Unit 3. Normalization

3.1. Importance of Normalization

3.2. Functional Dependencies-– definition, trivial and non-trivial FD, closure of FD set, closure of attributes

3.3. Integrity and Domain constraints

3.4. Normal forms (1NF, 2NF, 3NF, BCNF)

3.3. Integrity and Domain constraints

Introduction to Relational Model

- The **relational model** is the **theoretical** basis of relational databases
- The relational model of data is based on the concept of **relations**
- A "**Relation**" is a mathematical concept based on the ideas of **sets**
- The **Relational Model** was proposed by **E.F. Codd** for IBM in 1970 **to model data in the form of relations or tables.**

What is Relational Model?

- Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables).
  - After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages
  - RDMBS languages: Oracle, SQL, MySQL etc.

What is RDBMS?

- RDBMS stands for: **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A **Relational database management system** (**RDBMS**) is a database management system (DBMS) that is based on the **relational model** as introduced by E. F. Codd.
- Current popular **RDBMS** include:
  - DB2 & Informix Dynamic Server - *from IBM*
  - Oracle & Rdb - *from Oracle*
  - SQL Server & MS Access - *from Microsoft*

Relational Model concept

- **Relational model** can be represented as a table with ***columns*** and ***rows***.
  - Each **row** is known as a tuple.
  - Each table of the **column** has a name or attribute.

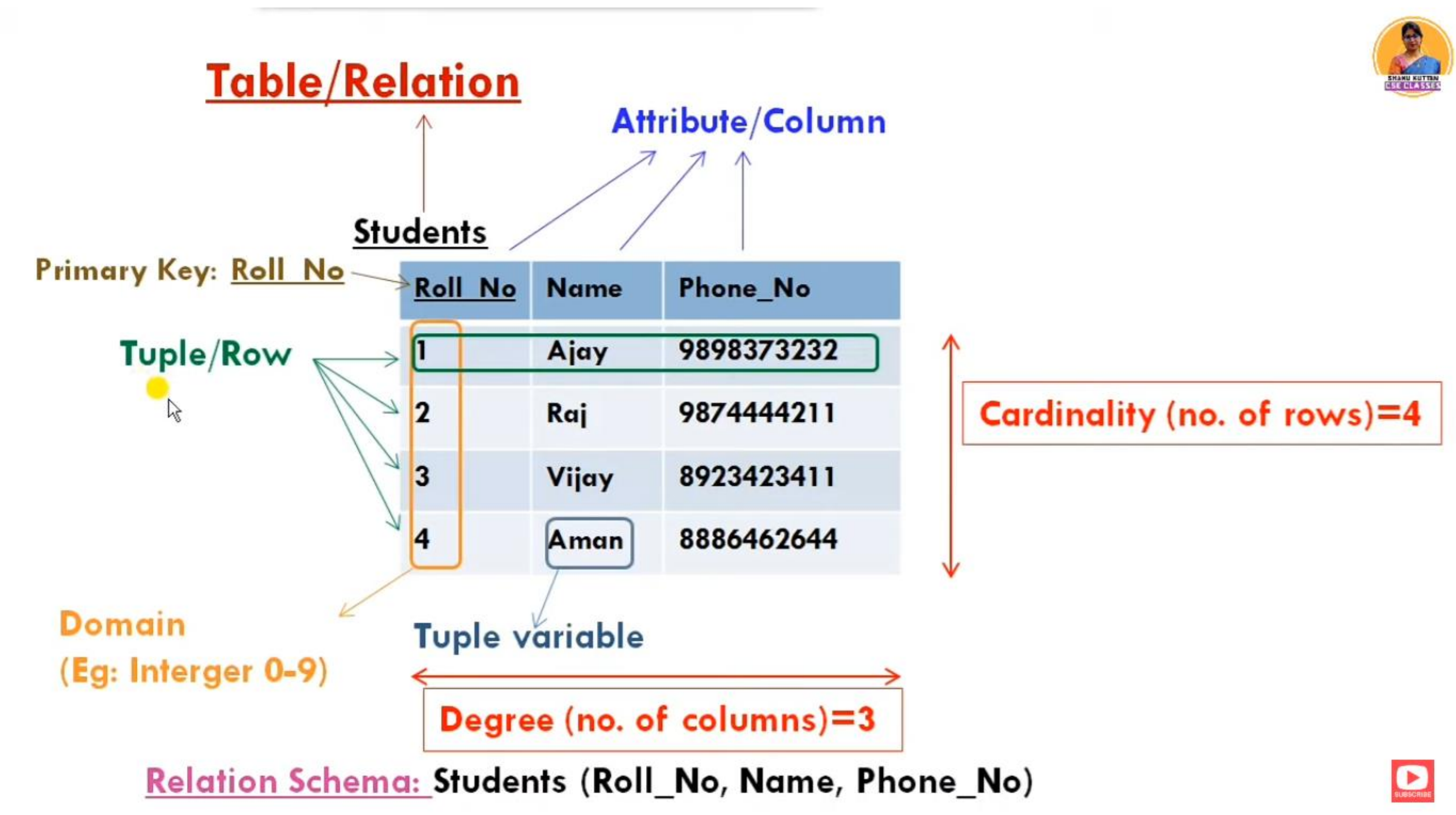
Students

Roll No	Name	Phone No
1	Ajay	9898373232
2	Raj	9874444211
3	Vijay	8923423411
4	Aman	8886462644

