Program : **B.Tech**

Subject Name: **Database Management System**

Subject Code: **IT-405**

Semester: **4th**

Unit IV

Normalisation of Database

Database Normalizations is a technique of organising the data in the database. Normalisation is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalisation is used for mainly two purposes,

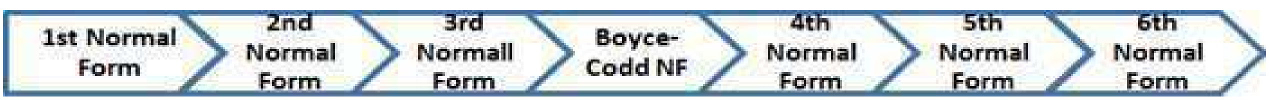
Eliminating redundant(useless) data.
perusalEnsuring data dependencies make sense, i.e. data is logically stored.

Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updating and Deletion Anomalies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

- **Updating Anomaly:** To update the address of a student who occurs twice or more than twice in a table, we will have to update the **Address** column in all the rows, else data will become inconsistent.
- **Insertion Anomaly:** Suppose for a new admission, we have a Student id(S_id), name and address of a student but if the student has not opted for any subjects yet then we must insert **NULL** there, leading to Insertion Anomaly.
- **Deletion Anomaly:** If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.
  Theory of Data Normalization in SQL is still being developed further. For example, there are discussions even on 6th Normal Form. However, in most practical applications normalisation achieves its best in 3rd Normal Form. The evolution of Normalization theories is illustrated below-



Functional Dependencies

A functional dependency is a relationship between two attributes. Typically, between the PK and other non-key attributes within the table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y.

X ————> Y

The left-hand side of the FD is called the determinant, and the right-hand side is the dependent.
Examples:

SIN ———-> Name, Address, Birthdate

SIN determines names and address and birthdays. Given SIN, we can determine any of the other attributes within the table.

Sin, Course ———> Date-Completed

Sin and Course determine date completed. This must also work for a composite PK.

ISBN ————> Title

ISBN determines the title.
Various Types of Functional Dependencies are –

- Single Valued Functional Dependency
- Fully Functional Dependency
- Partial Functional Dependency
- Transitive Functional Dependency
- Trivial Functional Dependency
- Non-Trivial Functional Dependency
    - Complete Non-Trivial Functional Dependency
    - Semi Non-Trivial Functional Dependency

Single Valued Functional Dependency –

The database is a collection of related information in which one information depends on another information. The information is either single-valued or multi-valued. For example, the name of the person or his / her date of birth is single-valued facts. However, the qualification of a person is a multivalued fact.

A simple example of single value functional dependency is when A is the primary key of an entity (e.g. SID), and B is some single-valued attribute of the entity (e.g. Sname). Then, A → B must always hold.

CID	SID	Sname
C1	S1	A
C1	S2	A
C2	S1	A
C3	S1	A

SID → Sname

S1	A
S1	A
S1	A

Sname → SID X

A	S1
A	S2

For every SID, there should be a unique name (X → Y)
Definition: Let R be the relational schema and X, Y be the set of attributes over R. t1, t2 be any tuples of R. X → Y exists in relation R only if t1.X = t2.X then t1.Y = t2.Y
If the condition fails – then the dependency is not there.

Fully Functional Dependency

In a relation R, an attribute Q is said to be fully functional dependent on attribute P, if it is functionally dependent on P and not functionally dependent on any proper subset of P. The dependency P → Q is left reduced, there is no extraneous attributes in the left-hand side of the dependency.

If AD → C, is a fully functional dependency, then we cannot remove A or D., I.e. C is fully functionally dependent on AD. If we can remove A or D, then it is not fully functional dependency.
Another Example, Consider the following Company Relational Schema,

EMPLOYEE

ENAME	SSN(P.K)	BDATE	ADDRESS	NUMBER

DEPARTMENT

DNAME	DNUMBER (P.K)	DMGRSSN (F.K)

DEPT_LOCATIONS

DNUMBER (P.k)	DLOCATION (P.K)

PROJECT

PNAME	PNUMBER	LOCATION	DNUM

WORKS_ON

SSN (P.K)	PNUMBER (P.K)	HOURS

{SSN, PNUMBER} → HOURS is a full FD since neither SSN → HOURS
nor NUMBER → HOURS hold

{SSN, PNUMBER} → ENAME is not a full FD (it is called a partial dependency) since SSN → ENAME also holds.

