# Database Normalization



---

## Introduction to Normalization Theory

> **How to design a "good" DATABASE?**

**Informal definition:**

- The schema should represent a "distinct" entity
- Little or no redundancy.
- Lesser no. of null values
- No modification anomaly
- No spurious tuple

> **Normalization theory helps us in designing a good database in a formal manner.**

## Database Anomalies

Consider the following schema: `(empid, empname, projid, projname)`

### Insertion Anomaly

- Inserting a new employee requires us to assign a project and vice versa.

### Update anomaly

- Updating the name of a project requires us to update all the employees who are assigned to that project.

### Delete anomaly

- Deleting a project might require us to delete all its employees.

## Decomposition and spurious tuples

- https://www.geeksforgeeks.org/spurious-tuples-in-dbms/
- https://www.geeksforgeeks.org/difference-between-lossless-and-lossy-join-decomposition/

> ### Make sure that we preserve the losslessness of the corresponding join while decomposing a table

## Functional Dependencies

- **Functional dependencies (FDs)** are ***constraints*** derived from the meaning of and relationships among attributes
- A set of attributes X **functionally determines** Y, denoted by `X → Y`, if the value of X determines a ***unique*** value of Y
- https://www.geeksforgeeks.org/introduction-of-database-normalization/

> ### Try to preserve the FD while splitting

### Examples

- `employees(empid, name)` with FD:
  - `(empid) -> name)`
- `users(uuid, email, name)` with FDs:
  - `(uuid, email) -> (email)`, **trivial FD**
  - `(uuid, email) -> (email, name)`, **semi non-trivial FD**
  - `(uuid, email) -> (name)`, **completely non-trivial FD**

## Normalization

- The process of decomposing relations into smaller relations that conform to certain norms is called **normalization**
- Keys and FDs of a relation determine which normal form a relation is in.
- Different normal forms
  - **1NF:** based on attributes only
  - **2NF, 3NF, BCNF:** based on keys and FDs
  - **4NF:** based on keys and multi-valued dependencies (MVDs)
  - **5NF or PJNF:** based on keys and join dependencies
  - **DKNF:** based on all constraints

> For now, we are just concerned about 1NF, 2NF, and 3NF

### Prime attribute, full functional dependency and transitive function dependency

- A **prime attribute** must be a member of some candidate key
  - Example: `roll`
- A **non-prime attribute** is not a member of any candidate key
  - Example: `gender`
- An FD `X → Y` is a **full functional dependency** if the FD does not hold when any attribute from X is removed

    - Example: `(roll) → (name)`
- It is a **partial functional dependency** otherwise
    - `(roll, gender) → (name)`
- A FD `X → Y` is a **transitive functional dependency** if it can be derived from two FDs `X → Z` and `Z → Y`
    - Example: `(roll) → (hod)` since `(roll) → (deptid)` and `(deptid) → (hod)` hold
- It is **non-transitive** otherwise
    - Example: `(roll) → (name)`

## Normal Forms

**Informal definition:**

- **1NF:** All attributes depend on the key
- **2NF:** All attributes depend on the whole key
- **3NF:** All attributes depend on nothing but the key

**Tests for normal forms:**

- **1NF:** The relation should have no multivalued attributes or nested relations
- **2NF:** For a relation where the candidate key contains multiple attributes, no non-key attribute should be functionally dependent on a part of the candidate key
- **3NF:** The relation should not have a non-key attribute functionally determined by a set of non-key attributes

**Remedies for normal forms:**

- **1NF:** Form new relations for each multi-valued attribute or nested relation
- **2NF:** Decompose and set up a relation for each partial key with its dependent(s); retain the primary key and attributes fully dependent on it
- **3NF:** Decompose and set up a relation for each non-key attribute with non-key attributes functionally dependent on it

# Examples

## Example 1 (2NF)

- Consider `( Id , ProjId , Hrs, Name, ProjName)` with FDs:
    - `(Id, ProjId) → (Hrs)`
    - `(Id) → (Name)`
    - `(ProjId) → (ProjName)`
- It is not in **2NF** since `(Name)` depends partially on `(Id, ProjId)`
- After **2NF** normalization,
    - `(Id, ProjId, Hrs)` with FD: `(Id, ProjId) → (Hrs)`
    - `(Id, Name)` with FD: `(Id) → (Name)`
    - `(ProjId, ProjName)` with FD: `(ProjId) → (ProjName)`

## Example 2 (3NF)

- Consider `( empid , empname, projid, projname)` with FDs:
    - `(empid) -> (empname, projid)`
    - `(projid) -> (projname)`
- it is not in **3NF** since `(projname)` depends transitively on `(Id)` through `(ProjId)`.
- After **3NF** normalization,
    - `( empid , empname, projid)` with FD: `(empid) -> (empname, projid)`
    - `( projid , projname)` with FD: `(projid) -> (projname)`

## Example 3 (2NF, 3NF)

- `L = (Id, Dist, Lot, Area, Price, Rate)` with FDs:
    - `(Id) → (Dist, Lot, Area, Price, Rate)`
    - `(Dist, Lot) → (Id, Area, Price, Rate)`
    - `(Dist) → (Rate)`
    - `(Area) → (Price)`
- L is not in **2NF** because `(Rate)` depends partially on `(Dist)`
- `L1 = (Id, Dist, Lot, Area, Price)` with FDs:
    - `(Id) → (Dist, Lot, Area, Price)`

  - `(Dist, Lot) → (Id, Area, Price)`
  - `(Area) → (Price)`
- `L2 = (Dist, Rate)` with FD:
  - `(Dist) → (Rate)`
- L1 is in **2NF** but not **3NF** because `(Price)` depends on `(Id)` through `(Area)`
- L2 is in **2NF** and in **3NF**
- `L11 = (Id, Dist, Lot, Area)` with FDs:
  - `(Id) → (Dist, Lot, Area)`
  - `(Dist, Lot) → (Id, Area)`
- `L12 = (Area, Price)` with FD:
  - `(Area) → (Price)`
- L11 and L12 are in **3NF**

# References


- https://www.geeksforgeeks.org/normal-forms-in-dbms/
- https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description