Relational Model concept

- **Relation:** A relation is a table with columns and rows.
- **Attribute:** An attribute is a named column of a relation.
- **Domain:** A domain is the set of allowable values for one or more attributes.
- **Tuple:** A tuple is a row of a relation.
- **Relation Schema:** A relation schema represents the name of the relation with its attributes.
- **Relation instance** (State): Relation instance is a finite set of tuples. Relation instances never have duplicate tuples.
- **Degree:** The total number of columns or attributes in the relation
- **Cardinality:** Total number of rows present in the Table.
- **Relation key:** Every row has one or multiple attributes, that can uniquely identify the row in the relation, which is called relation key (Primary key).
- **Tuple Variable:** it is the data stored in a record of the table

Roll_No	Name	Phone_No
1	Ajay	9898373232
2	Raj	9874444211
3	Vijay	8923423411
4	Aman	8886462644



Properties of Relational Model

- Each **Relation** has **unique name**
- Each **tuple/Row** is **unique**: No duplicate row
- Entries in any **column** have the **same domain.**
- Each **attribute/column** has a **unique name**
- **Order** of the columns or rows is **irrelevant** i.e *relations are unordered*
- Each **cell** of relation contains exactly **one value** i.e. *attribute values are required to be atomic*

Students

Roll_No	Name	Phone_No
1	Ajay	9898373232
2	Raj	9874444211
3	Vijay	8923423411
4	Aman	8886462644

Alternative Terminology for Relational Mode

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Integrity Constraints over Relation

- **Integrity constraints** are used *to ensure accuracy and consistency of the data in a relational database.*
- **Integrity constraints** are **set of rules** that the database is not permitted to violate.
- **Constraints** may apply to each **attribute** or they may apply to **relationships between tables.**
- **Integrity constraints** ensure that changes (update, deletion, insertion) made to the database by authorized users do not result in a loss of data consistency. Thus, ***integrity constraints guard against accidental damage to the database.***
  - **Example -** A blood group must be 'A' or 'B' or 'AB' or 'O' only (can not any other values else).

- *In DBMS (Database Management System), constraints are rules or restrictions enforced on the data in a database to ensure its accuracy, consistency, and reliability.*

- *Constraints are used to prevent invalid data from being entered into the database and to maintain the integrity of the database*

TYPES OF INTEGRITY CONSTRAINT

1. **Domain Constraint**
2. **Entity Integrity Constraint**
3. **Referential Integrity Constraint**
4. **Key Constraints**

1. Domain Constraints

- **Domain constraints** defines the domain or the valid set of values for an attribute.
- The **data type of domain** includes string, character, integer, time, date, currency, etc. *The value of the attribute must be available in the corresponding domain.*

STUDENT_ID	NAME	SEMESTER	AGE
101	Manish	1st	18
102	Rohit	3rd	19
103	Badal	5th	20
104	Amit	7th	A

Not allowed. Because AGE is an integer value

2. Entity Integrity Constraints

- The **entity integrity constraint** states that **primary key value can't be null.**
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

EMP_ID	EMP_NAME	SALARY
111	Mohan	20000
112	Rohan	30000
113	Sohan	35000
	Logan	20000

Not allowed as Primary Key can't contain NULL value

3. Referential Integrity Constraints

- A **referential integrity constraint** is specified between two tables.
- **Referential Integrity constraint** is enforced when a foreign key references the primary key of a table.
- In the **Referential integrity constraints**, if a foreign key in Table 1 refers to the Primary Key of Table 2, then ***either*** every value of the foreign Key in Table 1 must be available in primary key value of Table 2 **or** it must be null.
- The **rules** are:
  - You can't **delete** a record from a *primary table* if matching records exist in a related table.
  - You can't **change** a primary key value in the *primary table* if that record has related records.
  - You can't **insert** a value in the foreign key field of the *related table* that doesn't exist in the primary key of the *primary table*.
  - However, you can enter a **Null** value in the foreign key, specifying that the records are unrelated.

Foreign Key

EMP (Table 1): Related /Referencing Table

EMP_ID	EMP_NAME	AGE	DEP_NO
111	Mohan	21	1
112	Rohan	33	2
113	Sohan	27	3
114	Logan	25	5

Relationships

Not allowed as DEP_NO 5 is not defined as a Primary Key of Table 2 and In Table 1, DEP_NO is a foreign key defined

Primary Key

DEP (Table 2): Primary/Referenced Table

DEP_NO	LOCATION
1	Mumbai
2	Delhi
3	Noida

4. Key Constraints

- An entity set can have multiple keys or candidate keys (minimal superkey), but out of which one key will be the primary key.
- **Key constraint** specifies that in any relation-
  - All the values of primary key must be **unique.**
  - The value of primary key must **not be null.**

STUDENT_ID	NAME	SEMESTER	AGE
101	Manish	1st	18
102	Rohit	3rd	19
103	Badal	5th	20
102	Amit	7th	21

Not allowed. Because all rows must be **unique**

3.1. Importance of Normalization

Normalization is a process in a Database Management System (DBMS) used to organize data in a database by *reducing redundancy and improving data integrity*. It involves dividing large tables into smaller ones and defining relationships between them to minimize duplicate data.

