

Database Management System

IT 261

Bachelor of Information Technology

Prepared by Dr. Khaled Eskaf

Lecture 10



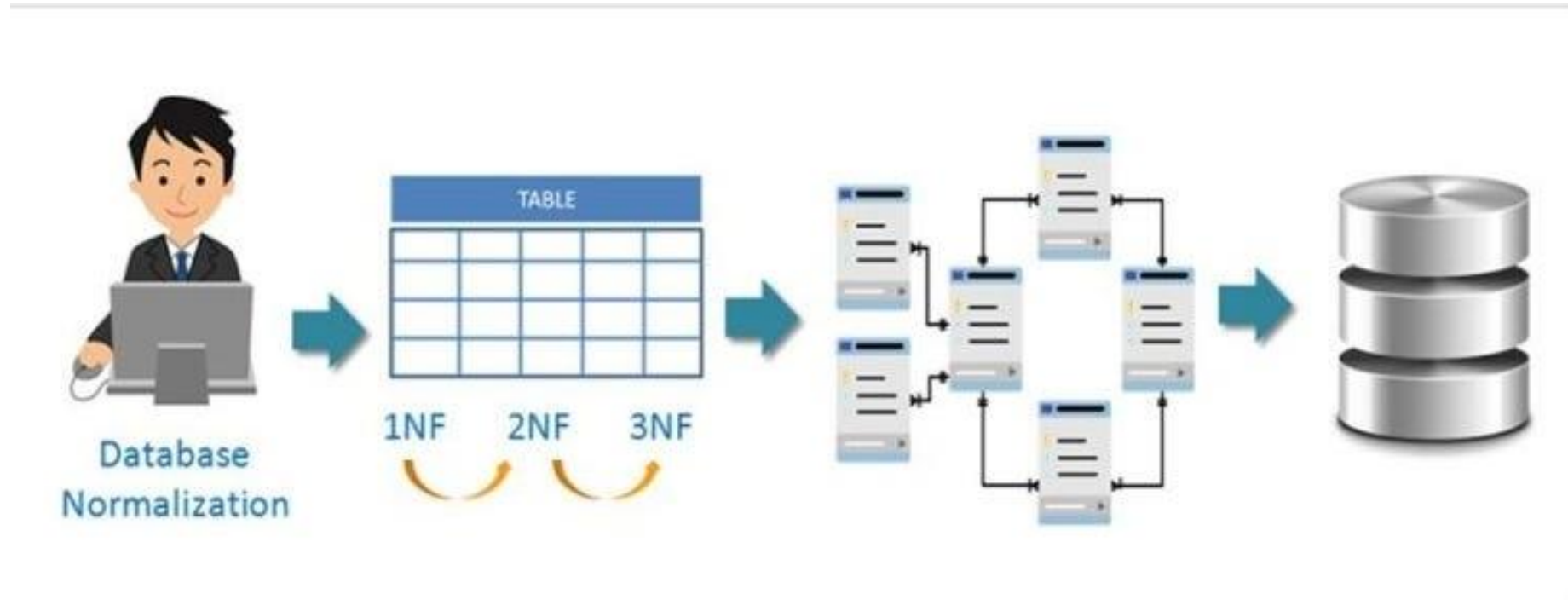


Revision

Database normalization

CHAPTER 9

Database normalization — Normal forms - 1nf 2nf 3nf



In relational database design, we not only want to create a structure that stores all of the data, but we also want to do it in a way that minimize potential errors when we work with the data. The default language for accessing data from a relational database is SQL. In particular, SQL can be used to manipulate data in the following ways: insert new data, delete unwanted data, and update existing data. Similarly, in an un-normalized design, there are 3 problems that can occur when we work with the data:

INSERT ANOMALY: This refers to the situation when it is impossible to insert certain types of data into the database.

DELETE ANOMALY: The deletion of data leads to unintended loss of additional data, data that we had wished to preserve.

UPDATE ANOMALY: This refers to the situation where updating the value of a column leads to database inconsistencies (i.e., different rows on the table have different values).

