RDM Normalization. Data Anomalies. Functional Dependency. Normal Forms.

Introduction.

This topic is a concise overview of normalization: what it is, why it's done, its pros and cons, its benefits and costs, the normalization process, and 0NF to 3NF transformation.

Definition

Normalization is a logical data base design method. **Normalization is a process of systematically breaking a complex table into simpler ones**. It is built around the concept of normal forms.

Why do we have to normalize?

Normalization is necessary if you do not do it then the overall integrity of the data stored in the database will eventually degrade.

Anomalies are caused when there is too much redundancy in the database's information or anomalies can often be caused when the tables that make up the database suffer from poor construction.

Purposes of Normalization

In the design of a data model, normalization is the process of adjusting table and relations to:

- eliminate certain types of data (redundancy/replication) to improve consistency,
- produce a clearer and readable data model.
- provide maximum flexibility to meet future information needs by keeping tables corresponding to object types in their simplified forms.
- avoid anomalies

Normalization: Pros and Cons

Pros

- Reduce data redundancy & space required
- Enhance data consistency

- Enforce data integrity
- Reduce update cost
- Provide maximum flexibility in responding special queries
- Allow the use of parallelism,
- Can reduce the total number of rows per block.
- Improve Software Design
    - Maintainability
    - Reusability
    - Readability

Cons

- Many complex queries will be **slower** because joins have to be performed to retrieve relevant data from several normalized tables
- Programmers/users have to **understand the data model** to perform proper joins among several tables
- The formulation of multiple-level queries is a **non-trivial task**.

Functional Dependencies

Definition: A functional dependency (FD) is a **relationship between two attributes**, typically between PK and other non-key table attributes.

For any relation R, attribute Y is functionally dependent on attribute X, if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by this representation: X → Y. The left side of the above FD diagram (X) is called the **determinant**, and the right side (Y) is the **dependent**. Here are a few examples:

1) SIN (Social Insurance Number) determines Name, Address and Birthdate. Using SIN, we can determine any of the other table attributes.

SIN → Name, Address, Birthdate

2) This also work for a composite PK. SIN and Course determine the course date completed (DateCompleted).

SIN, Course → DateCompleted

3) ISBN determines Book Title.

ISBN → Title

4) What of dependencies have between the attributes in Table R?

Table R

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	c2	d2	e1
a3	b2	c1	d1	e1
a4	b2	c2	d2	e1
a5	b3	c3	d1	e1

Looking at actual data can help understand which attributes are dependent and which are determinants.

Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

- A→B, A→C, A→D, A→E
- A→BC or any other subset of ABCDE
- This can be summarized a A→BCDE
- From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

- A→E, B→E, C→E, D→E
- However, we cannot generally summarize the above with ABCD→E

Combinations of BC are unique, therefore BC→ADE

Combinations of BD are unique, therefore BD→ACE

Data Anomalies

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.

Formally, the following kind of Data Anomalies can arise:

Update Anomalies - inconsistency may occur because of the existence of data redundancy, have to update all relevant tuples instead of just one.

Insertion Anomalies - happen when inserting vital data into the database is not possible because other data is not already there, E.g. A library database that cannot store the details of a new member until that member has taken out a book.

- What to do when inserting tuple without the extra data?
- Use null?
- But what if it's part of the primary key? (It typically will be)
- What about subsequently added data? Should it replace the null, or just add a new row?

Deletion Anomalies - happen when the deletion of unwanted information causes desired information to be deleted as well.

- Replace with null because it's the last tuple for this entity?
- Do we lose information when we delete a tuple?
- Eg. delete the last student from a course.
- E.g. Deleting a library member can remove all details of the particular book from the database such as the author, book title etc.

Wasted storage - redundant information has to be stored multiple times.

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

Simple Example of Data Anomalies.

People(ssn, name, addr, hobby)

ssn	name	addr	hobby
555	Homer Simpson	12 Oak Street, Springfield	Drinking
555	Homer Simpson	12 Oak Street, Springfield	TV
555	Homer Simpson	12 Oak Street, Springfield	Eating
666	Bart Simpson	12 Oak Street, Springfield	Pranks
489	Lisa Simpson	12 Oak Street, Springfield	Reading
489	Lisa Simpson	12 Oak Street, Springfield	Politics
323	Krusty Clown	45 First Street, Springfield	Games

Example Anomalies

- **insert**: try to insert someone's, who has no hobbies: keys can't have null values! ➔ then you cannot add Mona Simpson?
- **delete**: delete someone's last hobby: keys can't have null values! ➔ then you need delete the Bart Simpson?
- **update**: to change someone's address requires find and changing multiple tuples ➔ really it's not a big problem.
- **mixing**: What if Mona Simpsons without any hobby were added with a null hobby and then we do an insert: should it replace the null hobby?

