Database Normalization Code

Data normalization is a process used to organize a database efficiently, reduce data redundancy and improve Data Integrity. It involves structuring a relational database in accordance with a series of normal forms to improve data integrity and minimize anomalies. The normalization process typically results in a set of tables with clear relationships and a reduced likelihood of data inconsistencies.

```
Customer_ID | Customer_DOB
1001        | 17-Feb-2000
1002        | 19-Oct-1998
1001        | 06-Sep-2001


Data-Integrity Fail
    - Data Can't be trusted
    - Data disagrees with itself
```

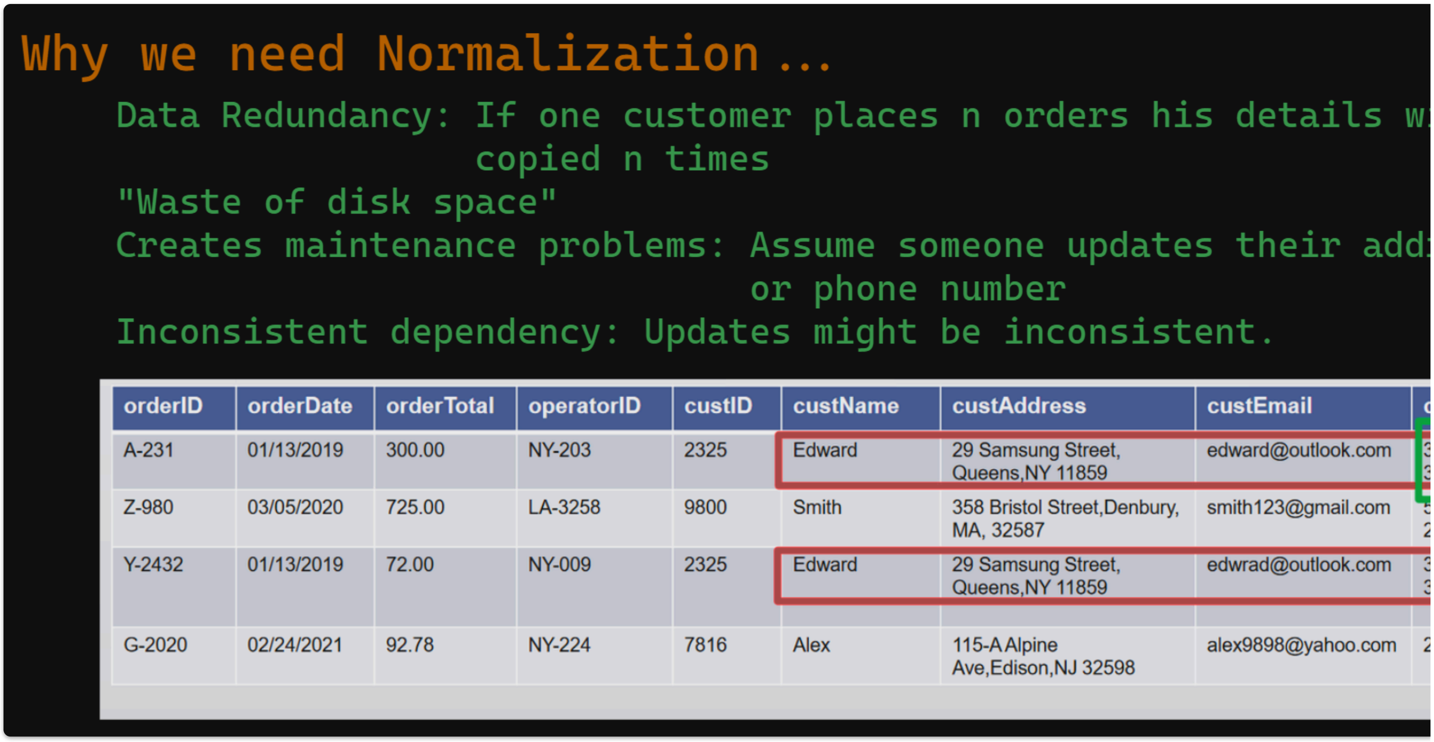
- This should never happen
- *Not Normalized*
- *Bad Database Design*
- There is no key!