PatientID	FirstName	LastName	Gender	Age	Address	PhoneNumber	AdmissionDate	DischargeDate	DoctorName	Diagnosis
1	Sajan	Shrestha	Male	32	Kathmandu, Nepal	9876543210	2023-01-15	2023-01-25	Dr. Sharma	Common Cold
2	Aarati	Dahal	Female	45	Pokhara, Nepal	9843210765	2023-02-02	2023-02-10	Dr. Khanal	Hypertension
3	Rajendra	Gurung	Male	28	Biratnagar, Nepal	9812345678	2023-03-12	2023-03-20	Dr. Bhattarai	Gastritis
4	Sunita	Magar	Female	60	Butwal, Nepal	9867543210	2023-04-05	2023-04-15	Dr. Thapa	Arthritis
5	Bibek	Dhakal	Male	38	Dharan, Nepal	9801234567	2023-05-20	2023-06-02	Dr. Acharya	Diabetes
2	Aarati	Dahal	Female	45	Pokhara, Nepal	9843210765	2023-02-02	2023-02-10	Dr. Khanal	Hypertension

Objectives of Normalization

1. Eliminate redundant data.
2. Ensure data dependency is logical.
3. Reduce the chances of data anomalies (Insertion, Update, Deletion anomalies).

- Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like ***Insertion, Update and Deletion Anomalies.***
- Normalization rules divides larger tables into smaller tables and links them using relationships.
- The purpose of Normalization in SQL is to ***eliminate redundant (repetitive) data and ensure datais stored logically.***
- 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***.

```
| PatientID | FirstName | LastName | Gender | Age | Address | PhoneNumber |

| PatientID | DoctorName | Diagnosis

| PatientID || RoomNumber |  DischargeDate | AdmissionDate
```

Why Do We Need Normalization?

➢ As we have discussed above, normalization is used to reduce data redundancy.
➢ It provides a method to remove the following anomalies from the database and bring it to a more consistent state.

➢ Below are the key reasons why normalization is needed:

1. Eliminate Data Redundancy

- **Problem**: Redundant data leads to wastage of storage space and the risk of inconsistencies.
- **Solution**: Normalization removes duplicate data by splitting large tables into smaller, related tables.
- **Example**: Instead of repeating instructor details in every row of a course table, store it in a separate Instructor table.

2. Avoid Data Anomalies

Normalization resolves three types of anomalies:

1. **Insertion Anomaly**:

- Without normalization, adding new data might require entering irrelevant data.
- Example: Adding a new course without students creates NULL values in a non-normalized table.

2. **Update Anomaly**:
   - Changes in one place must be updated everywhere, risking inconsistencies.
   - Example: Updating an instructor's contact information in multiple rows.

3. **Deletion Anomaly**:
   - Deleting specific data might accidentally remove important related data.
   - Example: Deleting a student's enrollment removes the entire course record.

3. Improve Data Integrity

- Normalization ensures data is logically stored and adheres to predefined rules (constraints).
- Dependencies like foreign keys ensure relationships remain valid across tables.

4. Enhance Query Performance

- Although normalization might involve more joins between tables, it ensures data retrieval is more consistent and meaningful, improving query accuracy.

5. Maintain Scalability

- A normalized database can easily accommodate changes like adding new fields, relationships, or tables without disrupting the existing structure.

6. Enforce Business Rules

- Normalization reflects real-world relationships, making it easier to enforce business rules within the database.

- Example: Ensuring that every enrolled student must belong to a valid course.

3.2. Functional Dependencies – definition,

trivial and non-trivial FD, closure of FD set, closure of attributes

https://github.com/sanjeevlcc/notes_2081/blob/main/DBMS_BIM_BSCIT_BCA/notes/Normalization/Functional%20Dependency%20and%20Normalisation.txt

What are anomalies in DBMS? / Anomalies in Relational Model

- A database anomaly is a fault in a database that usually emerges as a result of shoddy planning and storing everything in a flat database.
- In most cases, this is removed through the normalization procedure, which involves the joining and splitting of tables.
- These anomalies can be categorized into three types:
  - Insertion Anomalies
  - Deletion Anomalies
  - Update Anomalies.

How Are Anomalies Caused in DBMS?

Anomalies in DBMS are caused by poor management of storing everything in the flat database, lack of normalization, data redundancy, and improper use of primary or foreign keys. These issues result in inconsistencies during insert, update, or delete operations, leading to data integrity problems. The three primary types of anomalies are:

How Anomalies can be removed?

Anomalies can be removed with the process of Normalization. Normalization involves organizing data into tables and applying rules to ensure data is stored in a consistent and efficient

- **Insertion Anomalies:** These anomalies occur when it is not possible to insert data into a database because the required fields are missing or because the data is incomplete. For example, if a database requires that every record has a primary key, but no value is provided for a particular record, it cannot be inserted into the database.
- **Deletion anomalies:** These anomalies occur when deleting a record from a database and can result in the unintentional loss of data. For example, if a database contains information about customers and orders, deleting a customer record may also delete all the orders associated with that customer.

- **Update anomalies:** These anomalies occur when modifying data in a database and can result in inconsistencies or errors. For example, if a database contains information about employees and their salaries, updating an employee's salary in one record but not in all related records could lead to incorrect calculations and reporting.

These anomalies can be removed with the process of Normalization, which generally splits the database which results in reducing the anomalies in the

Functional dependencies

- In relational database management, functional dependency is a concept that specifies the relationship between two sets of attributes where one attribute determines the value of another attribute.
- It is denoted as **X → Y**,
  - where the attribute set on the left side of the arrow, **X** is called **Determinant**, and **Y** is called the **Dependent**.

It is denoted by X → Y (Y depends upon X).

emp_id → name

:

Here X is known as determinant of functional dependency.