To address the 3 problems above, we go through the process of normalization. When we go through the normalization process, we increase the number of tables in the database, while decreasing the amount of data stored in each table. There are several different levels of database normalization:

- 1st Normal Form (1NF)
- 2nd Normal Form (2NF)
- 3rd Normal Form (3NF)

Normalization is the branch of relational theory that provides design insights. It is the process of determining how much redundancy exists in a table.

The goals of normalization are to:

- Be able to characterize the level of redundancy in a relational schema
- Provide mechanisms for transforming schemas in order to remove redundancy

Normal Forms

All the tables in any database can be in one of the normal forms we will discuss next.

Ideally we only want minimal redundancy for PK to FK. Everything else should be derived from other tables.

There are three basic normal forms, which are:

- First normal form (1NF)
 - Second normal form (2NF)
 - Third normal form (3NF)
 - Boyce-Codd normal form (BCNF)**
- BCNF is rarely used.**

Advantages of Normal Form

- **Reduced data redundancy:** Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.
- **Improved data consistency:** Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.
- **Simplified database design:** Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.
- **Improved query performance:** Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.
- **Easier database maintenance:** Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

Important Points Regarding Normal Forms in DBMS:

- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.
- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.
- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.

	1NF	2NF	3NF
Decomposition of Relation	R	R₁₁ R₁₂	R₂₁ R₂₂ R₂₃
Conditions	Eliminate Repeating Groups	Eliminate Partial Functional Dependency	Eliminate Transitive Dependency

First Normal Form (1NF)

The first normal form (1NF), is the most basic form of data normalization. The core outcome of this rule ensures that there are no repeating entries in a group. This means:

- Every cell should only have **one** single value.
- Every record should be unique.

For a table to be in the first normal form, it must meet the following criteria:

- A single cell must not hold more than one value (atomic)
- There must be a primary key for identification
- No duplicated rows or columns
- Each column must have only one value for each row in the table

Suppose you have a table that stores information about customers and their orders. This table is not in 1NF because the “Orders” column contains multiple values separated by commas, violating the rule of atomic values:

CustomerID	CustomerName	Orders
1	John	Order1, Order2
2	Alice	Order3

To bring it into 1NF, you can split the “Orders” column into separate rows:

CustomerID	CustomerName	Order
1	John	Order1
1	John	Order2
2	Alice	Order3

How do we bring an unnormalized table into first normal form? Consider the following example:

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

This table is not in first normal form because the [Color] column can contain multiple values. For example, the first row includes values "red" and "green."

1st Normal Form Example

To bring this table to first normal form, we split the table into two tables and now we have the resulting tables:

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

**TABLE_PRODUCT_PRICE**

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Second Normal Form (2NF)

	1NF	2NF	3NF
Decomposition of Relation	R	R₁₁ R₁₂	R₂₁ R₂₂ R₂₃
Conditions	Eliminate Repeating Groups	Eliminate Partial Functional Dependency	Eliminate Transitive Dependency

Second Normal Form (2NF)

2NF is the second normal form that builds on the rules of the first normal form. Again, the goal is to ensure that there are no repeating entries in a dataset. Entries with this data normalization rule applied must:

- Adhere to all the 1NF requirements.
- Have one primary key applied.

The application of one primary key essentially means that a **separate table needs to be created** for subsets of data that can be placed in multiple rows. The data in each table can then be linked with foreign key labels (numbers in this case).

Second Normal Form (2NF)

A table is in 2NF if it meets the following criteria:

- It is in 1NF.
- All non-key attributes (columns) are fully functionally dependent on the primary key.

Example 1:

Consider a table that stores information about courses and the instructors who teach them:

CourseID	Instructor	CourseName
101	Smith	Biology
102	Johnson	Chemistry
103	Smith	Physics

In this table, the combination of “CourseID” and “Instructor” forms a composite primary key. However, the “CourseName” column depends only on “CourseID,” not on “Instructor.” To bring this table into 2NF, you can split it into two separate tables:

CourseID	CourseName
101	Biology
102	Chemistry
103	Physics

Table 1 (Courses)

CourseID	Instructor
101	Smith
102	Johnson
103	Smith

Table 2 (Instructors)

Consider the following example:

Example Not In Second Normal Form

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

**TABLE_PURCHASE**

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

	1NF	2NF	3NF
Decomposition of Relation	R	R₁₁ R₁₂	R₂₁ R₂₂ R₂₃
Conditions	Eliminate Repeating Groups	Eliminate Partial Functional Dependency	Eliminate Transitive Dependency

Third Normal Form (3NF)

A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency. 3NF is used to reduce the data duplication. It is also used to achieve the data integrity. If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

- 1) X is a super key.
- 2) Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Third Normal Form (3NF)

A table is in 3NF if it meets the following criteria:

- It is in 2NF.
- There is no transitive dependency between non-key attributes.

