# 📘 Documentation on Normalization

**Database normalization** is the process of structuring a relational database to reduce data redundancy and improve data integrity. It involves dividing larger tables into smaller, well-structured tables and defining relationships between them.The goal is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

The process is formalized through a series of "normal forms." A table is said to be in a certain normal form if it satisfies a specific set of constraints.

## First Normal Form (1NF)

The first and most basic rule of normalization.

* **Requirement:**
  1. The table must have a **primary key**.
  2. Each column must contain **atomic** (indivisible) values. This means no repeating groups or multivalued columns (like storing "SQL, Python, Java" in a single Skills column).
  3. All entries in a column must be of the same data type.
* **Problem Solved:** Eliminates repeating groups and multivalued attributes, making the data easier to query and manage.
* Example:  
  Consider a table that is not in 1NF:

|  |  |  |
| --- | --- | --- |
| StudentID | Name | PhoneNumbers |
| 101 | Alex | 555-1234, 555-5678 |
| 102 | Brenda | 555-8765 |

The PhoneNumbers column contains multiple values, violating the atomicity rule.  
**To achieve 1NF**, we create a separate table for the repeating attribute:

* Students Table (1NF)

|  |  |
| --- | --- |
| StudentID (PK) | Name |
| 101 | Alex |
| 102 | Brenda |

* Student\_PhoneNumbers Table (1NF)

|  |  |
| --- | --- |
| StudentID (FK) | PhoneNumber (PK) |
| 101 | 555-1234 |
| 101 | 555-5678 |
| 102 | 555-8765 |

## Second Normal Form (2NF)

Builds upon 1NF and deals with tables that have composite primary keys.

* **Requirement:**
  1. The table must be in **1NF**.
  2. All non-key attributes must be **fully functionally dependent** on the *entire* primary key. This means no partial dependencies.
* **Problem Solved:** Removes data redundancy that occurs when a non-key attribute depends on only a part of a composite primary key.
* Example:  
  Consider a table with a composite primary key (StudentID, CourseID) that is in 1NF but not in 2NF:

|  |  |  |  |
| --- | --- | --- | --- |
| StudentID (PK) | CourseID (PK) | CourseName | Grade |
| 101 | CS101 | Intro to CS | A |
| 101 | MA203 | Calculus II | B |
| 102 | CS101 | Intro to CS | B |

Here, the Grade depends on the full key (StudentID, CourseID). However, CourseName depends *only* on CourseID. This is a **partial dependency**.  
**To achieve 2NF**, we decompose the table:

Enrollments Table

|  |  |  |
| --- | --- | --- |
| StudentID (PK, FK) | CourseID (PK, FK) | Grade |
| 101 | CS101 | A |
| 101 | MA203 | B |
| 102 | CS101 | B |

Courses Table

|  |  |
| --- | --- |
| CourseID (PK) | CourseName |
| CS101 | Intro to CS |
| MA203 | Calculus II |

## Third Normal Form (3NF)

Builds upon 2NF and removes another type of dependency.

* **Requirement:**
  1. The table must be in **2NF**.
  2. There must be no **transitive dependencies**. A transitive dependency exists when a non-key attribute depends on another non-key attribute, rather than directly on the primary key. If A→B and B→C, then A→C is a transitive dependency.
* **Problem Solved:** Reduces redundancy by ensuring that attributes depend only on the primary key and nothing but the primary key.
* Example:  
  Consider a table that is in 2NF but not in 3NF:

|  |  |  |  |
| --- | --- | --- | --- |
| EmployeeID (PK) | Name | DepartmentID | DepartmentName |
| E101 | Alice | D01 | Marketing |
| E102 | Bob | D02 | Engineering |
| E103 | Charlie | D01 | Marketing |

Here, DepartmentName depends on DepartmentID, which in turn depends on the primary key EmployeeID. This is a **transitive dependency**.  
**To achieve 3NF**, we move the transitively dependent attribute to its own table:

* Employees Table (3NF)

|  |  |  |
| --- | --- | --- |
| **EmployeeID (PK)** | **Name** | **DepartmentID (FK)** |
| E101 | Alice | D01 |
| E102 | Bob | D02 |
| E103 | Charlie | D01 |

* Departments Table (3NF)

|  |  |
| --- | --- |
| **DepartmentID (PK)** | **DepartmentName** |
| D01 | Marketing |
| D02 | Engineering |

## Boyce-Codd Normal Form (BCNF)

A stricter version of 3NF, sometimes called 3.5NF.

* **Requirement:**
  1. The table must be in **3NF**.
  2. For any non-trivial functional dependency X→Y, X must be a **superkey**. A superkey is a set of one or more attributes that can uniquely identify a record.
* **Problem Solved:** Addresses certain anomalies that can arise even in 3NF tables, typically those with multiple candidate keys that overlap.
* Example:  
  Consider a table where students are assigned an advisor for a specific major. Assume a student can have multiple majors, and an advisor is assigned to only one major.

|  |  |  |
| --- | --- | --- |
| StudentID (PK) | Major (PK) | AdvisorName |
| 101 | Physics | Dr. Smith |
| 101 | Music | Dr. Jones |
| 102 | Physics | Dr. Smith |

* + **Candidate Keys:** (StudentID, Major)
  + **Functional Dependencies:**
    - (StudentID, Major) \rightarrow AdvisorName (This is fine, the determinant is a superkey).
    - AdvisorName \rightarrow Major (An advisor handles only one major).  
      The second dependency violates BCNF because AdvisorName is not a superkey.

