UNIT III

RELATIONAL DATABASE DESIGN

Basic Concepts

A database is a collection of logically related records. A relational database stores its data in 2-dimensional tables. A table is a two-dimensional structure made up of rows (tuples, records) and columns (attributes, fields). A relational database organizes data in tables (or relations). A table is made up of rows and columns. A row is also called a record (or tuple). A column is also called a field (or attribute). A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data. A language called SQL (Structured Query Language) was developed to work with relational databases.

Database Design Objective

A well-designed database shall:

- Eliminate Data Redundancy: the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- Ensure Data Integrity and Accuracy

Concepts:

- **Domains and attributes :** ***(refer ER data model)***
- **Keys:** ***(refer ER data model)***
- **Tuples:** ***(refer ER data model)***
- **Integrity rules:** ***(refer ER data model)***
- **Semantics**
- **Anomalies**
- **Functional dependency, Decomposition ,Normalization**

Semantics

Semantics specifies how to interpret the attribute values stored in a tuple of the relation-in other words, how the attribute values in a tuple relate to one another

REDUNDANT INFORMATION IN TUPLE (ANOMALIES)

- **ANOMALIES**

Anomalies are problems caused by bad database design. The problems arise from relations that are generated directly from user views are called anomalies.

Types:

1. *Insertion Anomaly*
2. *Updation Anomaly*
3. *Deletion Anomaly*

To understand these anomalies let us take an example of a **Student** table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.

Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but **Insertion anomalies**.

Updation Anomaly

What if Mr. X leaves the college? Or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is **Updation anomaly**.

Deletion Anomaly

In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is **Deletion anomaly**.

Cause of Anomalies

Anomalies are primarily caused by:

• ***Data redundancy****: replication of the same field in multiple times, other than foreign keys*

• ***Functional dependencies***

Fixing Anomalies

Anomalies can be corrected by

- ***Decomposition***
- ***Normalization***

FUNCTIONAL DEPENDENCIES

Functional dependency is defined as the attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table. If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Closure set of functional dependency (Rules of Functional Dependencies)

The set of al functional dependencies implied by a given set of F of functional dependencies is called closure of **Denoted by F^{+}.**

Below given are the three most important rules for Functional Dependency are given below and they are known by ***Armstrong's axioms***:

- **Reflexive rule** –. If X is a set of attributes and Y is subset of X, then X holds a value of Y(x -> y).
- **Augmentation rule**: When x -> y holds, and z is attribute set, then zx ->zy also holds. That is adding attributes which do not change the basic dependencies.
- **Transitivity rule**: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y

Additional rules:

- **Union rule**: Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z. If X → Y and X → Z then X → YZ
- **Decomposition Rule**: Decomposition rule is also known as project rule. It is the reverse of union rule.This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.If X → YZ then X → Y and X → Z
- **Pseudo transitive Rule**: In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.If X → Y and YZ → W then XZ → W

Types of Functional Dependencies

- ***Trivial functional dependency***
- ***non-trivial functional dependency***
- ***Multivalued dependency***
- ***Transitive dependency***
- ***Fully and partial functional dependency***

Trivial Functional Dependency (Reflexive Rule)

A → B has trivial functional dependency if B is a subset of A.

The following dependencies are also trivial like: A → A, B → B

- **Non-trivial functional dependency**

A → B has a non-trivial functional dependency if B is not a subset of A.

When A intersection B is NULL, then A → B is called as complete non-trivial.

Multi-Valued Dependency

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

1. For a dependency A → 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.

For an 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.

Transitive Dependency (Transitivity Rule)

When an indirect relationship causes functional dependency it is called Transitive Dependency.

If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

To achieve 3NF, eliminate the Transitive Dependency

Fully-Functionally Dependency

An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

Let us see an example:

<ProjectCost>

ProjectID	**ProjectCost**
001	1000
002	5000

<EmployeeProject>

EmpID	**ProjectID**	**Days** (spent on the project)
E099	001	320
E056	002	190

The above relations states:

EmpID, ProjectID, ProjectCost -> Days

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the **{Days}** spent on the project by the employee.This summarizes and gives our fully functional dependency:

{EmpID, ProjectID} -> (Days)

Partial Functional Dependency

Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.The 2nd Normal Form (2NF) eliminates the Partial Dependency.

