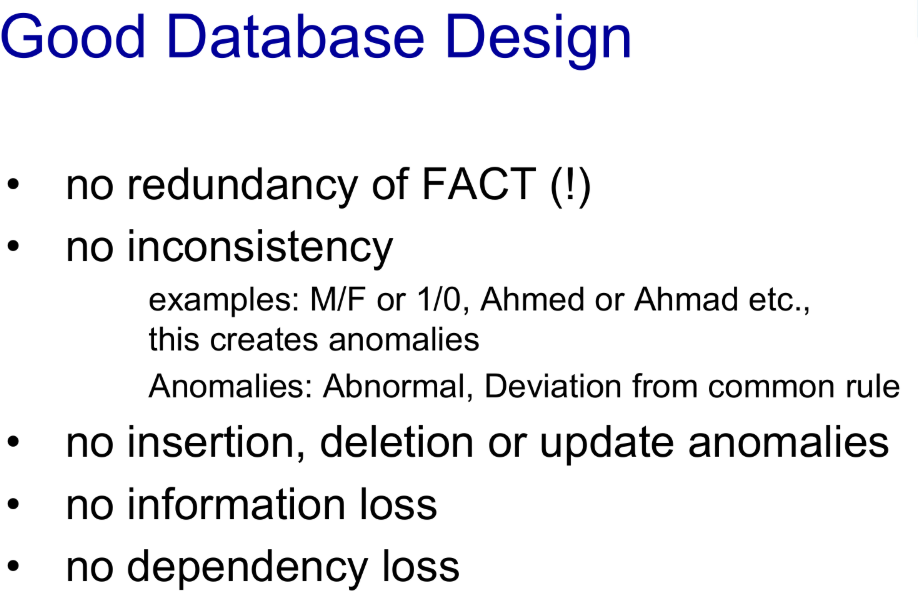
****

**Normalization**

Normalization in the context of relational databases is a process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing larger tables into smaller, more manageable tables and defining relationships between them. The goal is to eliminate or minimize data duplication and the possibility of data inconsistencies (anomalies) that can arise from updates, insertions, and deletions.

Normalization typically involves a series of "normal forms" (often referred to as 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, and 6NF), which represent increasingly stringent sets of rules for database design. Most transactional databases are normalized to the third normal form (3NF) as a good balance between minimizing redundancy and maintaining performance. Higher normal forms are often used for specialized applications like data warehousing.

**Importance**

* **Reduced Data Redundancy:** Minimizing duplicate data saves storage space and reduces the chances of inconsistencies. If the same information is stored in multiple places (cells), updating it in one place but not others can lead to inconsistencies.
* **Improved Data Integrity:** By eliminating redundancy, normalization helps ensure that data is consistent across the database. Changes to a piece of information only need to be made in one place.
* **Easier Data Maintenance:** Updating, inserting, and deleting data becomes simpler and less prone to errors when data is well-organized and not duplicated.
* **Better Database Design:** Normalization helps in creating a more logical and efficient database structure, making it easier to understand and query.
* **Reduced Data Anomalies:**
  + **Insertion Anomaly:** Difficulty in inserting a new record because some attributes of the primary key are not yet known.
  + **Deletion Anomaly:** Unintentional loss of related information when a record is deleted.
  + **Update Anomaly:** Needing to update the same information in multiple rows if it's duplicated.
* **Improved Query Performance (in some cases):** While highly normalized databases might require more joins to retrieve all the necessary information, the smaller, well-indexed tables can sometimes lead to faster query performance for specific data retrieval tasks.

**The Normal Forms (Simplified Overview)**

* **First Normal Form (1NF):**
  + Each column must contain atomic (indivisible) values. No multi-valued attributes or repeating groups of columns.
  + Each row must be unique (have a primary key).
* **Second Normal Form (2NF):**
  + Must be in 1NF.
  + All non-key attributes must be fully functionally dependent on the entire primary key. This means if the primary key is composite (made up of multiple columns), every non-key attribute must depend on *all* of them, not just a part.
* **Third Normal Form (3NF):**
  + Must be in 2NF.
  + All non-key attributes must be non-transitively dependent on the primary key. This means a non-key attribute should not depend on another non-key attribute.

