

UNIT-IV

Syllabus:

Schema Refinement (Normalization) : Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF, 2NF and 3 NF), concept of surrogate key, Boyce-codd normal form(BCNF), Lossless join and dependency preserving decomposition, Fourth normal form(4NF).

Introduction

When designing a database, you have to make decisions regarding how best to take some system in the real world and model it in a database. This consists of deciding which tables to create, what columns they will contain, as well as the relationships between the tables. While it would be nice if this process was totally intuitive and obvious, or even better automated, this is simply not the case. A well-designed database takes time and effort to conceive, build and refine.

The benefits of a database that has been designed according to the relational model are numerous. Some of them are:

- Data entry, updates and deletions will be efficient.
- Data retrieval, summarization and reporting will also be efficient.
- Since the database follows a well-formulated model, it behaves predictably.
- Since much of the information is stored in the database rather than in the application, the database is somewhat self-documenting.
- Changes to the database schema are easy to make.

Purpose of Normalization:-

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Concept of Functional Dependency:-

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign ($\rightarrow$) that is, $X \rightarrow Y$, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Normal Forms based on Functional Dependencies(1NF,2NF and 3 NF):-

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.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Each attribute must contain only a single value from its pre-defined domain.

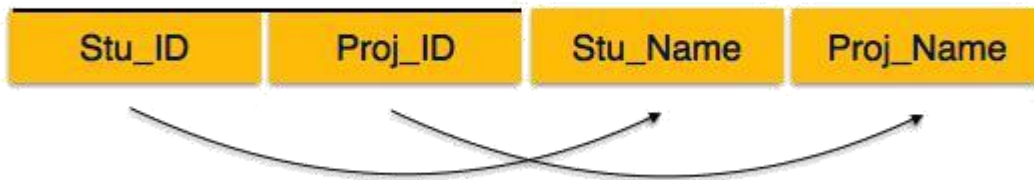
Second Normal Form

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

- **Prime attribute** – An attribute, which is a part of the candidate-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 second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X, for which $Y \rightarrow A$ also holds true.

Student_Project



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 attribute individually. But 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



Project



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 prime key attribute.
- For any non-trivial functional dependency, $X \rightarrow A$, then either –
 - X is a superkey or,
 - A is prime attribute.

Student_Detail



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, $\text{Stu_ID} \rightarrow \text{City}$ and $\text{City} \rightarrow \text{Zip}$, so there exists **transitive dependency**.

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

Student_Detail		
Stu_ID	Stu_Name	Zip
ZipCodes		
Zip	City	

Concept of a surrogate key in DBMS:-

A surrogate key is any column or set of columns that can be declared as the **primary key** instead of a "real" or natural key. Sometimes there can be several natural keys that could be declared as the **primary key**, and these are all called candidate keys. So a surrogate is a **candidate key**.

A **candidate key** is a column, or set of columns, in a table that can uniquely identify any **database** record without referring to any other data. Each table may have one or more **candidate keys**, but one **candidate key** is unique, and it is called the **primary key**.

Boyce-Codd normal form(BCNF):-

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency, $X \rightarrow A$, X must be a super-key.

Student_Detail		
Stu_ID	Stu_Name	Zip
ZipCodes		
Zip	City	

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,

$\text{Stu_ID} \rightarrow \text{Stu_Name}, \text{Zip}$

and

$\text{Zip} \rightarrow \text{City}$

Which confirms that both the relations are in BCNF.

Lossless join and dependency preserving decomposition:-

What is decomposition?

- Decomposition is the process of breaking down in parts or elements.
- It replaces a relation with a collection of smaller relations.
- It breaks the table into multiple tables in a database.
- It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.
- If there is no proper decomposition of the relation, then it may lead to problems like loss of information.

Properties of Decomposition

Following are the properties of Decomposition,

1. Lossless Decomposition
2. Dependency Preservation
3. Lack of Data Redundancy.

1. Lossless Decomposition

- Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed.
- It gives a guarantee that the join will result in the same relation as it was decomposed.

Example:

Let's take 'E' is the Relational Schema, With instance 'e'; is decomposed into: E1, E2, E3, ... En; With instance: e1, e2, e3, ... en, If $e1 \bowtie e2 \bowtie e3 \dots \bowtie en$, then it is called as '**Lossless Join Decomposition**'.

- In the above example, it means that, if natural joins of all the decomposition give the original relation, then it is said to be lossless join decomposition.

Example: <Employee_Department> Table

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing

E005	STU	32	Bangalore	25000	D005	Human Resource
------	-----	----	-----------	-------	------	----------------

- Decompose the above relation into two relations to check whether a decomposition is lossless or lossy.
- Now, we have decomposed the relation that is Employee and Department.

Relation 1 : <Employee> Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

- Employee Schema contains (Eid, Ename, Age, City, Salary).

Relation 2 : <Department> Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- Department Schema contains (Deptid, Eid, DeptName).
- o, the above decomposition is a Lossless Join Decomposition, because the two relations contains one common field that is 'Eid' and therefore join is possible.
- Now apply natural join on the decomposed relations.

Employee $\bowtie$ Department

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing

E005	STU	32	Bangalore	25000	D005	Human Resource
------	-----	----	-----------	-------	------	----------------

Hence, the decomposition is Lossless Join Decomposition.

- If the <Employee> table contains (Eid, Ename, Age, City, Salary) and <Department> table contains (Deptid and DeptName), then it is not possible to join the two tables or relations, because there is no common column between them. And it becomes **Lossy Join Decomposition**.

Dependency Preservation

Dependency is an important constraint on the database.

- Every dependency must be satisfied by at least one decomposed table.
- If $\{A \rightarrow B\}$ holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
 - This decomposition property can only be done by maintaining the functional dependency.
 - In this property, it allows to check the updates without computing the natural join of the database structure.

Fourth Normal Form(4NF):-

Fourth Normal Form comes into picture when **Multi-valued Dependency** occur in any relation. In this tutorial we will learn about Multi-valued Dependency, how to remove it and how to make any table satisfy the fourth normal form.

Rules for 4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation $R(A,B,C)$, if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Example

Below we have a college enrolment table with columns **s_id**, **course** and **hobby**.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

As you can see in the table above, student with **s id 1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

You must be thinking what problem this can lead to, right?

Well the two records for student with **s id 1**, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

And, in the table above, there is no relationship between the columns **course** and **hobby**. They are independent of each other.

So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

How to satisfy 4th Normal Form?

To make the above relation satisfy the 4th normal form, we can decompose the table into 2 tables.

CourseOpted Table

s_id	course
1	Science
1	Maths
2	C#
2	Php

And, **Hobbies Table**,

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form.

A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

If you design your database carefully, you can easily avoid these issues.