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

Example:

Continuing with the “Courses” and “Instructors” tables from 2NF, suppose you add an “InstructorEmail” column to the “Instructors” table. If the email is unique to each instructor and not related to a course, you have a transitive dependency issue. To achieve 3NF, you would create a separate “InstructorInfo” table:

CourseID	CourseName
101	Biology
102	Chemistry
103	Physics

Table 1 (Courses): (Unchanged from 2NF)

CourseID	Instructor
101	Smith
102	Johnson
103	Smith

Table 2 (Instructors): (Without email)

Instructor	InstructorEmail
Smith	<u>smith@email.com</u>
Johnson	<u>johnson@email.com</u>

Table 3 (InstructorInfo): (Instructor and Email)

Example:

In the following table Roll_No is the candidate key which determines ExamType and ExamType determines the MaxMarks. So the following table holds transition property. That's why it is not in 3NF.

RollNo	Name	ExamType	MaxMarks
1	Ali	Viva	20
2	Ahmad	Theory	100
3	Akram	Practical	50
4	Nauman	Viva	20
5	Haider	Practical	50

Exam table

Transitive property can be removed by dividing the table into its parts.

RollNo	Name	ExamType	MaxMarks
1	Ali	Viva	20
2	Ahmad	Theory	100
3	Akram	Practical	50
4	Nauman	Viva	20
5	Haider	Practical	50

Exam table

Transitive Property Removed

JOIN			ExamType	MaxMarks
RollNo	Name	ExamType		
1	Ali	Viva	Viva	20
2	Ahmad	Theory	Theory	100
3	Akram	Practical	Practical	50
4	Nauman	Viva		
5	Haider	Practical		

Table: Exam

Table: Marks

3rd Normal Form Example 3:

Consider the following example:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table above, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

**TABLE_BOOK**

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

Transitively Dependent

Dependent

Dependent

<u>UnitCode</u>	UnitName	CourseCode	CourseName
UG45783	Advance Database	COMP2009	Computing
UG45832	Network Systems	COMP2009	Computing
UG45734	Multi-User Operating Systems	COMP2009	Computing
UG45951	Project	BUS22009	Business & Computing

Transitively Dependent

Dependent

Dependent

<u>UnitCode</u>	UnitName	CourseCode	CourseName
UG45783	Advance Database	COMP2009	Computing
UG45832	Network Systems	COMP2009	Computing
UG45734	Multi-User Operating Systems	COMP2009	Computing
UG45951	Project	BUS22009	Business & Computing