Partial Functional Dependency –

A Functional Dependency in which one or more non-key attributes are functionally depending on the part of the primary key is called partial functional dependency. or
where the determinant consists of critical attributes, but not the entire primary key, and the determined consist of non-key attributes.
For example, consider a Relation R (A, B, C, D, E) having
FD: AB → CDE where PK is AB.

Then, {A → C; A → D; A → E; B → C; B → D; B → E} all are Partial Dependencies.

Transitive Dependency –

Given a relation R (A, B, C) then dependency like A–>B, B–>C is a transitive dependency, since A–>C is implied.

In the above Figure
SSN --> DMGRSSN is a transitive FD
{since SSN --> DNUMBER and DNUMBER --> DMGRSSN hold}

SSN --> NAME is non-transitive FD since there is no set of attributes X
where SSN --> X and X --> ENAME.

Trivial Functional Dependency –

Some functional dependencies are said to be trivial because they are satisfied by all relations. Functional dependency of form A–>B is trivial if B subset= A. or

A trivial Functional Dependency is the one where RHS is a subset of LHS.
Example, A-->A is satisfied by all relations involving attribute A.
SSN-->SSN
PNUMBER-->PNUMBER
SSN PNUMBER -->PNUMBER
SSN PNUMBER --> SSN PNUMBER

Non-Trivial Functional Dependency –

Non-Trivial Functional Dependency can be categorised into –

- Complete Non-Trivial Functional Dependency
- Semi Non-Trivial Functional Dependency

Complete Non-Trivial Functional Dependency –
A Functional Dependency is entirely non-trivial if none of the RHS attributes is part of the LHS attributes.

Example, SSN --> Ename,
PNUMBER --> PNAME
PNUMBER--> BDATE X

Semi Non-Trivial Functional Dependencies – A Functional Dependency is semi non-trivial if at least one of the RHS attributes are not part of the LHS attributes.

{TRIVIAL + NONTRIVIAL}
Question 1:

A	B	C
1	1	1
1	2	1
2	1	2
2	2	3

Identify Non-Trivial Functional Dependency?
Solution:

S.NO	Dependencies	Non-Trivial FD?
1	A→B	×
2	A→C	×
3	A→BC	×
4	B→A	×
5	B→C	×
6	B→AC	×
7	C→A	√
8	C→B	×
9	C→AB	×
10	AB→C	√
11	BC→A	√
12	AC→B	×

A→B is not a non-trivial FD because, for 2, it has two outputs. i.e 2→2 and 2→3.
for AB→C, 11→1, 12→1, 21→2, 22→3, so Non-trivial.

Question 2: R (A B C D) AB {Candidate Key} A→C B→D. Where is the redundancy existing?
Solution: (A C) and (B D) is suffering from redundancy.
Question 3: Consider a relation with schema R (A, B, C, D) and FDs {AB -> C, C -> D, D -> A}. a. What are some of the nontrivial FDs that can be inferred from the given FDs?

Some examples:

C -> ACD
D -> AD
AB -> ABCD
AC -> ACD
BC -> ABCD
BD -> ABCD
CD -> ACD
ABC -> ABCD
ABD -> ABCD
BCD -> ABCD

Inference Rules for Functional Dependencies –
Armstrong's Inference Rules –
Let A, B and C and D be arbitrary subsets of the set of attributes of the giver relation R, and let AB be the union of A and B. Then, ⇒→
Primary
Reflexivity:
If B is subset of A, then A → B

Augmentation:
If A → B, then AC → BC

Transitivity:
If A → B and B → C, then A → C.
Projectivity or Decomposition Rule:
If A → BC, Then A → B and A → C
Proof:
Step 1: A → BC (GIVEN)
Step 2: BC → B (Using Rule 1, since B ⊆ BC)
Step 3: A → B (Using Rule 3, on step 1 and step 2)