Normalized Tables are:

1. Easier to understand.
2. Easier to enhance and extend
3. Protected from:
    1. Insertion Anomalies
    2. Update Anomalies
    3. Deletion Anomalies

First Normal Form (1NF):

1. Using row order to convey information is not permitted
2. Mixing data types within the same column is not permitted
3. Having a table without a primary key is not permitted
4. Repeating groups are not permitted
    - Consider a table for storing customer orders. Instead of having columns for "order_item1," "order_item2," etc., each item is placed in a separate row linked to the customer's order.



orderID	orderDate	orderTotal	operatorID	custID	custName	custAddress	custEmail
A-231	01/13/2019	300.00	NY-203	2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com
Z-980	03/05/2020	725.00	LA-3258	9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com
Y-2432	01/13/2019	72.00	NY-009	2325	Edward	29 Samsung Street, Queens,NY 11859	edwrad@outlook.com
G-2020	02/24/2021	92.78	NY-224	7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com

Second Normal Form (2NF):

- There should not be any partial dependency
- In 2NF, the table is in 1NF, and all the non-key attributes are fully functional dependent on the primary key.
- Each non-key attribute must depend on the entire primary key.
- Example: A table with customer sales, where the primary key consists of the customer ID and product ID, and non-key attributes like product name, price, and quantity are functionally dependent on the entire primary key (customer ID and product ID).

Example:

player_ID	Item_Type	Item_Quantity	Player_Rating
P001	Gem	500	2000
P001	Gold	100000	2000
P002	Elixir	50000	1500
P002	Gem	100	1500

In this table, we have a composite primary key consisting of player_ID and Item_Type. However, this violates 2NF because Player_Rating is only dependent on player_ID, not on the entire primary key. This is a partial dependency, which 2NF aims to eliminate.

To correct this and achieve 2NF, we need to split this table into two separate tables:

Table 1: Player_Items

player_ID	Item_Type	Item_Quantity
P001	Gem	500
P001	Gold	100000
P002	Elixir	50000
P002	Gem	100

$$\text{PlayerID}, \text{ItemType} \rightarrow \text{ItemQuantity}$$

Table 2: Player_Ratings

player_ID	Player_Rating
P001	2000
P002	1500

$$\text{PlayerID} \rightarrow \text{PlayerRating}$$

Now, in the Player_Items table, all non-key attributes (Item_Quantity) depend on the entire primary key (player_ID and Item_Type). In the Player_Ratings table, Player_Rating depends on the entire primary key (player_ID).

Third Normal Form (3NF):

- In 3NF, the table is in 2NF, and no transitive dependencies exist.
- *Every non-key attribute in a table should depend on the key, the whole key and nothing but the key.*

PlayerID	ClanID	ClanName	TownHallLevel	TroopCapacity
P001	C001	Warriors	10	220
P002	C001	Warriors	8	200
P003	C002	Wizards	11	240
P004	C002	Wizards	9	220

In this table:

- PlayerID is the primary key
- ClanID and ClanName have a dependency: ClanID → ClanName
- TownHallLevel determines TroopCapacity: TownHallLevel → TroopCapacity

The problem here is that ClanName depends on ClanID (not on PlayerID), and TroopCapacity depends on TownHallLevel (not directly on PlayerID). These are transitive dependencies that violate 3NF.

To achieve 3NF, we need to separate these into different tables:

1. Player Table:

PlayerID	ClanID	TownHallLevel
P001	C001	10
P002	C001	8
P003	C002	11
P004	C002	9

2. Clan Table:

ClanID	ClanName
C001	Warriors
C002	Wizards

3. TownHall_Capacity Table:

TownHallLevel	TroopCapacity
8	200
9	220
10	220
11	240

Now, let's express the functional dependencies in LaTeX notation:

1. For the Player Table:

$$\text{PlayerID} \rightarrow \{\text{ClanID}, \text{TownHallLevel}\}$$

2. For the Clan Table:

$$\text{ClanID} \rightarrow \text{ClanName}$$

3. For the TownHall_Capacity Table:

$$\text{TownHallLevel} \rightarrow \text{TroopCapacity}$$

In these new tables:

1. Each table is in 2NF (as required for 3NF).
2. All attributes in each table are fully functionally dependent on the primary key of that table.
3. There are no transitive dependencies within any table.