X → Y

determinant dependent

emp_id → name

EXAMPLE SCNERIO

emp_id → name *// VALID*

6 → Suman Shrestha

7 → Sarita Rai

emp_id	name	citizenship_id	email_id	department_id
6	Suman Shrestha	NEP-123	suman@example.com	105
7	Sarita Rai	NEP-456	sarita@example.com	106
8	Raj Gurung	NEP-789	raj@example.com	107
9	Bibek Paudel	NEP-012	bibek@example.com	105
10	Anita Rana	NEP-345	anita@example.com	108

emp_id → name *// VALID*

6 → Suman Shresth

6 → Suman Shresth

emp_id → name *// VALID*

6 → Suman Shrestha

7 → Suman Shrestha

emp_id → name *// INVALID*

6 → Suman Shrestha

6 → Sarita Rai

Example.

Consider the following relation:

Professor (Pfcode, Dept, Head, Time) It is assumed that

(i) A professor can work in more than one dept.

(ii) The time he spends in each dept is given.

(iii) Each dept has only one head.

Draw the dependency diagram for the given relation by identifying the dependencies.

Pfcode → Dept

Dept → Head

Pfcode, Dept → Time

OR

R1 or T1 (Pfcode → Dept , Dept → Head, Pfcode Dept → Time)

Pfcode → Dept , Dept → Head, Pfcode Dept → Time

---A----------B---------B-------C----------------AB-----------D---

R1: FD (A → B, B → C, AB → D)

Functional Dependency

- A functional dependency occurs when one attribute uniquely determines another attribute within a relation. It is a constraint that describes how attributes in a table relate to each other. If attribute A functionally determines attribute B we write this as the **A→B**.
- Functional dependencies are used to mathematically express relations among database entities and are very important to understanding advanced concepts in Relational Database Systems.

Prime and Non-Prime Attributes

For a given relation R= {A1, A2, A3,………..,An} an attribute A is a prime attribute if A is a part of any candidate key of R otherwise A is a non-prime attribute

EXAMPLE

R (A, B, C, D, E, F, G, H)

FD = {D→AB, B→A, C→A, F→G, H→FGD, E→A}

Solution

RHS: AB A A G FGD A =>> C E H is missing

CK = CEH

Prime Attributes = CEH

Non-Prime Attributes = ABDFG

CEH ABDFG

--PA--------NPA----

Example

Functional dependencies:

{emp_id} -> {name, citizenship_id, email_id, department_id}

{email_id} -> {name}

{department_id} -> {citizenship_id, email_id}

```
+--------+----------------+----------------+--------------------+---------------+
| emp_id | name           | citizenship_id | email_id           | department_id |
+--------+----------------+----------------+--------------------+---------------+
   A          B                 C                D                    E
```

{emp_id} -> {name, citizenship_id, email_id, department_id}

A -> BCDE

{email_id} -> {name}

D-> B

{department_id} -> {citizenship_id, email_id}

E-> CD

FD:

A -> BCDE

D-> B

E->CD

RHS: A

A+ -> A -> A BCDE = ABCD = R

CK = A

PA = A

NPA = BCDE

Prime Attributes: PA = A

emp_id

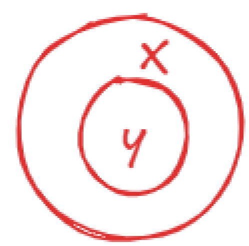
Non-Prime Attributes: NPA = BCDE

name, citizenship_id, email_id, department_id

```
+--------+---------------+---------------+------------------+--------------+
| emp_id | name          | citizenship_id | email_id        | department_id |
+--------+---------------+---------------+------------------+--------------+
   A           B                C                D                   E
----PA---------NPA--------------NPA----------------NPA-------------------NPA------
```

Armstrong's axioms/properties of functional dependencies: [RAT rule]

1. **Reflexivity:** If Y is a subset of X, then X→Y holds by reflexivity rule



   Example, {roll_no, name} → name is valid.

2. **Augmentation:** If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.

   Example, {roll_no, name} → dept_building is valid, hence

   {roll_no, name, dept_name} → {dept_building, dept_name} is also valid.

3. **Transitivity**: If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.

   Example, roll_no → dept_name & dept_name → dept_building,

   Then roll_no → dept_building is also valid.

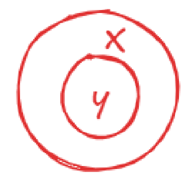
Types of Functional Dependencies in DBMS

1. ***Trivial functional dependency***
2. ***Non-Trivial functional dependency***
3. ***Multivalued functional dependency***
4. ***Transitive functional dependency***

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.

i.e. If **X** → **Y** and **Y is the subset of X**, then it is called trivial functional dependency



Example:

roll_no	**name**	**age**
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name}** → **name** is a trivial functional dependency,

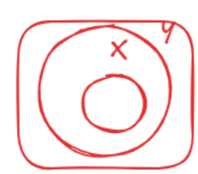
since the dependent **name** is a subset of determinant set **{roll_no, name}.**

Similarly, **roll_no** → **roll_no** is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If **X** → **Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.



Example:

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no.**

Similarly, **{roll_no, name} → age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.** i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, **roll_no → {name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

4. Transitive Functional Dependency [3NF]

In transitive functional dependency, dependent is indirectly dependent on determinant. i.e.

If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency.**

For example,

enrol_no	**name**	**dept**	**building_no**
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no → dept** and **dept → building_no.** Hence, according to the axiom of transitivity, **enrol_no → building_no** is a valid functional dependency.

This is an indirect functional dependency, hence called Transitive functional dependency.

5. Fully Functional Dependency

If X and Y are an attribute set of a relation, Y is fully functional dependent on X, if Y is functionally dependent on X but not on any proper subset of X.

Example

In the relation ABC->D, attribute D is fully functionally dependent on ABC and not on any proper subset of ABC.

That means that subsets of ABC like AB, BC, A, B, etc cannot determine D.