Let us see an example:

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

In the above table, we have partial dependency; let us see how:

The prime key attributes are **StudentID** and **ProjectNo.**

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID** that makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectID**, which that the relation Partial Dependent.

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

1. Lossless Decomposition (Good) and Lossy decomposition (Bad)
2. Dependency Preservation

Lossless Decomposition/ Good Decomposition

- If the information is not lost from the relation that is decomposed, ***then the decomposition will be lossless.***
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.

- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Lossy Decomposition/Bad Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved. If the information is lost from the relation that is decomposed, then the decomposition will be ***lossy decomposition.***

Let us see an example:

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation. Therefore, the above relation has lossy decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

NORMALIZATION

Database normalization is a database schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data. Normalization split a large table into smaller tables and define relationships between them to increases the clarity in organizing data.

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 the undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization divides the larger table into the smaller table and links them using relationship. The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)
- **Fifth normal form or project normal form(5NF or PJNF)**

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single (atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Rule 1: *Single Valued Attributes*

Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

Rule 2: *Attribute Domain should not change*

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

Rule 3: *Unique name for Attributes/Columns*

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.

If one or more columns have same name, then the DBMS system will be left confused.

Rule 4: *Order doesn't matters*

This rule says that the order in which you store the data in your table doesn't matter.

Example 1 – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Second Normal Form (2NF)

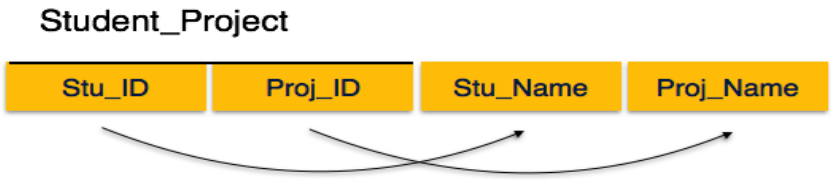
For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

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 (ref previous topic "functional dependency")

If we follow second normal 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 true.



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.

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

Third Normal Form (3NF)

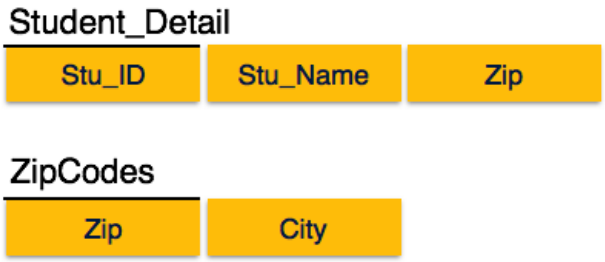
A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.
3. For any non-trivial functional dependency, X → A, then either −X is a superkey

Transitive Dependency (ref previous topic "functional dependency")

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 normal form, we break the relation into two relations as follows −



Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- And, for each functional dependency (X → Y), X should be a super Key.

For an Example

Below we have a college enrolment table with columns student_id, subject and professor.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

As you can see, we have also added some sample data to the table.

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

What do you think should be the **Primary Key**?

Well, in the table above student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the **2nd Normal Form** as there is no **Partial Dependency**.And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.But this table is not in **Boyce-Codd Normal Form**.To make this relation (table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.

Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2
and so on...	

And, **Professor Table**

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfy Boyce-Codd Normal Form. In the next tutorial we will learn about the **Fourth Normal Form**.

Fourth Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

For an 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 satify 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.

Fifth Normal Form / Projected Normal Form (5NF):

A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join. A relation is in Fifth Normal Form (5NF), if it is in 4NF, and won't have lossless decomposition into smaller tables.

Properties – A relation R is in 5NF if and only if it satisfies following conditions:

1. R should be already in 4NF.
2. It cannot be further non loss decomposed (join dependency)