This 3NF structure offers several benefits:

- Eliminates data redundancy (e.g., ClanName is stored only once per clan).
- Improves data integrity (e.g., updating a clan name only requires changing one record).
- Allows for more flexible data management (e.g., TroopCapacity can be updated for a TownHallLevel without affecting player records).

By organizing the data this way, we've achieved 3NF, ensuring that our database structure is free from insert, update, and delete anomalies that can occur in less normalized forms.

- Example: Consider an employee table where "employee_id," "department_id," and "department_name" are columns. The "department_name" is transitively dependent on the "employee_id" through the "department_id" column. The solution is to create a separate "departments" table with "department_id" and "department_name" columns.

  Certainly. Let's illustrate this scenario using Markdown tables to show how we can transform a table that violates 3NF into one that satisfies it.

Original Table (Violating 3NF):

employee_id	department_id	department_name	employee_name
E001	D01	Sales	John Doe
E002	D01	Sales	Jane Smith
E003	D02	Marketing	Bob Johnson
E004	D02	Marketing	Alice Brown
E005	D03	IT	Charlie Davis

In this table, we have a transitive dependency:
employee_id → department_id → department_name

To resolve this and achieve 3NF, we split this into two tables:

1. Employees Table:

employee_id	department_id	employee_name
E001	D01	John Doe
E002	D01	Jane Smith
E003	D02	Bob Johnson
E004	D02	Alice Brown
E005	D03	Charlie Davis

2. Departments Table:

department_id	department_name
D01	Sales
D02	Marketing
D03	IT

Now, let's express the functional dependencies:

For the Employees Table:

$$\text{employee\_id} \rightarrow \{\text{department\_id}, \text{employee\_name}\}$$

For the Departments Table:

$$\text{department\_id} \rightarrow \text{department\_name}$$

This structure satisfies 3NF because:

1. It's in 2NF (each table has a clear primary key with no partial dependencies).

2. There are no transitive dependencies within each table.
3. All non-key attributes in each table depend directly on the primary key.

Benefits of this 3NF structure:

1. Eliminates redundancy: department names are stored only once per department.
2. Improves data integrity: updating a department name requires changing only one record.
3. Prevents anomalies: adding a new department doesn't require an employee, and removing an employee doesn't lose department information.

Boyce-Codd Normal Form (BCNF):

- BCNF is a stricter form of 3NF, where every determinant is a candidate key.
- Example: In a table containing details of a university course, if the combination of "course_code" and "semester" uniquely determines "instructor," "room_number," and "meeting_time," it is not in BCNF. To achieve BCNF, the table must be split into multiple tables.

Fourth Normal Form (4NF) and Fifth Normal Form (5NF) focus on multi-valued dependencies and join dependencies, which are more advanced and typically not as commonly encountered in standard database design.

Normalization is an iterative process that aims to reduce data redundancy and improve data integrity within a database. By applying normalization forms, we can create well-structured, efficient databases that minimize the chances of data inconsistencies and anomalies. It's important to note that achieving higher normal forms might not always be necessary or practical for all databases, and the level of normalization depends on the specific needs of the application and the trade-offs involved in database design.

Database Normalization Code

First Normal Form (1NF)

First normal form (1NF) is the simplest level of normalization. It involves ensuring that each table in the database has a primary key and that each column in the table contains atomic values. In other words, each row in the table should have a unique identifier, and each value in the table should be indivisible.

> *Let's take an example to understand this better. Consider a table that stores information about employees. The table might have columns like* `employee_id`, `name`,

> *`address`, and `phone_number`. However, the `address` column could contain multiple values, like street name, city, state, and zip code.*

EMPLOYEE_ID	NAME	ADDRESS	PHONE_NUMBER
1	John Smith	Main St., Anytown, NY 12345	555-555-5555
2	Jane Doe	High St., Anytown, NY 12345	555-555-5555

Example Table

> *To bring this table to 1NF, we need to split the `address` column into separate columns, each containing a single value.*

EMPLOYEE_ID	NAME	STREET_NAME	TOWN	CITY	ZIP_CODE	PHONE_NUMBER
1	John Smith	Main St.	Anytown	NY	12345	555-555-5555
2	Jane Doe	High St.	Anytown	NY	12345	555-555-5555

1NF Output

Second Normal Form (2NF)

Second normal form (2NF) builds on the foundation of 1NF and involves ensuring that each non-key column in a table is dependent on the primary key. In other words, there should be no partial dependencies in the table.