Secondary Rule

Union or Additive Rule:
If A→B, and A→C Then A→BC.
Proof:
Step 1: A → B (GIVEN)
Step 2: A → C (given)
Step 3: A → AB (using Rule 2 on step 1, since AA=A)
Step 4: AB → BC (using rule 2 on step 2)
Step 5: A → BC (using rule 3 on step 3 and step 4)

Pseudo Transitive Rule:
If A → B, DB → C, then DA → C
Proof:
Step 1: A → B (Given)
Step 2: DB → C (Given)
Step 3: DA → DB (Rule 2 on step 1)
Step 4: DA → C (Rule 3 on step 3 and step 2)'

These are not commutative as well as associative.
i.e. if X → Y then
Y → X x (not possible)

Composition Rule:
If A → B, and C → D, then AC → BD.

Self Determination Rule:
A → A is a self-determination rule.
Let S be the set of functional dependencies that are specified on relation schema R. Numerous other dependencies can be inferred or deduced from the functional dependencies in S.

Example:

Let S = {A → B, B → C}

A multivalued dependency occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A multivalued dependency prevents the normalization standard Fourth Normal Form (4NF).

Functional dependency vs. Multivalued dependency
To understand this, let's revisit what a functional dependency is.

Remember that if an attribute X uniquely determines an attribute Y, then Y is functionally dependent on X. This is written as X -> Y. For example, in the Students table below, the Student_Name determines the Major:

Students

Student_Name	Major
Ravi	Art History
Beth	Chemistry

This functional dependency can be written: Student_Name -> Major. Each Student_Name determines exactly one Major, and no more.

Now, perhaps we also want to track the sports these students take. We might think the easiest way to do this is just to add another column, Sport:

Students

Student_Name	Major	Sport
Ravi	Art History	Soccer
Ravi	Art History	Volleyball
Ravi	Art History	Tennis
Beth	Chemistry	Tennis
Beth	Chemistry	Soccer

The problem here is that both Ravi and Beth play multiple sports. We need to add a new row for every additional sport.

This table has introduced a multivalued dependency because the major and the sport are independent of one another, but both depend on the student.

Note that this is a very simple example and easily identifiable — but this could become a problem in an extensive, complex database.

A multivalued dependency is written X ->-> Y. In this case:

Student_Name ->-> Major
Student_Name ->-> Sport

This is read as "Student_Name multidetermined Major" and "Student Name multidetermined Sport."

A multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

Multivalued dependency and normalization
A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NK) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, we can break this into two tables.

The table below now has a functional dependency of Student_Name -> Major, and no multi dependencies:

Students & Majors

Student_Name	Major
Ravi	Art History
Ravi	Art History
Ravi	Art History
Beth	Chemistry
Beth	Chemistry

While this table also has a single functional dependency of Student_Name -> Sport:

Students & Sports

Student_Name	Sport
Ravi	Soccer
Ravi	Volleyball
Ravi	Tennis
Beth	Tennis
Beth	Soccer

Normalisation is often addressed by simplifying complex tables so that they contain information related to a single idea or theme, rather than trying to make a single table contain too much disparate information.

Numerical on Functional Dependency: -
1. Let R= (A, B, C, D, E, F) be a relation scheme with the following dependencies: C->F, E->A, EC->D, A->B. Which of the following is a key for R?
(a) CD (b) EC (c) AE (d) AC

Ans: option (b)
Explanation:
Find the closure set of all the options given. If any closure covers all the attributes of the relation R then that is the key.

2. Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A–>B, BC–>D, E–>C, D–>A}. What are the candidate keys of R?
(a) AE, BE
(b) AE, BE, DE
(c) AEH, BEH, BCH
(d) AEH, BEH, DEH

Ans: option (d)
Explanation:
As explained in question 1, if any closure includes all attributes of a table then it becomes the candidate key.
Closure of AEH = AEHB {A->B}
= AEHBC {E->C}
= AEHBCD {BC->D}
GATE-2005(IT)

5. In a schema with attributes A, B, C, D and E, following set of functional dependencies are given:
A->B
A->C
CD->E
B->D
E->A
Which of the following functional dependencies is NOT implied by the above set?
(a) CD->AC (b) BD->CD (c) BC->CD (d) AC->BC

Ans: option (b)
Explanation:
For every option given, find the closure set of the left side of each FD. If the closure set of left side contains the right side of the FD, then the FD is implied by the given set.
Option (a): Closure set of CDs = CDEAB. Therefore CD->AC can be derived from the given set of FDs.
Option (c): Closure set of BCs = BCDEA. Therefore BC->CD can be derived from the given set of FDs.
Option (d): Closure set of AC = ACBDE. Therefore AC->BC can be derived from the given set of FDs.
Option (b): Closure set of BDs = BD. Therefore BD->CD cannot be derived from the given set of FDs.

Normalisation
First Normal Form
First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, C++
Web	HTML, PHP,ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Programming

Course	Content
Programming	JAVA
Programming	C++
Web	HTML
Web	PHP

Web	ASP

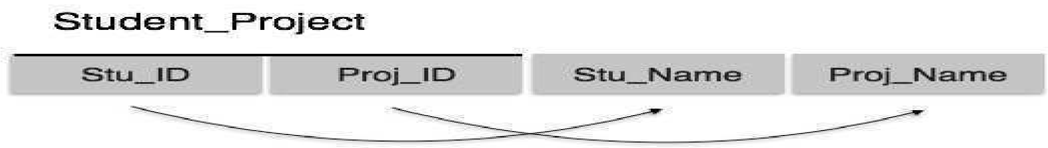
Second Normal Form

Before we learn about the second normal form, we need to understand the following –

Prime attribute – An attribute, which is a part of the prime-key, is known as a prime attribute.

Non-prime attribute – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow the second standard form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attributes individually. However, we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Student

Stu_ID	Stu_Name	Proj_ID

Project

Proj_ID	Proj_Name

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal form, and the following must satisfy –

No non-prime attribute is transitively dependent on crucial prime attribute.

For any non-trivial functional dependency, X → A, then either –

X is a superkey or,

A is a prime attribute.

STUDENT_DETAILS

Stu_ID	Stu_Name	City	Zip

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency.

To bring this relation into third standard form, we break the relation into two relations as follows –.

Student_Details

Stu_ID	Stu_Name	Zip

Zip Codes

Zip	City

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that for any non-trivial functional dependency, X → A, X must be a super-key.
In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,
Stu_ID → Stu_Name, Zip
and
Zip → City
Which confirms that both the relations are in BCNF.

Fourth Normal Form (4NF)

When attributes in a relation have a multi-valued dependency, further Normalization to 4NF and 5NF are required. Let us first find out what multi-valued dependency is.
A multi-valued dependency is a typical kind of dependency in which every attribute within a relation depends upon the other, yet none of them is a unique primary key.
We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organisation. The following are the assumptions:
A vendor can supply many items.
A project uses many items.
A vendor supplies many projects.
Many vendors may supply an item.
A multi-valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having a unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P3

The table can be expressed as the two 4NF relations given as following. The fact that vendors can supply certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor-Supply

Vendor Code	Item Code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

Vendor-Project

Vendor Code	Project No.
V1	P1
V1	P3
V2	P1
V3	P2

Fifth Normal Form (5NF)

These relations still have a problem. While defining the 4NF, we mentioned that all the attributes depend upon each other. While creating the two tables in the 4NF, although we have preserved the dependencies between Vendor Code and Item code in the first table and Vendor Code and Item code in the second table, we have lost the relationship between Item Code and Project No. If there were a primary key, then this loss of dependency would not have occurred. To revive this relationship, we must add a new table like the following. Please note that during the entire process of normalisation, this is the only step where a new table is created by joining two attributes, rather than splitting them into separate tables.

Project No.	Item Code
P1	11
P1	12
P2	11
P3	11
P3	13