**Trade-offs:**

While normalization reduces redundancy and improves integrity, highly normalized databases can sometimes require more joins to retrieve data, which can impact query performance. Database designers often need to find a balance between the level of normalization and the performance requirements of the application.

A screenshot of a black box

AI-generated content may be incorrect.

**Normalization Steps (Simplified)**

1. **1NF:** The table is already in 1NF as each column has atomic values and there's an implied primary key (OrderID, ProductID).
2. **2NF:** ProductName depends on ProductID, and CustomerName, CustomerAddress depend on CustomerID, not the composite primary key (OrderID, ProductID). We split the table:
   * **Orders Table:** OrderID, CustomerID
   * **OrderDetails Table:** OrderID, ProductID, Quantity
   * **Products Table:** ProductID, ProductName
   * **Customers Table:** CustomerID, CustomerName, CustomerAddress
3. **3NF:** In the Customers table, if CustomerAddress depended on another non-key attribute (e.g., ZipCode), we would further normalize it. (In this simplified example, it doesn't).

A table with numbers and text

AI-generated content may be incorrect.

**Problems:** Data redundancy and Data anomalies.

A close up of a text

AI-generated content may be incorrect.A table with text and numbers

AI-generated content may be incorrect.

**FACT Tables**

"FACT" refers to the central tables in a star or snowflake schema that contain the quantitative measurements or metrics of a business process (e.g., sales amount, order quantity, website visits). These tables are typically surrounded by dimension tables that provide the context for these facts (e.g., product, customer, date, location).

The principle of "no redundancy of FACT" means that **fact tables should primarily store the measures themselves and the foreign keys that link these measures to the relevant dimensions. They should avoid storing descriptive attributes that belong to the dimensions.**

Here's a more detailed breakdown:

* **Focus on Measures and Foreign Keys:** Fact tables are designed to record *what* happened (the event or transaction) and *when*, *where*, *who*, and *what* it involved. The "what" (the measures) are the core data, and the "when," "where," "who," and "what" (the context) are represented by foreign keys referencing the dimension tables.
* **Avoid Dimension Attributes:** Storing descriptive attributes that are already present in the dimension tables within the fact table would introduce redundancy. For example, storing the ProductName in a sales fact table along with the ProductID (which links to the Product dimension table) would be redundant. If the product name changes, you would have to update it in potentially millions of rows in the fact table, leading to maintenance overhead and potential inconsistencies. The correct approach is to retrieve the ProductName by joining the fact table with the Product dimension table using the ProductID.

**1. No Insertion, Deletion, or Update Anomalies**

**Insertion Anomaly:** This occurs when you cannot insert a new record into a table because you don't have complete information for all the required attributes, even though the new entity exists.

* **Example:** In a table with both employee and department information, you might not be able to add a new department until you have at least one employee to assign to it (if the employee information is part of the primary key or has a NOT NULL constraint). A well-normalized design would separate employees and departments into different tables, allowing you to add a new department even if no employees are currently assigned.

**Deletion Anomaly:** This occurs when deleting a record unintentionally removes related information as well.

* **Example:** In the same employee-department table, if you delete the last employee belonging to a specific department, you might also lose all information about that department if the department details are only stored with employee records. Normalization would place department information in a separate Departments table, preventing this loss.

**Update Anomaly:** This occurs when the same piece of information is stored in multiple rows, and updating it in one row but not others leads to data inconsistency.

* **Example:** If an employee's address is stored in multiple records across different projects they are working on, updating their address would require modifying all those records. Forgetting to update even one would lead to inconsistent data. Normalization would store employee details, including address, in a separate Employees table, and other tables would refer to the employee via a foreign key, requiring only a single update to change the address.

**2. No Information Loss**

This principle states that during the normalization process (decomposing larger tables into smaller ones), no essential data or relationships should be lost. It should be possible to reconstruct the original information by joining the normalized tables.

* **Achieving No Information Loss:** This is typically ensured by correctly identifying primary keys and establishing foreign key relationships between the decomposed tables. These keys act as the links that allow you to join the tables back together to retrieve the complete set of information.