> *Let's continue with our employee table example. Suppose we add a column for `department` to the table. If we find that the value in the `department` column is dependent on the `employee_id` and `name` columns, but not on the `phone_number` column, we need to split the table into two tables, one for employee information and one for department information.*

EMPLOYEE

EMPLOYEE_ID	NAME	PHONE_NUMBER
1	John Smith	555-555-5555
2	Jane Doe	555-555-5555

DEPARTMENT

EMPLOYEE_ID	DEPARTMENT
1	Sales
2	Marketings

2NF Output

Third Normal Form (3NF)

Third normal form (3NF) builds on the foundation of 2NF and involves ensuring that each non-key column in a table is not transitively dependent on the primary key. In other words, there should be no transitive dependencies in the table.

> *Let's take another example. Consider a table that stores information about books. The table might have columns like `book_id`, `title`, `author`, and `publisher`.*
>
> *However, the `publisher` column could be dependent on the `author` column, rather than on the `book_id` column. To bring this table to 3NF, we need to split it into two tables, one for book information and one for author information.*

BOOK

BOOK_ID	TITLE	AUTHOR_ID
1	The Great Gatsby	1
2	To Kill a Mockingbird	2
3	1984	3

AUTHOR

AUTHOR_ID	AUTHOR_NAME	PUBLISHER_ID
1	F. Scott Fitzgerald	1
2	Harper Lee	2
3	George Orwell	3

PUBLISHER

PUBLISHER_ID	PUBLISHER_NAME
1	Scribner
2	J. B. Lippincott & Co.
3	Secker & Warburg

3NF Output

BCNF — Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is a higher level of normalization than 3NF. It is used to eliminate the possibility of functional dependencies between non-key attributes. A table is in BCNF if and only if every determinant in the table is a candidate key.

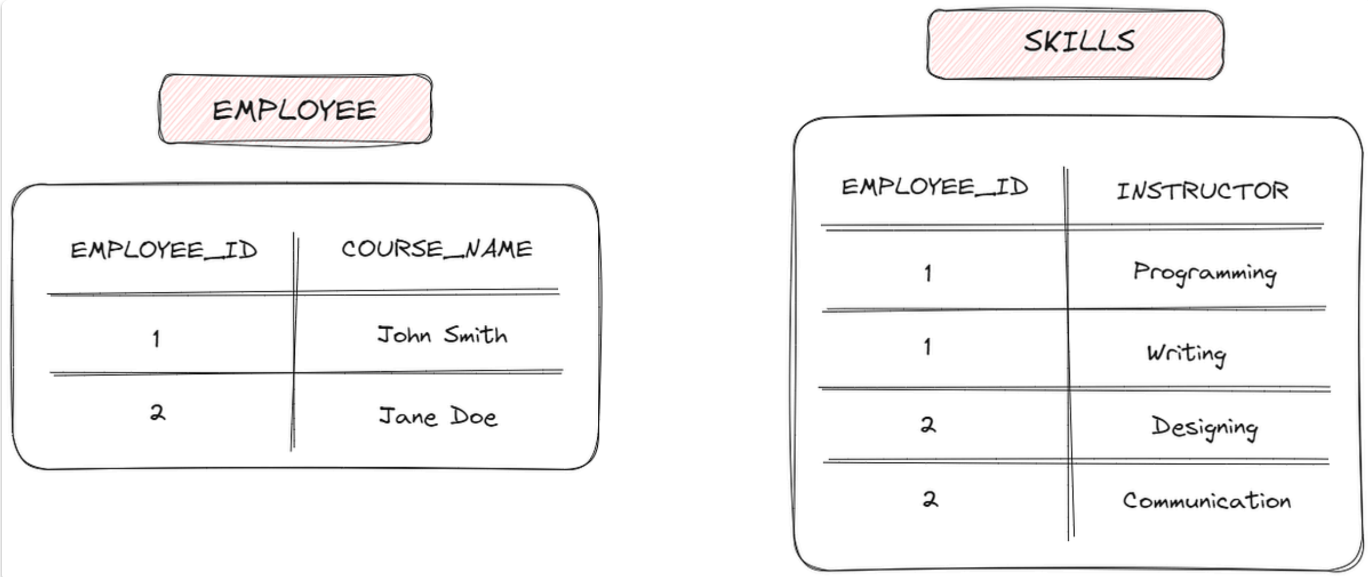
> *To understand BCNF better, consider a table that stores information about students and their courses. The table might have columns like `student_id`, `course_id`, `instructor`, and `instructor_office`. In this table, the determinant is `course_id`, and the non-key attribute is `instructor`. However, a course can have multiple instructors, so there is a possibility of functional dependencies between non-key attributes. To bring this table to BCNF, we need to split it into two tables, one for course information and one for instructor information.*