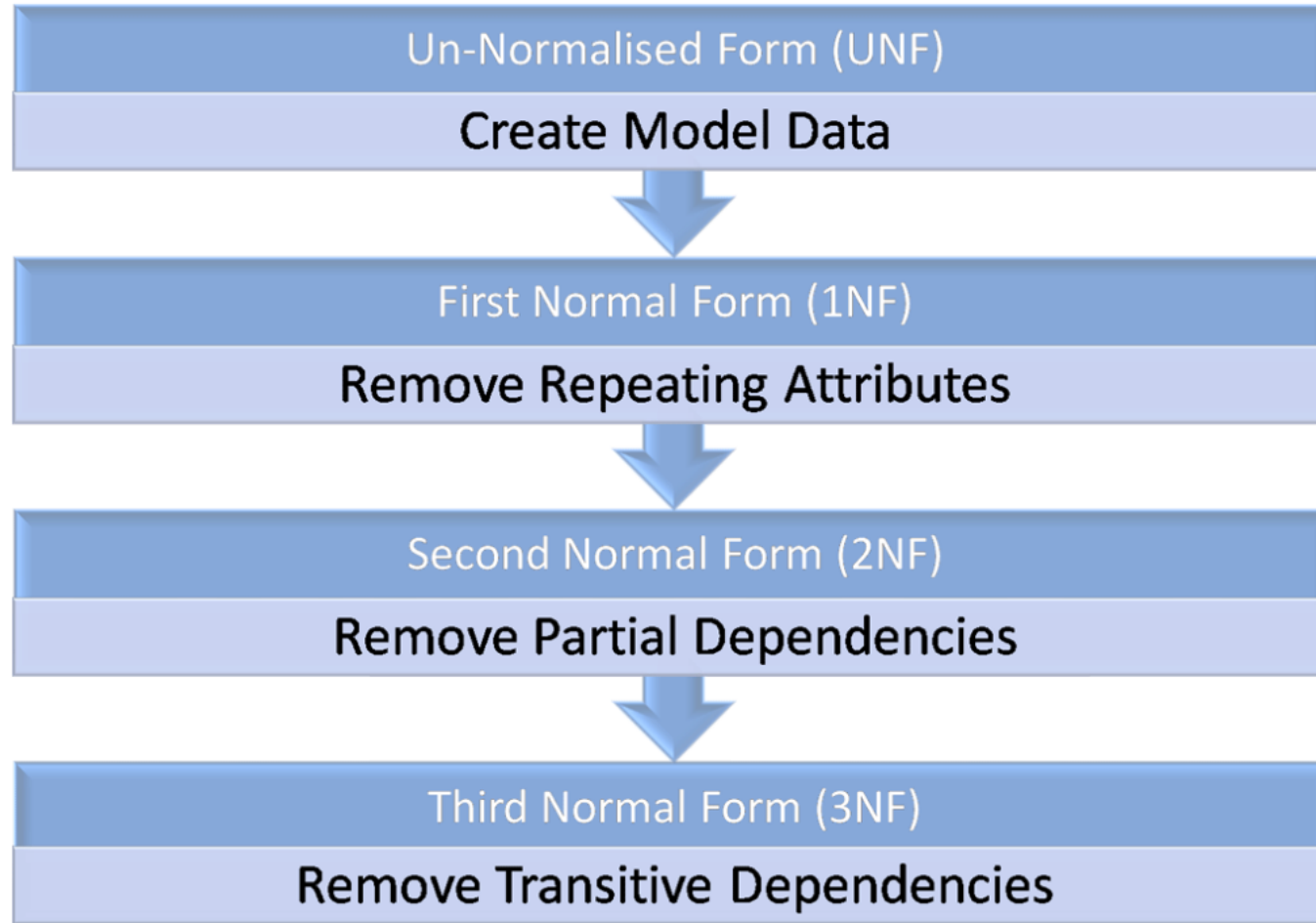
Foreign Key

Tables in Third Normal Form

Primary Key

<u>UnitCode</u>	UnitName	CourseCode*
UG45783	Advance Database	COMP2009
UG45832	Network Systems	COMP2009
UG45734	Multi-User Operating Systems	COMP2009
UG45951	Project	BUS22009