**3. No Dependency Loss:**

This principle is closely related to the normal forms (especially 3NF and beyond). It means that the dependencies between attributes in the database should be logical and related to the primary key of the table they reside in. Specifically, non-key attributes should depend only on the primary key and not on other non-key attributes within the same table (transitive dependency).

* **Why it's important:** Losing dependencies can lead to the anomalies mentioned earlier. If a non-key attribute depends on another non-key attribute, updating the latter might require updating all rows where the former is also present, leading to update anomalies.
* **Example:** Consider a table Orders with OrderID (PK), CustomerID, CustomerName, and SalespersonID. CustomerName depends on CustomerID, not directly on OrderID. If we don't separate customer information into a Customers table, updating a customer's name would require updating all their orders. This is a dependency loss in terms of good design. Normalization would create a Customers table with CustomerID as the primary key and CustomerName as an attribute, and the Orders table would only store CustomerID as a foreign key.

**Functional Dependencies (FDs)**

They describe the relationship between attributes (columns) within a relation (table). They are constraints on the data that must hold true for all valid instances of the relation.

In simple terms, a functional dependency states that the value of one or more attributes in a relation **uniquely determines** the value of another attribute (or set of attributes) in the same relation.

We write a functional dependency as:

**X → Y**

where:

* **X** is a set of one or more attributes in the relation (the determinant).
* **Y** is a set of one or more attributes in the same relation (the dependent).

This notation means "the value of X functionally determines the value of Y" or "given a value for X, there is only one possible value for Y at any point in time in the relation," then "Y is functionally dependent on X."

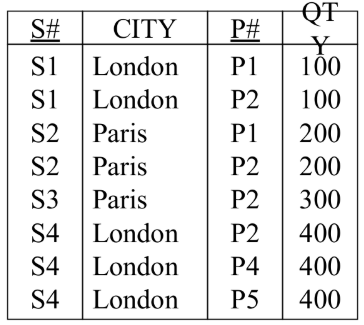
Functional Dependencies are the foundation of the normalization process. By identifying the FDs within a relation, we can determine if the relation is in a particular normal form (1NF, 2NF, 3NF, BCNF, etc.). The goal of normalization is to decompose relations in such a way that they meet the requirements of higher normal forms, which helps to eliminate data redundancy and anomalies.

**Identifying Primary Keys:** FDs help in identifying candidate keys and ultimately the primary key of a relation. A candidate key is a minimal set of attributes that functionally determines all other attributes in the relation.

**Detecting Redundancy:** If a non-key attribute is functionally dependent on only a part of the primary key (in 2NF violation) or on another non-key attribute (in 3NF violation), it indicates redundancy that can be eliminated by decomposition into separate relations.

**Guiding Decomposition:** FDs provide the rules for how to decompose a relation into smaller relations while preserving the original dependencies and ensuring no loss of information.

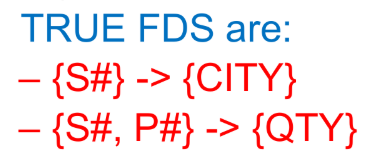
**Example:**

A black text on a white background

AI-generated content may be incorrect.

S# -- Supplier number City – Supplier city

P# -- Product number QTY – Product quantity



**What is Functional Dependency?**

In a database, a **functional dependency (FD)** occurs when one attribute uniquely determines another attribute.

In simple words:

If you know the value of **A**, you can find the value of **B**.  
This is written as: **A → B** (A determines B)

A screenshot of a computer

AI-generated content may be incorrect.

* **Student\_ID → Name**Because each Student\_ID gives **one unique Name**.
* **Student\_ID → Age**Because Student\_ID also determines the student's Age.

**Trivial Functional Dependency**

A functional dependency is **trivial** if the right-hand side is already a part of the left-hand side.

**L.H.S n R.H.S != NULL (LHS intersection RHS not equal to NULL):**

**Means [ Student\_ID 🡪 {Student\_ID, Name} ]**

Also, **when X is subset of Y** then its **trivial**, Example: **[** **Student\_ID 🡪 Student\_ID ]**

**Example:**

* **{Student\_ID, Name} → Name**  
  This is trivial because "Name" is already included on the left side.