BCNF Output

Fourth Normal Form (4NF)

Fourth Normal Form (4NF) is the highest level of normalization and is used to eliminate the possibility of multi-valued dependencies in a table. A multi-valued dependency occurs when one or more attributes are dependent on a part of the primary key, but not on the entire primary key.

> *To understand 4NF better, consider a table that stores information about employees and their skills. The table might have columns like `employee_id`, `skill`, and `proficiency_level`. In this table, the primary key is a combination of `employee_id` and `skill`. However, the proficiency level is dependent on the skill, but not on the entire primary key. To bring this table to 4NF, we need to split it into two tables, one for employee information and one for skill information.*



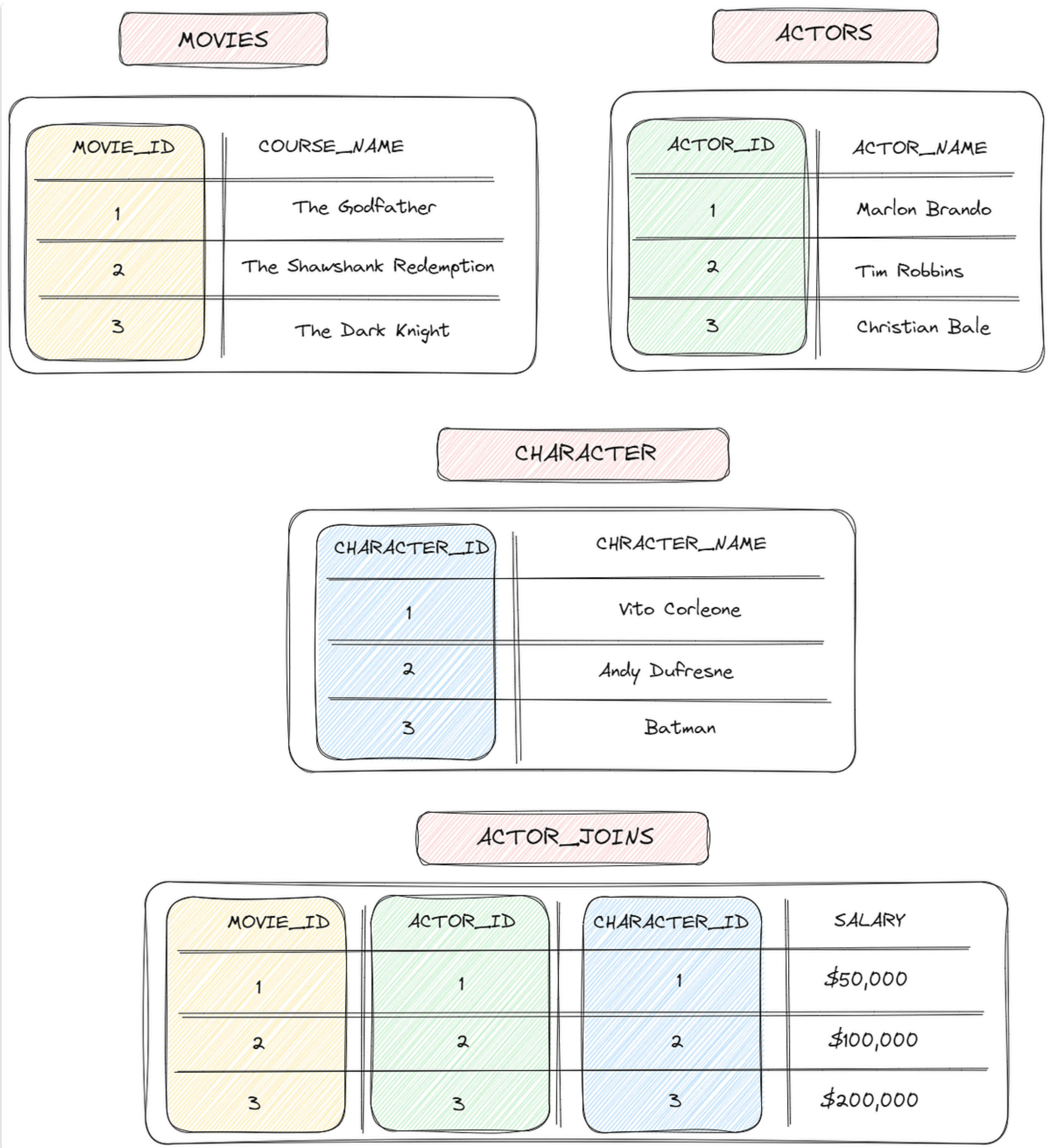
4NF Output

Fifth Normal Form (5NF)

Fifth normal form (5NF) is the highest level of normalization and is also known as Project-Join Normal Form (PJNF). It is used to handle complex many-to-many relationships in a database.

In a many-to-many relationship, where each table has a composite primary key, it is possible for a non-trivial functional dependency to exist between the primary key and a non-key attribute. 5NF deals with these situations by decomposing the tables into smaller tables that preserve the relationships between the attributes.

> *To understand this better, consider a database that stores information about movies and their actors. The tables might have columns like `movie_id`, `actor_id`, `character_name`, and `salary`. In this database, it is possible for a non-trivial functional dependency to exist between the primary key (`movie_id`, `actor_id`) and the `salary` attribute.*
>
> *To bring this database to 5NF, we need to decompose the tables into smaller tables. For example, we might create tables for movies, actors, and characters, and then use a join table to connect them. Each table would have a single primary key, and the join table would include foreign keys to the other tables.*



5NF Output

Reflection