Let us take another example – **Supply table**

- From the table, we can clearly see that neither supplier_id nor item_id can uniquely determine the price but both supplier_id and item_id together can do so.

Supply table

supplier_id	item_id	price
1	1	540
2	1	545
1	2	200
2	2	201
1	1	540
2	2	201
3	1	542

- So we can say that **price** is fully functionally dependent on **{ supplier_id, item_id }**. This summarizes and gives our fully functional dependency −

  { supplier_id , item_id } -> price

6. Partial Functional Dependency [2NF]

- A functional dependency X->Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.

- For example, we have a relationship AC->B, A->D, and D->B.
  Now if we compute the closure of {A+}=ADB

Here A is alone capable of determining B, which means B is partially dependent on AC.

Let us take another example –

Here, we can see that both the attributes name and roll_no alone are able to uniquely identify a course. Hence we can say that the relationship is partially dependent.

Student table

name	roll_no	course
Ravi	2	DBMS
Tim	3	OS
John	5	Java

Closure of a Set

Closure of an Attribute can be defined as a set of attributes that can be functionally determined from it.

OR

Closure of a set F of FDs is the set F+ of all FDs that can be inferred from F

Closure of a set of attributes X concerning F is the set X+ of all attributes that are functionally determined by X

Example

Consider the relation scheme

R = {E, F, G, H, I, J, K, L, M, N} and

the set of functional dependencies

{{E, F} -> {G},

{F} -> {I, J},

{E, H} -> {K, L},

K -> {M},

L -> {N} on R.

What is the key for R?

Solution:

Finding attribute closure of all given options, we get:

{E,F}+ = {EFGIJ} $\neq$ R

{E,F,H}+ = {EFHGIJKLMN} = R

{E,F,H,K,L}+ = {{EFHGIJKLMN} = R

{E}+ = {E} $\neq$ R

{EFH}+ and {EFHKL}+ results in set of all attributes $\neq$ R,

but EFH is minimal.

So EFH will be candidate key.

Example

Consider the relation schema R = {H, D, X, Y, Z} and the functional dependencies

X→YZ, DX→W, Y→H Find the closure F+ of FD's.

Sol.

Applying Decomposition on X→YZ gives X→Y and X→Z

Applying Transitivity on X→Y and Y→H gives X→H

Thus the closure F+ has the FD's X→YZ, DX→W, Y→H, X→Y, X→Z, X→H

Advantages of Functional Dependencies

Functional dependencies having numerous applications in the field of database management system. Here are some applications listed below:

1. Data Normalization

- Data normalization is the process of organizing data in a database in order to minimize redundancy and increase data integrity.
- Functional dependencies play an important part in data normalization. With the help of functional dependencies we are able to identify the primary key, candidate key in a table which in turns helps in normalization.

2. Query Optimization

- With the help of functional dependencies we are able to decide the connectivity between the tables and the necessary attributes need to be projected to retrieve the required data from the tables. This helps in query optimization and improves performance.

3. Consistency of Data

- Functional dependencies ensures the consistency of the data by removing any redundancies or inconsistencies that may exist in the data.
- Functional dependency ensures that the changes made in one attribute does not affect inconsistency in another set of attributes thus it maintains the consistency of the data in database.

4. Data Quality Improvement

- Functional dependencies ensure that the data in the database to be accurate, complete and updated.
- This helps to improve the overall quality of the data, as well as it eliminates errors and inaccuracies that might occur during data analysis and decision making, thus functional dependency helps in improving the quality of data in database.

3.4. Normal forms (1NF, 2NF, 3NF, BCNF)

- ➢ The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form.
- ➢ Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Database Normal Forms

1NF (First Normal Form)

2NF (Second Normal Form)

3NF (Third Normal Form)

BCNF (Boyce-Codd Normal Form)

4NF (Fourth Normal Form)

5NF (Fifth Normal Form)

6NF (Sixth Normal Form)

- ➢ The Theory of Data Normalization in MySQL server is still being developed further. For example, there are discussions even on 6th Normal Form.
- ➢ However, in most practical applications, normalization achieves its best in 3rd Normal Form.

Summary:

- **1NF**: Atomicity (no repeating groups).
- **2NF**: No partial dependencies (all non-key attributes depend on the whole key).
- **3NF**: No transitive dependencies (non-key attributes depend only on the key).
- **BCNF**: Every determinant is a superkey.
- **4NF**: No multi-valued dependencies.
- **5NF**: Decomposition without loss of information.

1NF: Atomicity (no repeating groups).

emp_id	emp_name	emp_mobile	emp_skills
1	John Tick	9999957773	Python, JavaScript
2	Darth Trader	8888853337	HTML, CSS, JavaScript
3	Rony Shark	7777720008	Java, Linux, C++

Sol

emp_id	emp_name	emp_mobile
1	John Tick	9999957773
2	Darth Trader	8888853337
3	Rony Shark	7777720008

emp_id	skill
1	JavaScript
1	Python
2	CSS
2	HTML
2	JavaScript
3	C++
3	Java
3	Linux

emp_id	emp_name	emp_mobile	emp_skill
1	John Tick	9999957773	Python
1	John Tick	9999957773	JavaScript
2	Darth Trader	8888853337	HTML
2	Darth Trader	8888853337	CSS
2	Darth Trader	8888853337	JavaScript
3	Rony Shark	7777720008	Java
3	Rony Shark	7777720008	Linux
3	Rony Shark	7777720008	C++

Example 2: Perform normal form of the following student table.