**To achieve BCNF**, we decompose the table:

Student\_Advisors Table (BCNF)

|  |  |
| --- | --- |
| **StudentID (PK, FK)** | **AdvisorName (PK, FK)** |
| 101 | Dr. Smith |
| 101 | Dr. Jones |
| 102 | Dr. Smith |

Advisor\_Majors Table (BCNF)

|  |  |
| --- | --- |
| **AdvisorName (PK)** | **Major** |
| Dr. Smith | Physics |
| Dr. Jones | Music |

## Fourth Normal Form (4NF)

Deals with a different kind of dependency.

* **Requirement:**
  1. The table must be in **BCNF**.
  2. There must be no **multivalued dependencies**. A multivalued dependency exists when the presence of one row implies the presence of other rows. Specifically, for a dependency A→→B, the values of B are determined by A, independent of any other attribute C.
* **Problem Solved:** Removes redundancy caused by storing two or more independent many-to-many relationships in a single table.
* Example:  
  Consider a table storing which courses a student is taking and which sports they play. Courses and sports are independent.

|  |  |  |
| --- | --- | --- |
| StudentID (PK) | Course (PK) | Sport (PK) |
| 101 | CS101 | Tennis |
| 101 | CS101 | Soccer |
| 101 | MA203 | Tennis |
| 101 | MA203 | Soccer |

This table implies that for student 101, every course is paired with every sport, creating redundant data. This is a multivalued dependency: StudentID \rightarrow\rightarrow Course and StudentID \rightarrow\rightarrow Sport.

**To achieve 4NF**, we separate the independent relationships:

### Student\_Courses Table (4NF)

|  |  |
| --- | --- |
| **StudentID (PK, FK)** | **Course (PK)** |
| 101 | CS101 |
| 101 | MA203 |

### 

### Student\_Sports Table (4NF)

|  |  |
| --- | --- |
| **StudentID (PK, FK)** | **Sport (PK)** |
| 101 | Tennis |
| 101 | Soccer |

## Fifth Normal Form (5NF) & Project-Join Normal Form (PJ/NF)

The requirements for 5NF and PJ/NF are identical. This is a very rare scenario.

* **Requirement:**
  1. The table must be in **4NF**.
  2. There must be no **join dependencies**. A join dependency exists if a table can be decomposed into smaller tables and then losslessly rejoined to recreate the original table. 5NF is violated when this decomposition is possible but not implied by the candidate keys.
* **Problem Solved:** Isolates semantically related multiple relationships and prevents data redundancy that is not covered by previous normal forms.
* Example:  
  Consider a table showing which sales agents work for which companies and sell which products.

|  |  |  |
| --- | --- | --- |
| Agent | Company | Product |
| Smith | Acme | Widgets |
| Jones | Globex | Gadgets |
| Smith | Globex | Gadgets |

* Now, add a business rule: **An agent must sell a product for a company only if the agent works for that company and that company manufactures the product.** If we store the relationships separately (Agent-Company, Company-Product, Agent-Product), we might create a "phantom" row upon joining. For instance, if Smith also sold Gadgets (but not for Globex), joining the tables could incorrectly imply that he sells Gadgets for Globex. 5NF requires that such interrelated constraints be handled by decomposing the table.  
  **To achieve 5NF**, you decompose the table into the smallest possible relations that can be joined without losing data or creating false data.  
  Agent\_Companies (Agent, Company)  
  Company\_Products (Company, Product)  
  Agent\_Products (Agent, Product)

## Higher Normal Forms

### Sixth Normal Form (6NF)

6NF is not standardized in the same way as the others and is primarily of academic or niche interest, especially in temporal (time-variant) databases.

* **Requirement:**
  1. The table must be in **5NF**.
  2. The relation is decomposed into its irreducible components. A table is in 6NF if it consists of a primary key and at most **one** non-key attribute.
* **Problem Solved:** Provides the ultimate level of decomposition, eliminating all non-trivial dependencies and maximizing flexibility, particularly for tracking changes to attributes over time.
* Example:

Original Table (Before 6NF)

|  |  |  |  |
| --- | --- | --- | --- |
| **EmployeeID (PK)** | **Name** | **Salary** | **HireDate** |
| E101 | Sarah | 80000 | 2022-01-15 |

(After 6NF)

Employee\_Names

|  |  |
| --- | --- |
| **EmployeeID (PK, FK)** | **Name** |
| E101 | Sarah |

#### 

#### Employee\_Salaries

|  |  |
| --- | --- |
| **EmployeeID (PK, FK)** | **Salary** |
| E101 | 80000 |

#### 

#### Employee\_HireDates

|  |  |
| --- | --- |
| **EmployeeID (PK, FK)** | **HireDate** |
| E101 | 2022-01-15 |

### Domain-Key Normal Form (DKNF)

This is considered the "ultimate" normal form but is largely theoretical.

* **Requirement:** Every constraint on the relation is a logical consequence of the definition of **domains** and **keys**.
  + **Domain Constraint:** Restricts a value in a column to a specific set (e.g., Gender must be 'Male', 'Female', or 'Other').
  + **Key Constraint:** Uniquely identifies a row.
* **Problem Solved:** Theoretically, a table in DKNF is free of all modification anomalies.
* **Practicality:** There is no simple methodology for converting a table to DKNF. It serves as a guiding ideal. In practice, achieving 3NF or BCNF is sufficient for the vast majority of database designs.