Today, many organizations rely on databases to store, manage, and retrieve their data. In order to ensure that the data is organized in a way that is both efficient and consistent, normalization is often used. There are several levels of normalization that can be applied, with 1NF, 2NF, and 3NF being the most commonly used.

In addition to 1NF, 2NF, and 3NF, there are also advanced normalization techniques such as Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF). BCNF is used to eliminate the possibility of functional dependencies between non-key attributes. 4NF is used to eliminate the possibility of multi-valued dependencies in a table. 5NF, also known as Project-Join Normal Form (PJNF), is used to handle complex many-to-many relationships in a database.

While these levels of normalization can provide further data consistency and management benefits, they can also result in more complex table relationships, slower queries, and larger numbers of tables. Therefore, it's important to carefully consider the use cases and benefits of each technique before applying them in database design.

Normal Form	Description	Benefits	Drawbacks
1NF	Each table has a primary key and each column contains atomic values.	Eliminates duplicate data and ensures data consistency.	Can result in large tables with many columns.
2NF	Each non-key column is dependent on the primary key.	Reduces data redundancy and simplifies data management.	Can result in complex table relationships and slower queries.
3NF	Each non-key column is not transitively dependent on the primary key.	Further reduces data redundancy and improves data consistency.	Can result in even more complex table relationships and slower queries.
BCNF	Every determinant in the table is a candidate key.	Eliminates the possibility of functional dependencies between non-key attributes.	Can be difficult to achieve and may result in large numbers of tables.
4NF	Eliminates the possibility of multi-valued dependencies in a table.	Reduces data redundancy and simplifies data management.	Can result in large numbers of tables and slower queries.
5NF	Handles complex many-to-many relationships in a database.	Improves data consistency and simplifies data management.	Can result in a large number of tables and slower queries.