Solution: separate relations with relationship creation, must be **lossless** information. That means we can get all the information back by using a **JOIN**.

People(ssn, name, addr)
PeopleHobby(ssn, hobby)
Hobby(hobby)

Note not all redundancy removed, have foreign key (ssn and hobby). Can't be helped.

Normalization Process.

A flat file database contains all the data in one table. This is said to be un-normalized form (UNF or 0NF).

Applying the process of normalization to 3NF reduces many problems that can be found in an un-normalized database.

0NF to 1NF.

1.1. Make all columns atomic

Do you have any column which can be split into more than one column?

For example, address can be split into street, city, country and zip.

1.2. Have a primary key

Do you have a primary key?

Hint: A primary key is a column (or combination of columns) that uniquely identify all rows.

Add a primary key on existing column(s). If it's not possible to make existing column combinations as primary key

1.3. Move repeating group.

Do you have a group of two or more columns that are closely related and are all repeating the same attribute?

For example, a table that holds data on books might have columns such as book_id, book_name, author1, author2, author3 and so on which form a repeating group. In this case a new table (book_id, author) should be created; and author1, author2, author3 should be removed from initial table; and relationship 1:m between 2 table should be created.

1.4. Remove redundant column

Do you have a group of columns which on combining gives an existing column?

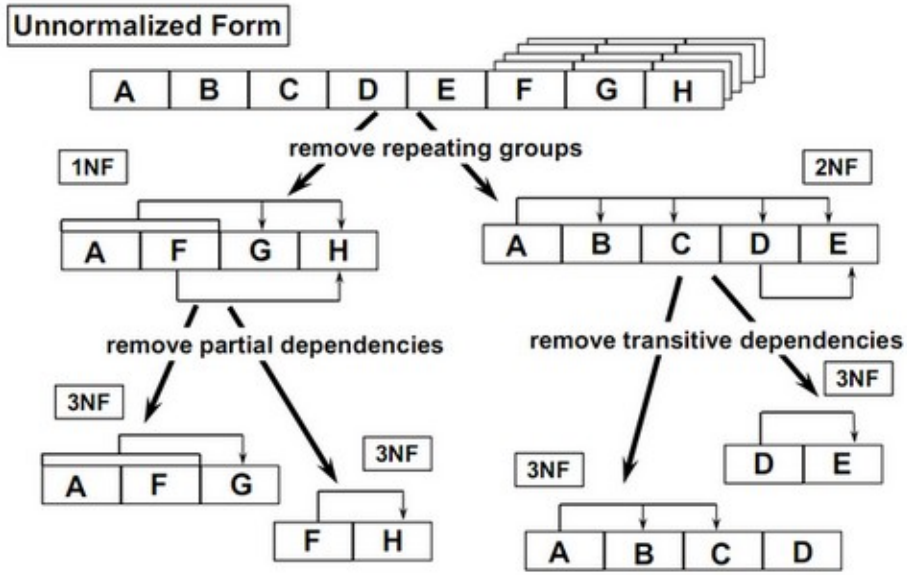
For example, if you have first_name, last_name and full_name then combining first_name and last_name gives full_name which is redundant. Or date-of-birth and age. In this case redundant columns full_name or age should be removed from initial tables.

1NF to 2NF.

No partial dependencies possible as the primary key has just one column.

If primary key has just only one column, then table is already in second normal form.

Normalization Process in Brief.

Normalization Process	Normalization steps	
	**0 Normal Form**	
	Do any attributes have multiple values for a single instance of an entity? Do any attributes have value from only one domain? (Are the any redundant or repeating attributes or groups?).	Yes: Delete redundant attributes and remove repeating attributes and groups along with a copy of the key to a new entity that describes this attributes. Usually you will need to add a relationship between the old and new entities. No: The data is in 1NF.
	1 Normal Form	
	Is the identifier consist of more than one attribute? If so, are any attribute values dependent on just part the identifier? (Are the any partial functional dependencies - FD?).	Yes: Remove the partial FD to a new entity along with a copy of the part of the key the entity is based on. Usually you will need to add a relationship between the old and new entities. No: The data is in 2NF.
	2 Normal Form	
	Do any attribute values depend on an attribute that is not the entity's identifier? (Are the any transitive functional dependencies - FD?).	Yes: Remove the derived attribute to a new entity in which their values are dependent on the identifier. Usually you will need to add a relationship between the old and new entities. No: The data model is in 3NF.
	3 Normal Form or BCNF	

Simple Examples of Normalization Steps.