**Non-Trivial Functional Dependency:**

It's **non-trivial** if the right side is **not** part of the left side.

**L.H.S n R.H.S = NULL (LHS intersection RHS equal to NULL):**

**Means {**

**Student\_ID 🡪 City** *or*  **Student\_ID 🡪 name** *or*  **Student\_ID 🡪 age**

**}**

**Example:**

* **Student\_ID → Name**This is non-trivial because Name is not part of Student\_ID.

**Full Functional Dependency:**

A → B is **fully functional** if removing any part of A means B is no longer determined.

A screenshot of a computer

AI-generated content may be incorrect.

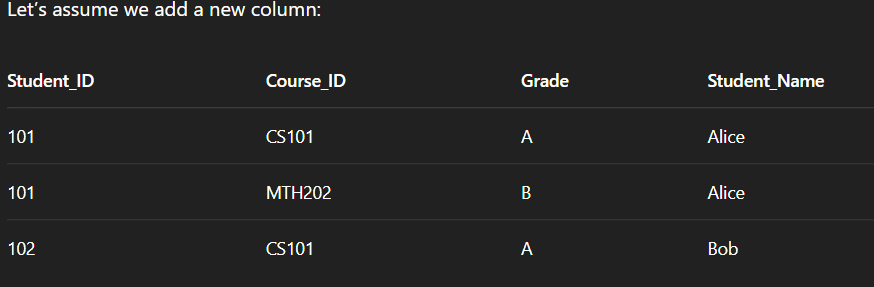
**{Student\_ID, Course\_ID} → Grade**  
Grade depends on both columns together — full dependency.

If you remove either one:

* **Student\_ID → Grade ❌ Not always true.**
* **Course\_ID → Grade ❌ Not always true.**

**Partial Dependency**

An attribute depends on part of **a composite key**, not the whole key.



Now:

* **Student\_Name depends only on Student\_ID**, not on Course\_ID.

So:

* **Student\_ID → Student\_Name**But the table's primary key is (Student\_ID, Course\_ID), and **Student\_Name** depends on only part of that key.

**Properties Of Functional Dependency:**A white board with writing on it

AI-generated content may be incorrect.

**A white board with writing on it

AI-generated content may be incorrect.**

**A white board with writing on it

AI-generated content may be incorrect.**

**A white board with red circles and black text

AI-generated content may be incorrect.**

**1. Reflexivity**

If Y is a subset of X, then X → Y holds.

Example:  
If X = **{Student\_ID, Name},**  
then **X → Student\_ID** is true  
(because Student\_ID is part of X).

**2. Augmentation**

If **X → Y**, then **XZ → YZ** also holds (for any Z).

Example:  
If **Student\_ID → Name**,  
then **Student\_ID, Course\_ID** **→** **Name, Course\_ID**  
(This means you can add the same attributes to both sides.)

**3. Transitivity**

If **X → Y** and **Y → Z**, then **X → Z**

Example:  
If **Student\_ID → Department\_ID** and **Department\_ID → Department\_Name**,  
then **Student\_ID → Department\_Name**

**4. Union**

If **X → Y** and **X → Z**, then **X → YZ**

Example:  
If Student\_ID → Name and Student\_ID → Address,  
then Student\_ID → {Name, Address}

**5. Decomposition (or Projectivity)**

Reverse of Union

If **X → YZ**, then **X → Y** and **X → Z**

Example:  
If Student\_ID → {Name, Address},  
then:

* Student\_ID → Name
* Student\_ID → Address

**6. Pseudotransitivity**

If **X → Y** and **WY → Z**, then **WX → Z**

Example:  
If Course\_ID → Instructor\_ID and Instructor\_ID, Semester → Room,  
then Course\_ID, Semester → Room

**7. Composition**

If **X → Y** and **Z → W**, then **XZ → YW**

📘 Example:  
If A → B and C → D,  
then AC → BD

**PRACTICE QUESTIONS:**

**Multiple Choice Questions (MCQs)**

**1. Which of the following is a property of Functional Dependencies?**  
A. Reflexivity  
B. Transitivity  
C. Augmentation  
D. All of the above  
**Answer:** D. All of the above