StudentID	StudentName	Class	Subjects	Teachers	Address	ParentName
1	Aarav Shrestha	10	Maths, Science, English	Mr. Sharma, Miss Rai, Miss Karki	Kathmandu, Nepal	Ram Shrestha
2	Anjali Karki	10	Maths, Nepali, Computer	Mr. Sharma, Miss Shrestha, Miss Koirala	Pokhara, Nepal	Hari Karki
3	Binod Thapa	9	Science, Social Studies, English	Miss Rai, Mr. Ghimire, Miss Karki	Chitwan, Nepal	Krishna Thapa
4	Kriti Subedi	8	Nepali, Social Studies, English	Miss Shrestha, Mr. Ghimire, Miss Karki	Butwal, Nepal	Shiva Subedi
5	Prakash Bista	10	Maths, Science, English	Mr. Sharma, Miss Rai, Miss Karki	Biratnagar, Nepal	Mohan Bista
6	Sneha Khadka	9	Nepali, Computer, Maths	Miss Shrestha, Miss Koirala, Mr. Sharma	Dharan, Nepal	Ramesh Khadka

OR

StudentID	StudentName	Class	Subjects	Teachers	Address	ParentName
1	Aarav Shrestha	10	- Maths - Science - English	- Mr. Sharma - Miss Rai - Miss Karki	Kathmandu, Nepal	Ram Shrestha
2	Anjali Karki	10	- Maths - Nepali - Computer	- Mr. Sharma - Miss Shrestha - Miss Koirala	Pokhara, Nepal	Hari Karki
3	Binod Thapa	9	- Science - Social Studies - English	- Miss Rai - Mr. Ghimire - Miss Karki	Chitwan, Nepal	Krishna Thapa
4	Kriti Subedi	8	- Nepali - Social Studies - English	- Miss Shrestha - Mr. Ghimire - Miss Karki	Butwal, Nepal	Shiva Subedi
5	Prakash Bista	10	- Maths - Science - English	- Mr. Sharma - Miss Rai - Miss Karki	Biratnagar, Nepal	Mohan Bista
6	Sneha Khadka	9	- Nepali - Computer - Maths	- Miss Shrestha - Miss Koirala - Mr. Sharma	Dharan, Nepal	Ramesh Khadka

Solution:

1st Normal Form (1NF)

Rule: Remove multivalued attributes and ensure atomic data.

- Break down *Subjects, Teachers, and Address* into atomic values.

1NF Table:

StudentID	StudentName	Class	Subject	Teacher	City	Country	ParentName
1	Aarav Shrestha	10	Maths	Mr. Sharma	Kathmandu	Nepal	Ram Shrestha
1	Aarav Shrestha	10	Science	Miss Rai	Kathmandu	Nepal	Ram Shrestha
1	Aarav Shrestha	10	English	Miss Karki	Kathmandu	Nepal	Ram Shrestha
2	Anjali Karki	10	Maths	Mr. Sharma	Pokhara	Nepal	Hari Karki
2	Anjali Karki	10	Nepali	Miss Shrestha	Pokhara	Nepal	Hari Karki
2	Anjali Karki	10	Computer	Miss Koirala	Pokhara	Nepal	Hari Karki
3	Binod Thapa	9	Science	Miss Rai	Chitwan	Nepal	Krishna Thapa
3	Binod Thapa	9	Social Studies	Mr. Ghimire	Chitwan	Nepal	Krishna Thapa
3	Binod Thapa	9	English	Miss Karki	Chitwan	Nepal	Krishna Thapa
4	Kriti Subedi	8	Nepali	Miss Shrestha	Butwal	Nepal	Shiva Subedi
4	Kriti Subedi	8	Social Studies	Mr. Ghimire	Butwal	Nepal	Shiva Subedi
4	Kriti Subedi	8	English	Miss Karki	Butwal	Nepal	Shiva Subedi
5	Prakash Bista	10	Maths	Mr. Sharma	Biratnagar	Nepal	Mohan Bista
5	Prakash Bista	10	Science	Miss Rai	Biratnagar	Nepal	Mohan Bista
5	Prakash Bista	10	English	Miss Karki	Biratnagar	Nepal	Mohan Bista
6	Sneha Khadka	9	Nepali	Miss Shrestha	Dharan	Nepal	Ramesh Khadka
6	Sneha Khadka	9	Computer	Miss Koirala	Dharan	Nepal	Ramesh Khadka
6	Sneha Khadka	9	Maths	Mr. Sharma	Dharan	Nepal	Ramesh Khadka

2nd Normal Form (2NF)

Rule: Eliminate partial dependency. Each non-key attribute should depend on the whole primary key.

- Identify functional dependencies and decompose the table.
- Create separate tables for *Subjects* and *Teachers* relationships.

Decomposed Tables in 2NF:

Students Table:

StudentID	StudentName	Class	City	Country	ParentName
1	Aarav Shrestha	10	Kathmandu	Nepal	Ram Shrestha
2	Anjali Karki	10	Pokhara	Nepal	Hari Karki
3	Binod Thapa	9	Chitwan	Nepal	Krishna Thapa

Subjects Table:

StudentID	Subject
1	Maths
1	Science
1	English
2	Maths
2	Nepali
2	Computer

Teachers Table:

Subject	Teacher
Maths	Mr. Sharma
Science	Miss Rai
English	Miss Karki
Nepali	Miss Shrestha
Computer	Miss Koirala

3rd Normal Form (3NF)

Rule: Eliminate transitive dependencies. Non-prime attributes should not depend on other non-prime attributes.

- Separate location details (City, Country).

