# Functional dependency

- In DBMS, functional dependency is a constraint between two sets of attributes in a relation (or table). It is a property that describes the relationship between two or more attributes in a relation.
- A functional dependency is a rule that specifies the dependency of one attribute on another attribute(s) in a relation.
- In other words, a functional dependency between two attributes A and B in a relation R means that for every unique value of attribute A, there exists a unique value of attribute B. This means that the value of B is determined by the value of A, and therefore, attribute B is said to be functionally dependent on attribute A.

## Functional dependency

- For example, consider a relation "employees" with attributes (employee_id, employee_name, department_id, department_name). Here, department_name is functionally dependent on department_id, as each department has a unique name associated with it. Similarly, employee_name is functionally dependent on employee_id, as each employee has a unique name associated with them.
- Functional dependencies are important in database design, as they help to eliminate redundancy and improve the efficiency of data retrieval and updates.

## Attribute closure

- In DBMS, the closure of an attribute set with respect to a set of functional dependencies is the set of all attributes that are functionally determined by that set of attributes, according to the given functional dependencies.
- More specifically, given a relation schema R with attributes A1, A2, ..., An, and a set of functional dependencies F, the closure of a set of attributes X, denoted by X+, is the set of all attributes that can be determined by X through the functional dependencies in F.
- For example, consider a relation schema R with attributes (A, B, C, D) and functional dependencies F = {A -> B, B -> C, C -> D}. The closure of attribute set {A} is {A, B, C, D}, since A determines B, B determines C, and C determines D. Similarly, the closure of attribute set {A, B} is {A, B, C, D}, since A and B together determine C and D.
- Attribute closure is important in database design, as it can be used to determine the keys of a relation schema and to identify potential redundancies in the schema. It can also be used in query optimization, as it can help to identify the minimal set of attributes that need to be retrieved in order to satisfy a given query.

# Armstrong's axioms

- Armstrong's axioms are a set of rules that can be used to infer all functional dependencies that hold in a relation based on a given set of functional dependencies. The axioms are named after William W. Armstrong, who first formulated them in the 1970s.
- The three axioms are:
- Reflexivity: If Y is a subset of X, then X -> Y.

  Example: If {A, B} -> {A} holds, then {A, B} -> {A, B} also holds.
- Augmentation: If X -> Y holds, then XZ -> YZ also holds for any set of attributes Z.

  Example: If {A} -> {B} holds, then {A, C} -> {B, C} also holds.
- Transitivity: If X -> Y and Y -> Z hold, then X -> Z also holds.

  Example: If {A} -> {B} and {B} -> {C} hold, then {A} -> {C} also holds.
- These three axioms can be used to derive all other functional dependencies that hold in a relation. For example, if we have the functional dependencies {A -> B} and {B -> C}, we can use the transitivity axiom to infer that {A -> C} also holds.
- Armstrong's axioms are important in database design, as they provide a systematic way to determine all the functional dependencies that hold in a relation. This information can be used to identify keys and superkeys, and to eliminate redundancies in the relation. They are also used in the process of normalizing a relation into a higher normal form.

# Equivalence of Functional Dependencies

- Definition:
  - Two or more than two sets of functional dependencies are called equivalence if the right-hand side of one set of functional dependency can be determined using the second FD set, similarly the right-hand side of the second FD set can be determined using the first FD set.

## canonical cover

In DBMS,

- A canonical cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, it is also called as Irreducible set.

# Types of Keys in Relational Model

## Super Key:

- Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation. All super keys can't be candidate keys but the reverse is true. In relation, a number of super keys is more than a number of candidate keys.

## Candidate Key -

- The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.

## Primary Key -

- The primary key is selected from one of the candidate keys and becomes the identifying key of a table. It can uniquely identify any data row of the table.

# Types of Keys in Relational Model

## Composite Key -

- If any single attribute of a table is not capable of being the key i.e it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.

## Secondary Key -

- Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.

## Foreign Key -

- A foreign key is an attribute value in a table that acts as the primary key in another table. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.

## Types of Keys in Relational Model

### Composite Key -

- If any single attribute of a table is not capable of being the key i.e it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.

### Secondary Key -

- Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.

### Foreign Key -

- A foreign key is an attribute value in a table that acts as the primary key in another table. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.