<u>CourseCode</u>	CourseName
COMP2009	Computing
BUS22009	Business & Computing



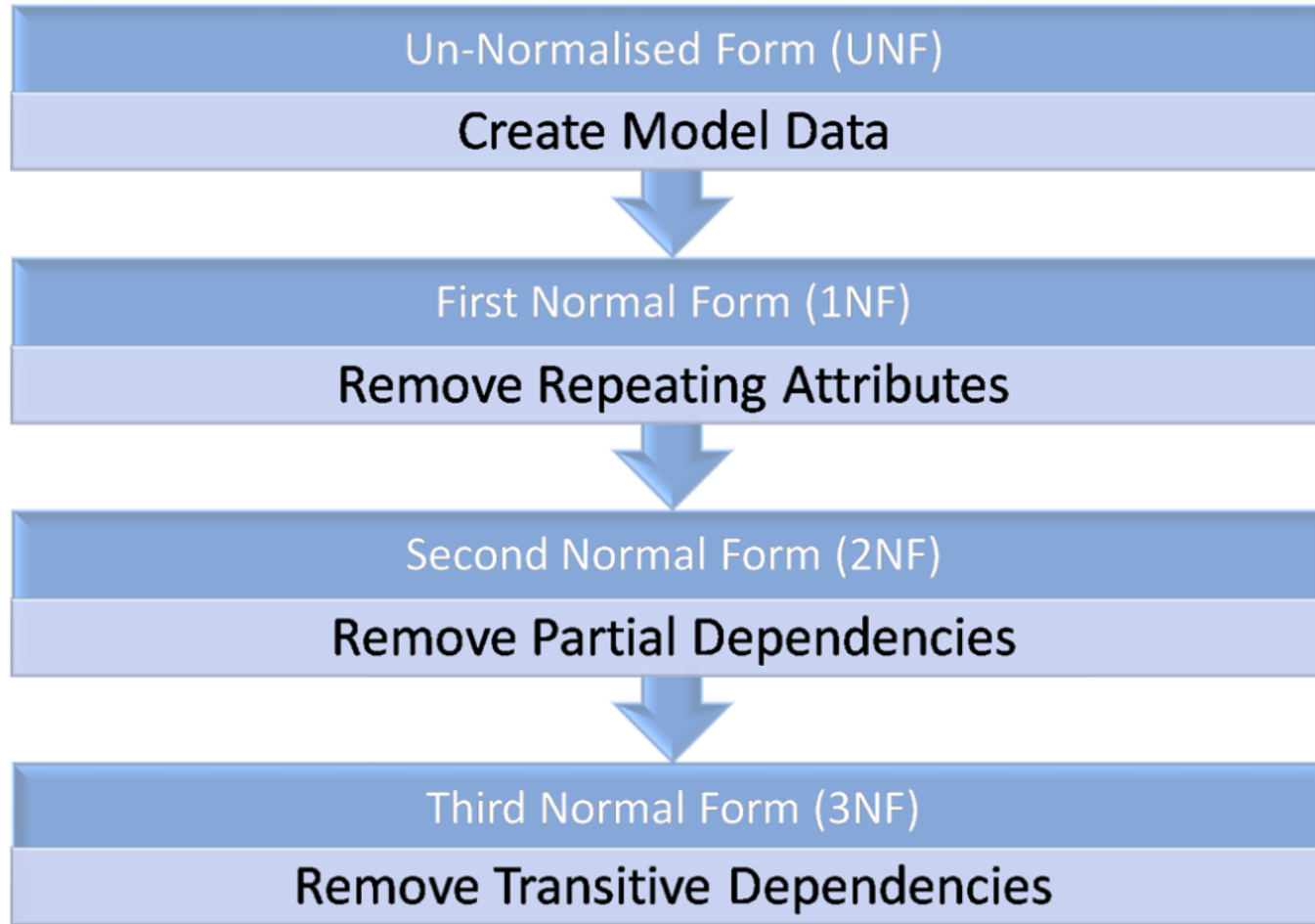
1.Un-Normalised Form (UNF) – Data Modelling

