1. DB Normalisation

Prerequisites

- PK
- Candidate Key
- Prime Attributes
- Non-prime Attributes

Student (EnrollmentNumber, Email, Name, Phone, City, Age)

CandidateKeys:

1. Email
2. Phone
3. EnrollmentNumber

PK

EnrollmentNumber

Following are the important differences between Primary Key and Candidate key.

#	criteria	Primary Key	Candidate key
1	Definition	Primary Key is a unique and non-null key which identify a record uniquely in table. A table can have only one primary key.	Candidate key is also a unique key to identify a record uniquely in a table but a table can have multiple candidate keys.
2	Null	Primary key column value can not be null.	Candidate key column can have null value.
3	Objective	Primary key is most important part of any relation or table.	Candidate key signifies as which key can be used as Primary Key.

#	criteria	Primary Key	Candidate key
4	Use	Primary Key is a candidate key.	Candidate key may or may not be a primary key.

Attribute aka column aka properties.
Prime and Non-prime Attributes

The columns in a candidate key are called prime attributes, and a column that does not occur in any candidate key is called a non-prime attribute.

Non-Prime Attributes: Name, Age, City

Prime Attributes: EnrollmentNumber, Email, Name, Phone

Normalisation

Normalization is a process of organizing the data in database to avoid data redundancy (duplication), insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

The purpose of normalization is to maximize the efficiency of a database.

An anomaly is something that is different from what is normal or usual.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomalies. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

Employee Table

Employee Table

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in another then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss normalization.

Normalization

Here are the most commonly used normal forms, each new normal form provides a different level of refinement.

- First normal form(1NF)
- Second normal form(2NF)

- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)

Many databases require that you plan your tables so they can handle multiple occurrences of the same type of item.

A book might have several authors. The following table shows one way to handle such a circumstance

`BookID  ISBN  Title Author1 Author2 Date  Pages Publisher City`

Book table containing columns that are not ideal for data storing.

The Author1 and Author2 fields are able to handle up to two authors for each book, but there are two obvious problems with the table:
1. If a book has only one author, the Author2 column is wasted space.
2. There is no place to store the name of a third, fourth, or succeeding author.There is also a problem from a theoretical standpoint. One of the rules of tables is that every column in a table must represent a *unique attribute* of an entity. In the case of the Books table, the Author1 and Author2 columns represent the same attribute: an author. The technical term for columns that represent the same attribute is a *repeating group*. (Do not let the semantics fool you, since the lead and second authors of a book are separate individuals, but they are both members of the set of Authors.)

For a table to be in *first normal form (1NF),* the table must not contain any repeating groups.*first normal form (1NF):* A table is in first normal form if it contains no repeating groups.

First Normal Form Defined

A table is in first normal form (1NF) if it meets the following criteria:
1. The data are stored in a two-dimensional table.
2. There are no repeating groups.

By definition, an entity that does not have any repeating columns or data groups can be termed as the First Normal Form. In the First Normal Form, every column is unique.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Let’s take another example:

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Drawbacks for 1NF:

There should not be repeating values present in the table. Repeating values consumes a lot of extra memory and makes the search and update slow and maintenance of the database difficult. For example, so a new table needs to be created for this in order to reduce the repetition of values.

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table. In other words: Non prime attributes should not be dependent on the subset of a candidate key.

e.g.

Teacher_age should not be dependent on (teacher_id or subject).

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subject, the table can have multiple rows for the same teacher.

Teacher

Subject_Table

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Set s1={1,2,3,4};

Set s2={1};

Set s3={5}

Candidate Keys: {teacher_id, subject}.