0NF to 1NF.

- 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.
- We re-arrange the relation (table) as below, to convert it to First Normal Form.
- Each attribute must contain only a single value from its pre-defined domain. Need to add a primary key Course_ID.

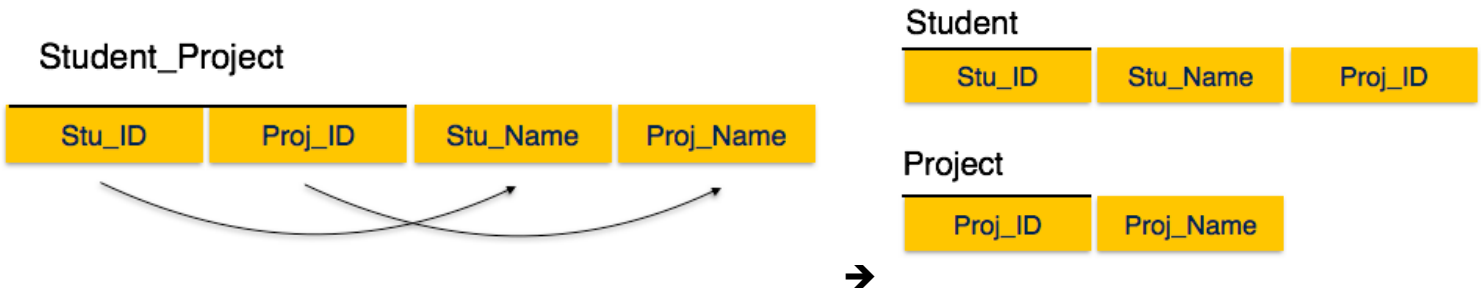
Course	Course_Content
Programming	Java, C++
Web	HTML, PHP, ASP

➔

Course_ID	Course_Name	Course_Content
1	Programming	Java
1	Programming	C++
2	Web	HTML
2	Web	PHP
2	Web	ASP

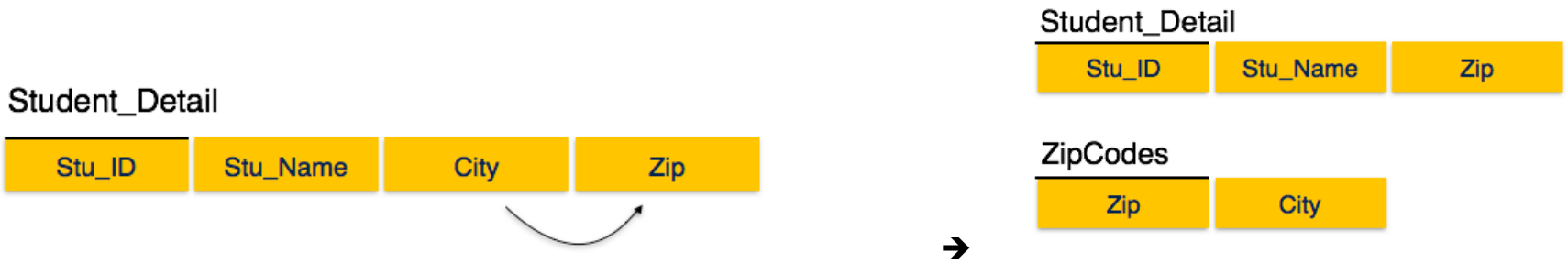
1NF to 2NF.

- 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 party 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 not exists a partial dependency.



2NF to 3NF or BCNF.

- For a relation to be in Third Normal Form, it must be in Second Normal form and no non-prime attribute is transitively dependent on prime key attribute over the not super-key.
- 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 follow.



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.
- Only in rare cases does a 3NF table not meet the requirements of BCNF. A 3NF table that does not have multiple overlapping candidate keys is guaranteed to be in BCNF
- 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.

Denormalization

Sometimes, it's better not to normalize, even to combine tables that were already separate. Criteria to consider:

- decomposition makes complex queries slower
- decomposition makes simple queries faster
- decomposition makes simple updates faster
- decomposition reduces storage
- decomposition can sometimes increase storage