2.First Normal Form (1NF) – Repeating Groups

3.Second Normal Form (2NF) – Partial Dependencies

4.Third Normal Form (3NF) – Transitive Dependencies

CHAPTER 10



1.Un-Normalised Form (UNF) – Data Modelling

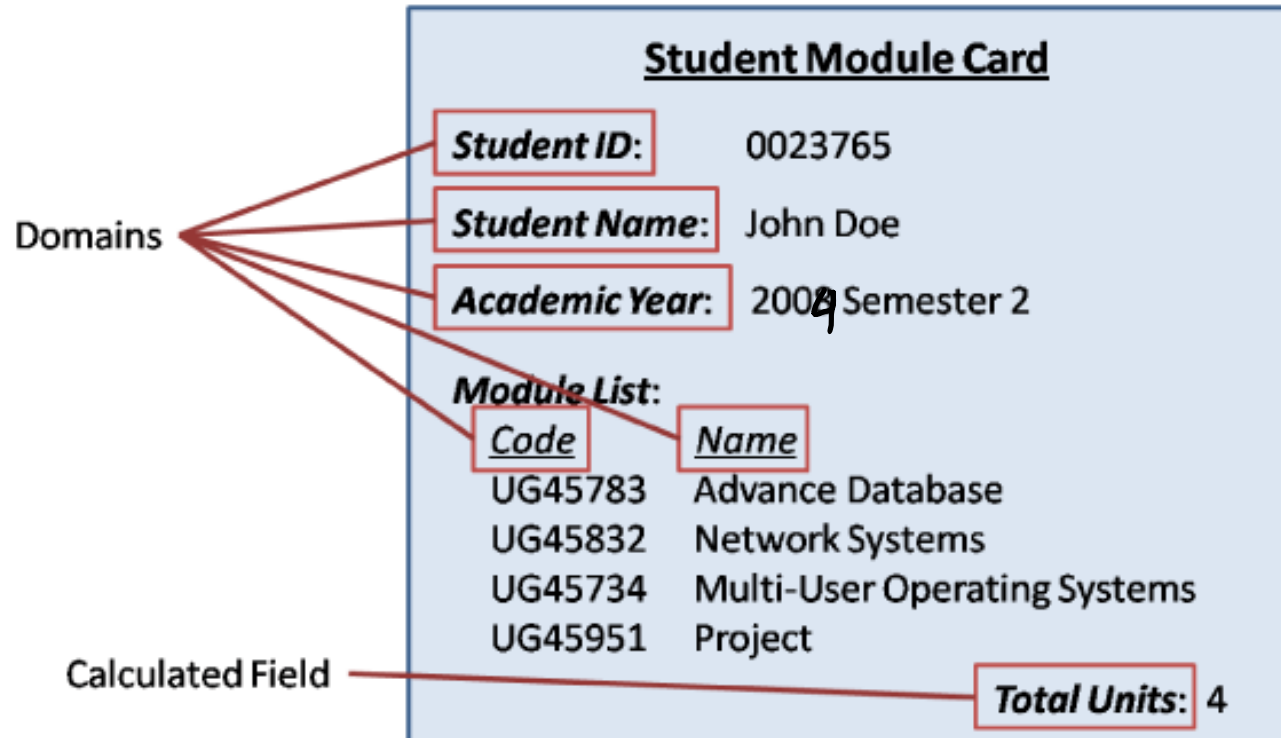
2.First Normal Form (1NF) – Repeating Groups

3.Second Normal Form (2NF) – Partial Dependencies

4.Third Normal Form (3NF) – Transitive Dependencies

Create Model Data

Un-Normalised Form (UNF)



Primary key

Sample Data

<u>StudentId</u>	StudentName	Year	Semester	UnitCode	UnitName
0023765	John Doe	2009	2	UG45783	Advance Database
				UG45832	Network Systems
				UG45734	Multi-User Operating Systems
0035643	Ann Smith	2009	2	UG45832	Network Systems
				UG45951	Project
0061234	Peter Wolfe	2009	2	UG45783	Advance Database

Model Data

First-Normal Form (1NF)

With First Normal Form we are looking to **remove repeating groups**. A repeating group is a domain or set of domains, directly relating to the key, that repeat data across tuples in order to cater for other domains where the data is different for each tuple.

Repeating Groups of Data

<u>StudentId</u>	StudentName	Year	Semester	UnitCode	UnitName
0023765	John Doe	2009	2	UG45783	Advance Database
0023765	John Doe	2009	2	UG45832	Network Systems
0023765	John Doe	2009	2	UG45734	Multi-User Operating Systems
0035643	Ann Smith	2009	2	UG45832	Network Systems
0035643	Ann Smith	2009	2	UG45951	Project
0061234	Peter Wolfe	2009	2	UG45783	Advance Database

The steps from UNF to 1NF are:

1. **Identify repeating groups of data**. Make sure your model data is of good quality to help identify the repeating groups and don't be afraid to move the domains around to help with the process.
2. **Remove the domains of the repeating groups** to a new relation leaving a copy of the primary key with the relation that is left.
3. **The original primary key will not now be unique so assign a new primary key** to the relation using the original primary key as part of a compound or **composite key**.
4. **Underline the domains that make up the key to distinguish** them from the other domains.