Normal Forms Comparison Table

1NF

orderID	orderDate	orderTotal	operatorID	custID	custName	custAddress	custEmail	custPhone
A-231	01/13/2019	300.00	NY-203	2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com	324-987-3587, 324-897-3582
Z-980	03/05/2020	725.00	LA-3258	9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com	589-698-8547, 212-396-1380
Y-2432	01/13/2019	72.00	NY-009	2325	Edward	29 Samsung Street, Queens,NY 11859	edwrad@outlook.com	324-987-3587, 324-897-3582
G-2020	02/24/2021	92.78	NY-224	7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com	214-859-9852

Tables in 1NF

orderID	orderDate	orderTotal	operatorID	custID
A-231	01/13/2019	300.00	NY-203	2325
Z-980	03/05/2020	725.00	LA-3258	9800
Y-2432	01/13/2019	72.00	NY-009	2325
G-2020	02/24/2021	92.78	NY-224	7816

custID	custName	custAddress	custEmail	custPriPhone	custAltPhone
2325	Edward	29 Samsung Street, Queens,NY 11859	edward@outlook.com	324-987-3587	324-897-3582
9800	Smith	358 Bristol Street,Denbury, MA, 32587	smith123@gmail.com	589-698-8547	212-396-1380
7816	Alex	115-A Alpine Ave,Edison,NJ 32598	alex9898@yahoo.com	214-859-9852	NULL

2NF

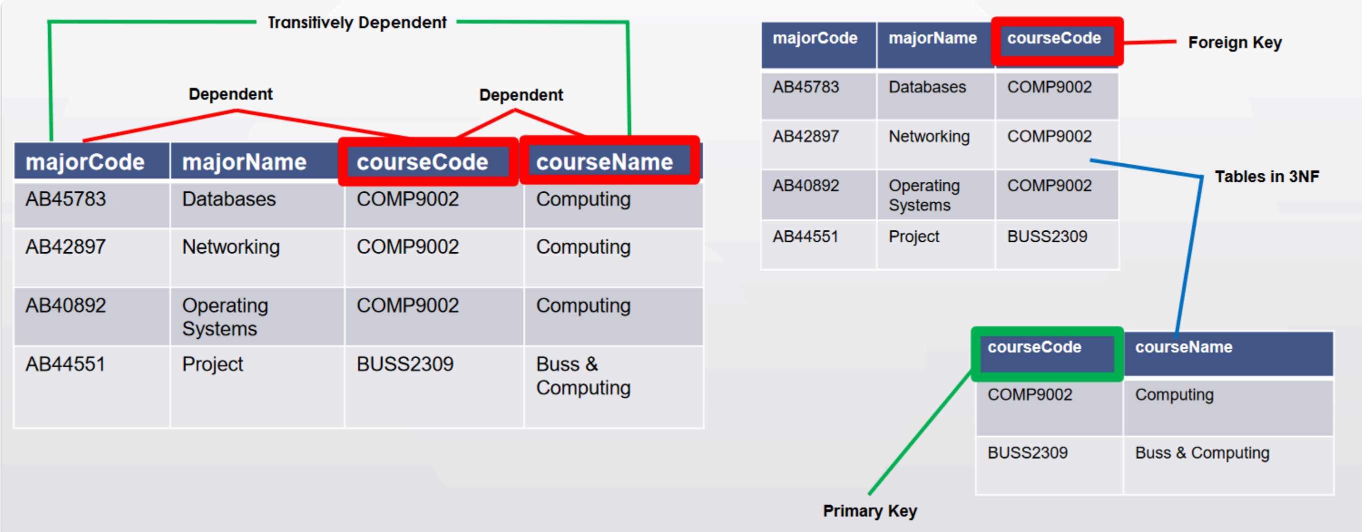
orderID	productID	productName
A-231	PR1258	Desk
Z-980	PR2298	Chair
Y-2432	PR7780	Lamp
G-2020	PR1258	Desk

Tables in 2NF

orderID	productID
A-231	PR1258
Z-980	PR2298
Y-2432	PR7780
G-2020	PR1258

productID	productName
PR1258	Desk
PR2298	Chair
PR7780	Lamp

3NF



majorCode	majorName	courseCode	courseName
AB45783	Databases	COMP9002	Computing
AB42897	Networking	COMP9002	Computing
AB40892	Operating Systems	COMP9002	Computing
AB44551	Project	BUSS2309	Buss & Computing

majorCode	majorName	courseCode
AB45783	Databases	COMP9002
AB42897	Networking	COMP9002
AB40892	Operating Systems	COMP9002
AB44551	Project	BUSS2309

courseCode	courseName
COMP9002	Computing
BUSS2309	Buss & Computing