Decomposed Tables in 3NF:

Students Table:

StudentID	StudentName	Class	CityID	ParentName
1	Aarav Shrestha	10	101	Ram Shrestha
2	Anjali Karki	10	102	Hari Karki
3	Binod Thapa	9	103	Krishna Thapa

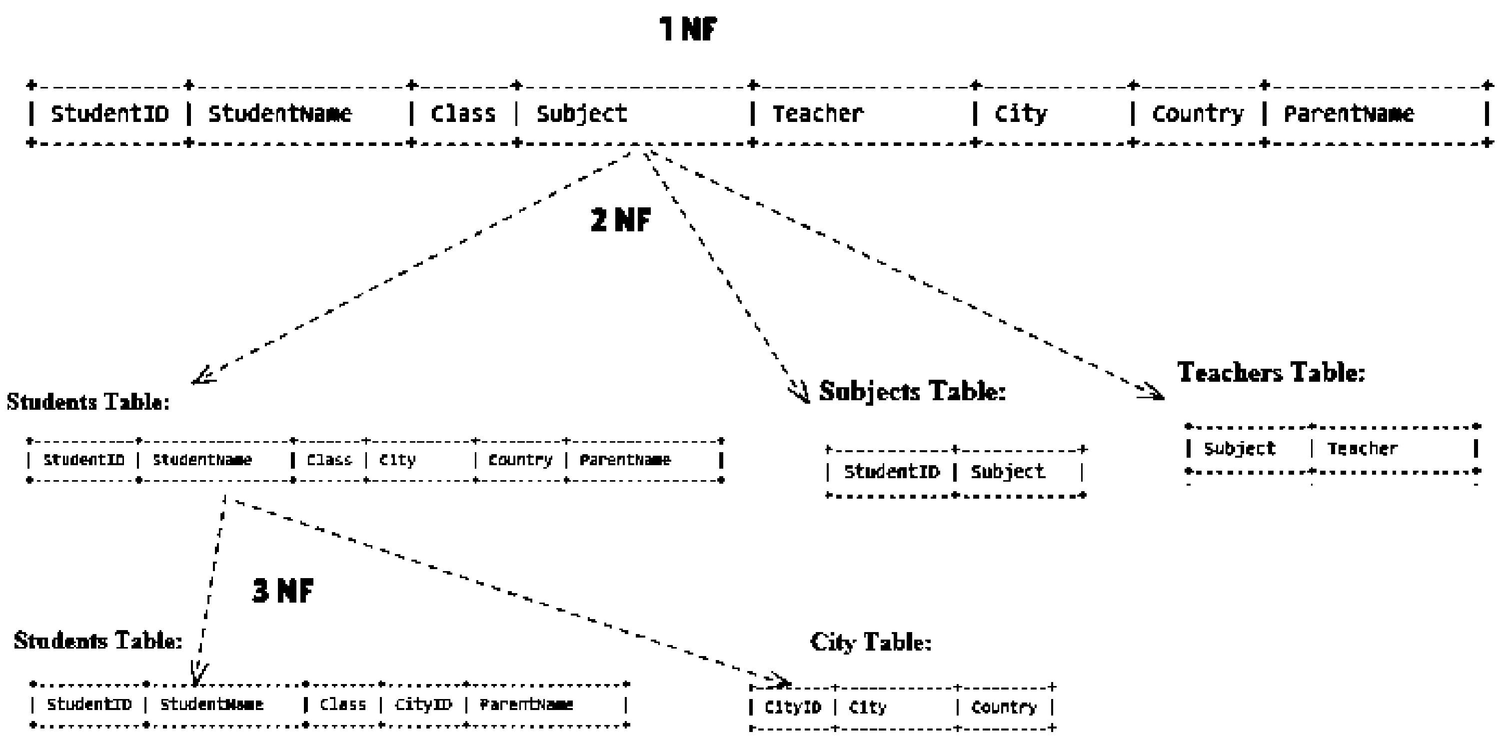
City Table:

CityID	City	Country
101	Kathmandu	Nepal
102	Pokhara	Nepal
103	Chitwan	Nepal

BCNF (Boyce-Codd Normal Form)

Rule: Ensure every determinant is a candidate key.

- In this case, the tables in 3NF already satisfy BCNF as there are no remaining dependencies where a determinant is not a candidate key.



Reasons for Normalization:

1. **1NF**: To eliminate repeating groups and ensure atomic data.
2. **2NF**: To remove partial dependencies by creating smaller, more focused tables.
3. **3NF**: To eliminate transitive dependencies and ensure non-prime attributes are fully functionally dependent on the primary key.
4. **BCNF**: To handle edge cases of dependency anomalies by ensuring all determinants are candidate keys.

Example: Perform normal form of the following student table , Normalize upto BCNF.

Given Functional dependencies

studentID -> studentName

courseID -> courseName

courseID -> credit

studentID	studentName	courseID	courseName	credit	contact
101	Ram	IT220	DBMS	3	9841,5573
102	Sita	IT220	DBMS	3	9850
103	John	IT218	DSA with Java	3	9871,2365
104	Jenny	ECO201	Micro Economics	3	9985,4395

SOLUTION:

1st Normal Form (1NF)

1NF Rule: Ensure that all attributes have atomic values (no repeating groups).

In the original table, the **contact** attribute contains multiple phone numbers, so we need to split that into atomic values.

studentID	studentName	courseID	courseName	credit	contact
101	Ram	IT220	DBMS	3	9841
101	Ram	IT220	DBMS	3	5573
102	Sita	IT220	DBMS	3	9850
103	John	IT218	DSA with Java	3	9871
103	John	IT218	DSA with Java	3	2365
104	Jenny	ECO201	Micro Economics	3	9985
104	Jenny	ECO201	Micro Economics	3	4395

2nd Normal Form (2NF)

2NF Rule: Eliminate partial dependency. A non-key attribute must depend on the whole primary key.