Composite Key

<u>StudentId</u>	StudentName	Year	Semester
0023765	John Doe	2009	2
0035643	Ann Smith	2009	2
0061234	Peter	2009	2

<u>StudentId</u>	<u>UnitCode</u>	UnitName
0023765	UG45783	Advance Database
0023765	UG45832	Network Systems
0023765	UG45734	Multi-User Operating Systems
0035643	UG45832	Network Systems
0035643	UG45951	Project
0061234	UG45783	Advance Database

Tables in
First Normal Form

Now our data is grouped into sets of related data we still need to check we are not keeping more data than we need to in our relation. We know we don't have any repeating groups as we removed these with First Normal Form. But if we look at our example we can see for every UnitCode we are also storing the UnitName.

<u>StudentId</u>	<u>UnitCode</u>	UnitName
0023765	UG45783	Advance Database
0023765	UG45832	Network Systems
0023765	UG45734	Multi-User Operating Systems
0035643	UG45832	Network Systems
0035643	UG45951	Project
0061234	UG45783	Advance Database

<u>StudentId</u>	<u>UnitCode</u>	UnitName
0023765	UG45783	Advance Database
0023765	UG45832	Network Systems
0023765	UG45734	Multi-User Operating Systems
0035643	UG45832	Network Systems
0035643	UG45951	Project
0061234	UG45783	Advance Database

Would it not seem more sensible to have a different relation we could use to look up UG45783 and find the unit name 'Advanced Database'? This way we wouldn't have to store lots of additional duplicate information in our Student/Unit relation.

This is exactly what we aim to achieve with Second Normal Form and its purpose is to remove partial dependancies.

We can consider a relation to be in Second Normal Form when: The relation is in First Normal Form and all partial key dependencies are removed so that all non key domains are functionally dependant on all of the domains that make up the primary key.

Before we start with the steps, if we have a table with only a single simple key this can't have any partial dependencies as there is only one domain that is a key therfroe these relations can be moved directly to 2nd normal form.

from 1NF to 2NF are:

1. Take each non-key domain in turn and check if it is only dependant on part of the key?
2. If yes
 1. Remove the non-key domain along with a copy of the part of the key it is dependent upon to a new relation.
 2. Underline the copied key as the primary key of the new relation.
3. Move down the relation to each of the domains repeating steps 1 and 2 till you have covered the whole relation.
4. Once completed with all partial dependencies removed, the table is in 2nd normal form.

In our example above, UnitName is only dependant on unitCode and has no dependency on studentId. Applying the steps above we move the unitName to a new relation with a copy of the part of the key it is dependent upon. Our table in second normal form would subsequently look like this:

Tables in Second
Normal Form

<u>StudentId</u>	<u>UnitCode</u>
0023765	UG45783
0023765	UG45832
0023765	UG45734
0035643	UG45832
0035643	UG45951
0061234	UG45783

<u>UnitCode</u>	UnitName
UG45783	Advance Database
UG45832	Network Systems
UG45734	Multi-User Operating Systems
UG45951	Project

Third Normal Form deals with something called **'transitive' dependencies**. This means if we have a primary key A and a non-key domain B and C where C is more dependent on B than A and B is directly dependent on A, then C can be considered transitively dependant on A.

<u>UnitCode</u>	UnitName	CourseCode	CourseName
UG45783	Advance Database	COMP2009	Computing
UG45832	Network Systems	COMP2009	Computing
UG45734	Multi-User Operating Systems	COMP2009	Computing
UG45951	Project	BUS22009	Business & Computing

from 2NF to 3NF are:

1. Take each non-key domain in turn and check it is more dependent on another non-key domain than the primary key.
2. If yes
 - a. Move the dependent domain, together with a copy of the non-key attribute upon which it is dependent, to a new relation.
 - b. Make the non-key domain, upon which it is dependent, the key in the new relation.
 - c. Underline the key in this new relation as the primary key.
 - d. Leave the non-key domain, upon which it was dependent, in the original relation and mark it a foreign key (*).
3. Move down the relation to each of the domains repeating steps 1 and 2 till you have covered the whole relation.
4. Once completed with all transitive dependencies removed, the table is in 3rd normal form.

In our example above, we have unitCode as our primary key, we also have a courseName that is dependent on courseCode and courseCode, dependent on unitCode. Though courseName could be dependent on unitCode it more dependent on courseCode, therefore it is transitively dependent on unitCode.

So following the steps, remove courseName with a copy of course code to another relation and make courseCode the primary key of the new relation. In the original table mark courseCode as our foreign key.