**2. If A → B and B → C, what can you infer using transitivity?**  
A. A → C  
B. C → A  
C. B → A  
D. A → B  
**Answer:** A. A → C

**3. What kind of dependency is it if part of a composite key determines a non-key attribute?**  
A. Full Dependency  
B. Transitive Dependency  
C. Partial Dependency  
D. Trivial Dependency  
**Answer:** C. Partial Dependency

**4. Which of these FDs is trivial?**  
A. A → B  
B. AB → A  
C. A → AB  
D. A → C  
**Answer:** B. AB → A (because A is a subset of AB)

**5. Which rule allows us to infer A → BC from A → B and A → C?**  
A. Union  
B. Transitivity  
C. Decomposition  
D. Augmentation  
**Answer:** A. Union

**🔹 Short Questions with Answers**

**Q1: What is a functional dependency?**  
**A:** A functional dependency (FD) in a relation means that one attribute uniquely determines another. If A → B, then knowing A lets us find exactly one B.

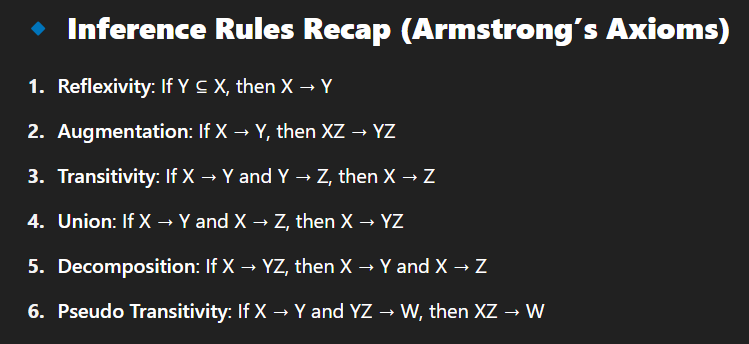
**Q2: What is the difference between full and partial dependency?**  
**A:**

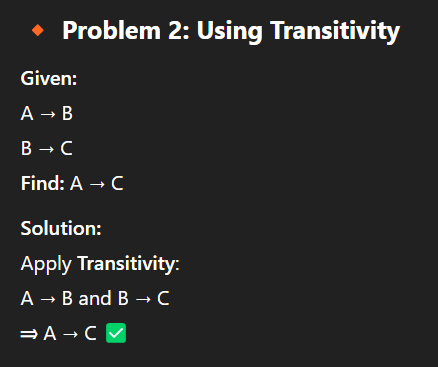
* **Full Dependency**: A non-key attribute depends on the **whole** composite key.
* **Partial Dependency**: A non-key attribute depends on **part of** the composite key.

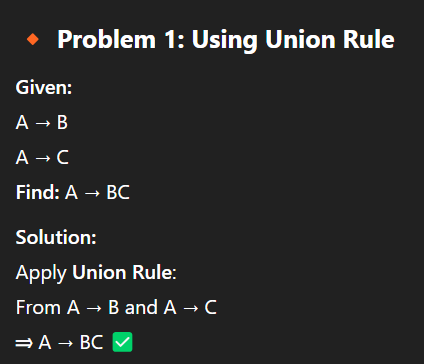
**Q3: What is a trivial functional dependency?**  
**A:** An FD is trivial if the right-hand side is a subset of the left-hand side. Example: AB → A.

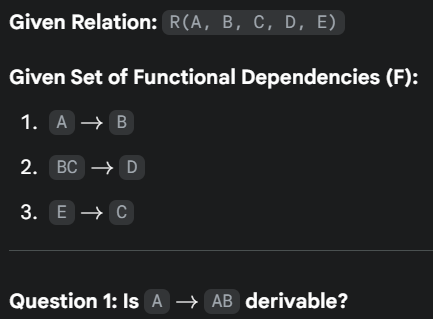
**Q4: What is the transitive rule in functional dependencies?**  
**A:** If A → B and B → C, then A → C. This is called the transitive property.

**Q5: Why are functional dependencies important in normalization?**  
**A:** FDs help detect and eliminate redundancy by identifying how attributes depend on each other, which guides normalization up to 3NF or BCNF.









A black background with white text

AI-generated content may be incorrect.