# Normalization

- Normalization

  A large database defined as a single relation may result in data duplication. This repetition of data may result in:

  - Making relations very large.
  - It isn't easy to maintain and update data as it would involve searching many records in relation.
  - Wastage and poor utilization of disk space and resources.
  - The likelihood of errors and inconsistencies increases.

  So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties.

- Normalization is a process of decomposing the relations into relations with fewer attributes.

# Normalization

- What is Normalization?
  - Normalization is the process of organizing the data in the database.
  - Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
  - Normalization divides the larger table into smaller and links them using relationships.
  - The normal form is used to reduce redundancy from the database table.

## Normalization

- Why do we need Normalization?
  - The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.
  - Data anomalies can be categorized into three types:
    - Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
    - Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
    - Updatation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

## Normalization

- First Normal Form
  - A relation is in first normal form if every attribute in that relation is singled valued attribute.
- Second Normal Form
  - To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
  - Partial Dependency - If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

# Normalization

Example 1 - Consider table.

| STUD_NO | COURSE_NO | COURSE_FEE |
|---|---|---|
| 1 | C1 | 1000 |
| 2 | C2 | 1500 |
| 1 | C4 | 2000 |
| 4 | C3 | 1000 |
| 4 | C1 | 1000 |
| 2 | C5 | 2000 |

- {Note that, there are many courses having the same course fee. }

# Normalization

Here,

COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;

COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;

COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,

COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,

- we need to split the table into two tables such as :
- Table 1: STUD_NO, COURSE_NO
- Table 2: COURSE_NO, COURSE_FEE

## Normalization

Table 1

| STUD_NO | COURSE_NO |
|---|---|
| 1 | C1 |
| 2 | C2 |
| 1 | C4 |
| 4 | C3 |
| 4 | C1 |

Table 2

| COURSE_NO | COURSE_FEE |
|---|---|
| C1 | 1000 |
| C2 | 1500 |
| C3 | 1000 |
| C4 | 2000 |
| C5 | 2000 |

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

# Normalization

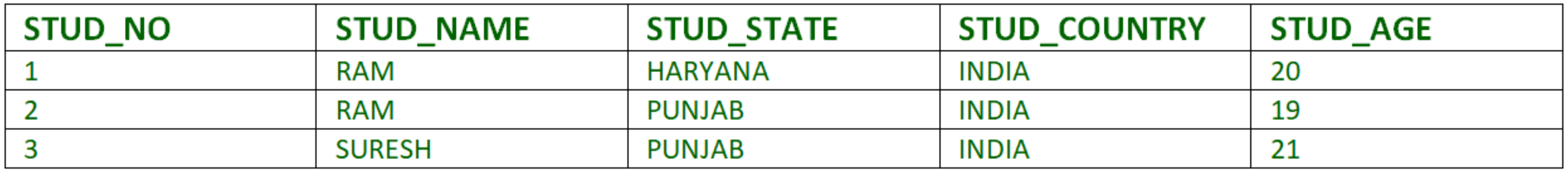
## Third Normal Form

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X -> Y

- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).

# Normalization



| STUD_NO | STUD_NAME | STUD_STATE | STUD_COUNTRY | STUD_AGE |
|---|---|---|---|---|
| 1 | RAM | HARYANA | INDIA | 20 |
| 2 | RAM | PUNJAB | INDIA | 19 |
| 3 | SURESH | PUNJAB | INDIA | 21 |

**Table 4**

Transitive dependency - If A->B and B->C are two FDs then A->C is called transitive dependency.

- Example 1 - In relation STUDENT

FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}

Candidate Key: {STUD_NO}

For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form

## Normalization

To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)

STATE_COUNTRY (STATE, COUNTRY)

# Normalization

- Boyce-Codd Normal Form (BCNF) -
  - A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key.
  - X -> Y, X is a super key.

# Normalization

- Lossless-join decomposition/non-additive join decomposition
  - Lossless-join decomposition is a process in which a relation is decomposed into two or more relations. This property guarantees that the extra or less tuple generation problem does not occur and no information is lost from the original relation during the decomposition. It is also known as non-additive join decomposition.
  - When the sub relations combine again then the new relation must be the same as the original relation was before decomposition.
  - It is a mandatory property it always holds good.
  - If a relation R is decomposed into two relations R1 & R2 then it will be lossless iff:
    - Attribute (R1) U attribute(R2)=attribute(R)
    - Attribute (R1)∩ attribute(R2)≠ Φ
    - Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)