Composite Primary Key: {teacher_id, subject}

/H

Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:

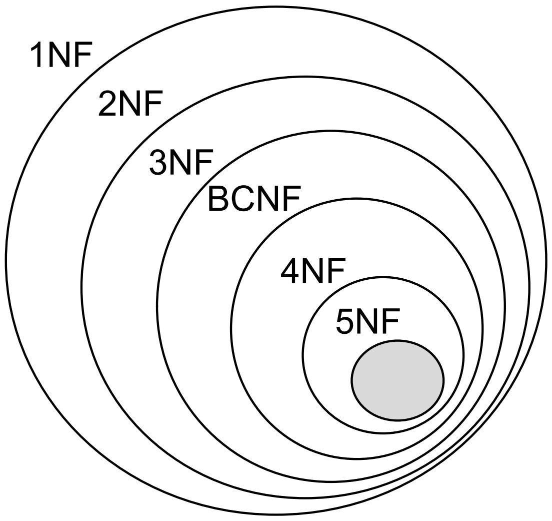
teacher_details table:

Title	teacher_id	teacher_age
Untitled	111	38
Untitled	222	38
Untitled	333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).



We Problem:

#1) 1NF (First Normal Form)

Following is how our Employees and Department table would have looked if in first normal form (1NF):

empNum	lastName	firstName	deptName	deptCity	deptCountry
1001	Andrews	Jack	Accounts	New York	United States
1002	Schwatz	Mike	Technology	New York	United States
1009	Beker	Harry	HR	Berlin	Germany
1007	Harvey	Parker	Admin	London	United Kingdom
1007	Harvey	Parker	HR	London	United Kingdom

Here, all the columns of both Employees and Department tables have been clubbed into one and there is no need of connecting columns, like deptNum, as all data is available in one place.

But a table like this with all the required columns in it, would not only be difficult to manage but also difficult to perform operations on and also inefficient from the storage point of view.

#2) 2NF (Second Normal Form)

By definition, an entity that is 1NF and one of its attributes is defined as the primary key and the remaining attributes are dependent on the primary key.

Following is an example of how the employees and department table would look like:

Employees Table:

empNum	lastName	firstName
1001	Andrews	Jack
1002	Schwatz	Mike
1009	Beker	Harry
1007	Harvey	Parker
1007	Harvey	Parker

Departments Table:

deptNum	deptName	deptCity	deptCountry
1	Accounts	New York	United States
2	Technology	New York	United States
3	HR	Berlin	Germany
4	Admin	London	United Kingdom

EmpDept Table:

Title	empDeptID	empNum	deptNum
Untitled	1	1001	1
Untitled	2	1002	2
Untitled	3	1009	3
Untitled	4	1007	4

Title	empDeptID	empNum	deptNum
Untitled	5	1007	3

Here, we can observe that we have split the table in 1NF form into three different tables. the Employees table is an entity about all the employees of a company and its attributes describe the properties of each employee. The primary key for this table is empNum.

Similarly, the Departments table is an entity about all the departments in a company and its attributes describe the properties of each department. The primary key for this table is the deptNum.

In the third table, we have combined the primary keys of both the tables. The primary keys of the Employees and Departments tables are referred to as Foreign keys in this third table.

If the user wants an output similar to the one, we had in 1NF, then the user has to join all the three tables, using the primary keys.

A sample query would look as shown below:

```
SELECT empNum, lastName, firstName, deptNum, deptName, deptCity, deptCountry

FROM Employees A, Departments B, EmpDept C

WHERE A.empNum = C.empNum

AND B.deptNum = C.deptNum

WITH UR;
```

▼ You Problem:

A particular database is normalized to satisfy a particular level of normalization (either 1NF or 2NF or 3NF). One of the tables contains, among other fields, a column for the **City**
and a column for the **Zip Code**
. Assuming that there is a **many-to-one**
mapping between the set of **Zip Code(s)**
and **City**
, we may conclude that the database is definitely **NOT**

in **NF**
form. What is the integer **x**
(1, 2, or 3)?

References:
https://www.softwaretestinghelp.com/database-normalization-tutorial/

https://www.relationaldbdesign.com/database-analysis/module3/normalization-objective.php

https://aksakalli.github.io/2012/03/12/database-normalization-and-normal-forms-with-an-example.html

14. Design Patterns - day 2 of 2