partial Dependency?

When a table has a primary key that is made up of two or more columns, then all the columns(not included in the primary key) in that table should depend on the entire primary key and not on a part of it. If any column(which is not in the primary key) depends on a part of the primary key then we say we have Partial dependency in the table.

Confused? Let's take an example.

If we have two tables Students and Subjects, to store student information and information related to subjects.

Student table:

student_id	student_name	branch
1	Akon	CSE
2	Bkon	Mechanical

Subject Table:

subject_id	subject_name
1	C Language
2	DSA
3	Operating System

And we have another table **Score** to store the marks scored by students in any subject like this,

student_id	subject_id	marks	teacher_name
1	1	70	Miss. C
1	2	82	Mr. D
2	1	65	Mr. OP

Now in the above table, the primary key is **student_id + subject_id**, because both these information are required to select any row of data.
But in the **Score** table, we have a column **teacher_name**, which depends on the subject information or just the **subject_id**, so we *should not keep* that information in the **Score** table.
The column **teacher_name** should be in the **Subjects** table. And then the entire system will be Normalized as per the Second Normal Form.

Updated **Subject** table:

subject_id	subject_name	teacher_name
1	C Language	Miss. C
2	DSA	Mr. D
3	Operating System	Mr. Op

Updated **Score** table:

student_id	subject_id	marks
1	1	70
1	2	82
2	1	65

4. 3. Third Normal Form (3NF): Eliminating Transitive Dependency

A table is said to be in the **Third Normal** Form when,

1. It satisfies the **First Normal Form** and the **Second Normal form.**
2. And, it doesn't have **Transitive Dependency.**

What is Transitive Dependency?

In a table we have some column that acts as the primary key and other columns depends on this column. But what if a column that is not the primary key depends on another column that is also not a primary key or part of it? Then we have Transitive dependency in our table.

Let's take an example. We had the **Score** table in the Second Normal Form above. If we have to store some extra information in it, like,

1. **exam_type**
2. **total_marks**

To store the type of exam and the total marks in the exam so that we can later calculate the percentage of marks scored by each student.

The **Score** table will look like this,

student_id	subject_id	marks	exam_type	total_marks
1	1	70	Theorv	100
1	2	82	Theorv	100
2	1	42	Practical	50

- In the table above, the column **exam_type** depends on both **student_id** and **subject_id**, because,
    - a student can be in the CSE branch or the Mechanical branch,
    - and based on that they may have different exam types for different subjects.
    - The CSE students may have both Practical and Theory for Compiler Design,
    - whereas Mechanical branch students may only have Theory exams for Compiler Design.
- But the column **total_marks** just depends on the **exam_type** column. And the **exam_type** column is not a part of the primary key. Because the primary key is **student_id + subject_id**, hence we have a Transitive dependency here.

How to Transitive Dependency?

You can create a separate table for **ExamType** and use it in the **Score** table.

New **ExamType** table,

exam_type_id	exam_type	total_marks	duration
1	Practical	50	45
2	Theory	100	180
3	Workshop	150	300

We have created a new table **ExamType** and we have added more related information in it like **duration**(duration of exam in mins.), and now we can use the **exam_type_id** in the **Score** table.

Next Chapter: Linear Algebra

FAQs Here are some frequently asked questions related to the normalization in DBMS.

Q. Why do we need Normalization in DBMS?

Database Normalization helps you design and structure your table properly so that you have proper relationships between tables. It helps you with the following:

1. Data Integrity
2. Data consistency
3. Better relationship between tables
4. More scalable design for tables.
5. No large tables, small tables with a proper relationship.
6. Removing dependencies, like Partial Dependency, Transitive Dependency, Join Dependency, etc.

Q. What are the different Normal Forms in DBMS?

Following are the different Database Normal Forms:

1. First Normal Form also known as 1NF
2. Second Normal Form or 2NF
3. Third Normal Form or 3NF
4. Boyce-Codd Normal Form or BCNF
5. Fourth Normal Form or 4NF
6. Fifth Normal Form or 5NF or PJNF (Project-Join Normal Form)

Q. What is a Primary Key in DBMS?

A **Primary key** is a column that can be used to uniquely identify each row in a table. It can be a single column, or it can be multiple columns together. Yes, a primary key can have two columns or even more than two columns in it.

Q. What are non-key attribute in a Table?

All the columns that are not a primary key or not a part of the primary key are called as non-Key columns in a Table.

For example, if we have a table **Students** with columns *student_id*, *student_name*, *student_address*, and *student_id* is the **primary key** in this table, then *student_name* and *student_address* will be the **non-Key attributes**.