Foreign Key

<u>UnitCode</u>	UnitName	CourseCode*
UG45783	Advance Database	COMP2009
UG45832	Network Systems	COMP2009
UG45734	Multi-User Operating Systems	COMP2009
UG45951	Project	BUS22009

Tables in Third Normal Form

<u>CourseCode</u>	CourseName
COMP2009	Computing
BUS22009	Business & Computing

Primary Key

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++

1. First Normal Form (1NF)

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.

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,

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

And the new **Employee_Skill** table:

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++

2. Second Normal Form (2NF)

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**.

Let's take an example to understand Partial dependency and the Second Normal Form.

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.

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,

<u>student_id</u>	<u>subject_id</u>	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

3. Third Normal Form (3NF)

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**.

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	Theory	100
1	2	82	Theory	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.

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

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

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

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.



Normalization process using MySQL, focusing on transforming an unnormalized table through the first, second, and third normal forms using a practical example involving employees and their skills.

Original Table - Unnormalized Data

we start with a table called EmployeeSkills that combines employee information and their multiple skills into one table, which violates the First Normal Form (1NF) due to multivalued attributes.

```
CREATE TABLE EmployeeSkills (  
    emp_id INT,  
    emp_name VARCHAR(255),  
    emp_mobile VARCHAR(20),  
    emp_skills VARCHAR(255) -- Multivalued attribute  
);
```

Sample Data:

```
INSERT INTO EmployeeSkills (emp_id, emp_name, emp_mobile, emp_skills) VALUES  
(1, 'John Doe', '9999999999', 'Python,JavaScript'),  
(2, 'Jane Doe', '8888888888', 'HTML,CSS');
```

To satisfy 1NF, we need to eliminate multivalued attributes by ensuring that each column contains atomic values only.

Normalization:

- Split **EmployeeSkills** into two tables: **Employees** and **Skills**.

Employees Table:

```
CREATE TABLE Employees (  
  emp_id INT PRIMARY KEY,  
  emp_name VARCHAR(255),  
  emp_mobile VARCHAR(20)  
);
```

Skills Table:

```
CREATE TABLE Skills (  
  skill_id INT PRIMARY KEY AUTO_INCREMENT,  
  emp_id INT,  
  skill VARCHAR(255),  
  FOREIGN KEY (emp_id) REFERENCES Employees(emp_id)  
);
```

```
INSERT INTO Employees (emp_id, emp_name, emp_mobile) VALUES  
(1, 'John Doe', '9999999999'),  
(2, 'Jane Doe', '8888888888');
```

```
INSERT INTO Skills (emp_id, skill) VALUES  
(1, 'Python'), (1, 'JavaScript'),  
(2, 'HTML'), (2, 'CSS');
```

Now, each table conforms to 1NF, as every column holds atomic values, and there are no repeating groups.

For a table to be in 2NF, it must first satisfy 1NF, and all non-key attributes must be fully functionally dependent on the primary key.

Given our Employees and Skills tables are already in 1NF and do not have composite primary keys, they inherently satisfy 2NF since there are no **partial dependencies** (each non-key attribute in these tables is dependent on the primary key).

A table is in 3NF if it is in 2NF and all its columns are only directly dependent on the primary key.

Let's introduce a new scenario that violates 3NF to further normalize:

Scenario: Adding a new table to handle transitive dependency:

Assume there's a **Departments table**, which introduces **transitive dependency** in Employees table:

```
CREATE TABLE Departments (  
    dept_id INT PRIMARY KEY,  
    dept_name VARCHAR(255),  
    manager_id INT  
);
```

If manager_id in Departments table refers to emp_id in Employees, creating a transitive dependency (Employee -> Department -> Manager), we must refactor to maintain 3NF.

To adhere to 3NF, ensure that non-key attributes (dept_name) do not depend on other non-key attributes (manager_id).

Refactored Structure:

Keep Departments focused on department information:

```
CREATE TABLE Departments (  
    dept_id INT PRIMARY KEY,  
    dept_name VARCHAR(255)  
);
```

And possibly create another table to associate departments with their managers if needed, ensuring all attributes are directly dependent on the primary key, avoiding transitive dependencies.



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