- In the 1NF table, the primary key is a composite key: (**studentID, courseID, contact**).
- studentID -> studentName, courseID -> courseName, and courseID -> credit violate the 2NF rule because studentName depends only on studentID, and courseName and credit depend only on courseID.

To remove partial dependency, we create separate tables for **Student**, **Course**, and a **StudentCourse** relationship table.

Decomposed Tables in 2NF:

Student Table:

studentID	studentName
101	Ram
102	Sita
103	John
104	Jenny

Course Table:

courseID	courseName	credit
IT220	DBMS	3
IT218	DSA with Java	3
ECO201	Micro Economics	3

StudentCourse Table

studentID	courseID	contact
101	IT220	9841
101	IT220	5573
102	IT220	9850
103	IT218	9871
103	IT218	2365
104	ECO201	9985
104	ECO201	4395

3rd Normal Form (3NF)

3NF Rule: Eliminate transitive dependency. A non-prime attribute should not depend on another non-prime attribute.

- In the 2NF tables, there is no transitive dependency, as studentName, courseName, and credit only depend on their respective keys (studentID and courseID).

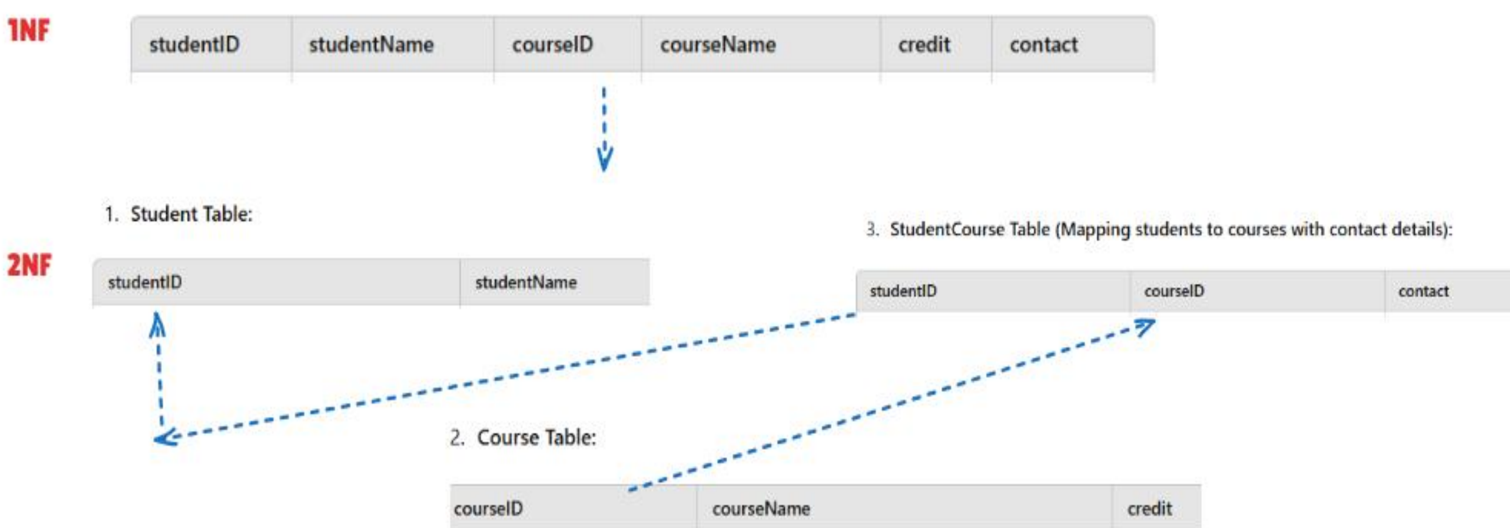
Since there are no transitive dependencies, the tables are already in **3NF**.

Boyce-Codd Normal Form (BCNF)

BCNF Rule: Every determinant must be a candidate key.

- In the Student, Course, and StudentCourse tables, all determinants (attributes that determine other attributes) are either primary keys or part of the primary key.

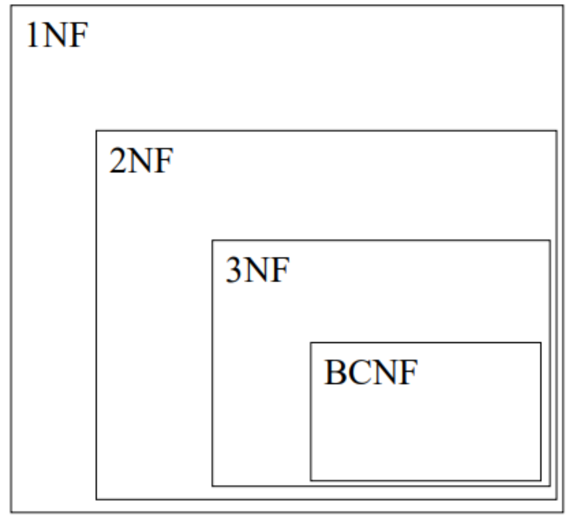
Thus, the tables satisfy **BCNF** as well.



a relation in BCNF,

is also in 3NF a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF



Q/A

Fill in the Blanks (20 Questions)

Multiple Choice Questions (MCQ) (20 Questions)

Short Questions (20 Questions)

Comprehensive Questions (20 Questions)

ANSWER of (MCQ)

ANSWER of Short Questions