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.

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}

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:

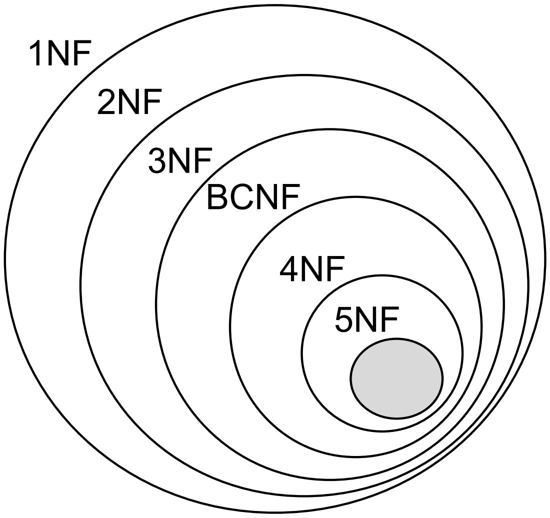
teacher_details table:

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

teacher_subject table:

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

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



Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

In other words remove any transitive dependency from super key to non prime attribute.

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

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan

emp_zip	emp_state	emp_city	emp_district
222999	MP	Gwalior	Ratan

Functional dependency in DBMS

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.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally: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)

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

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
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;
```

#3) 3NF (Third Normal Form)

By definition, a table is considered in third normal if the table/entity is already in the second normal form and the columns of the table/entity are non-transitively dependent on the primary key.

Let’s understand non-transitive dependency, with the help of the following example.

Say a table named, Customer has the below columns:

CustomerID – Primary Key identifying a unique customer

CustomerZIP – ZIP Code of the locality customer resides in

CustomerCity – City the customer resides in

In the above case, the CustomerCity column is dependent on the CustomerZIP column and the CustomerZIP column is dependent on CustomerID.

The above scenario is called transitive dependency of the CustomerCity column on the CustomerID i.e. the primary key. After understanding transitive dependency, now let’s discuss the problem with this dependency.

There could be a possible scenario where an unwanted update is made to the table for updating the CustomerZIP to a zipcode of a different city without updating the CustomerCity, thereby leaving the database in an inconsistent state.

In order to fix this issue, we need to remove the transitive dependency that could be done by creating another table, say, CustZIP table that holds two columns i.e. CustomerZIP (as Primary Key) and CustomerCity.

The CustomerZIP column in the Customer table is a foreign key to the CustomerZIP in the CustZIP table. This relationship ensures that there is no anomaly in the updates wherein a CustomerZIP is updated without making changes to the CustomerCity.

#4) Boyce-Codd Normal Form (3.5 Normal Form)

By definition, the table is considered Boyce-Codd Normal Form, if it’s already in the Third Normal Form and for every functional dependency between A and B, A should be a super key.

This definition sounds a bit complicated. ***Let’s try to break it to understand it better.***

- **Functional Dependency:** The attributes or columns of a table are said to be functionally dependent when an attribute or column of a table uniquely identifies another attribute(s) or column(s) of the same table. the empNum or Employee Number column uniquely identifies the other columns like Employee Name, Employee Salary, etc. in the Employee table.
- **Super Key:** A single key or group of multiple keys that could uniquely identify a single row in a table can be termed as Super Key. In general terms, we know such keys as Composite Keys.

Let’s consider the following scenario to understand when there is a problem with Third Normal Form and how does Boyce-Codd Normal Form comes to rescue.

empNum	firstName	empCity	deptName	deptHead
1001	Jack	New York	Accounts	Raymond
1001	Jack	New York	Technology	Donald
1002	Harry	Berlin	Accounts	Samara
1007	Parker	London	HR	Elizabeth

empNum	firstName	empCity	deptName	deptHead
1007	Parker	London	Infrastructure	Tom

In the above example, employees with empNum 1001 and 1007 work in two different departments. Each department has a department head. There can be multiple department heads for each department. Like for the Accounts department, Raymond and Samara are the two heads of departments.

In this case, empNum and deptName are super keys, which implies that deptName is a prime attribute. Based on these two columns, we can identify every single row uniquely.

Also, the deptName depends on deptHead, which implies that deptHead is a non-prime attribute. This criterion disqualifies the table from being part of BCNF.

To solve this we will break the table into these different tables as mentioned below:

Employees Table:

empNum	firstName	empCity	deptNum
1001	Jack	New York	D1
1001	Jack	New York	D2
1002	Harry	Berlin	D1
1007	Parker	London	D3
1007	Parker	London	D4

Department Table:

deptNum	deptName	deptHead
D1	Accounts	Raymond
D2	Technology	Donald
D1	Accounts	Samara
D3	HR	Elizabeth
D4	Infrastructure	Tom

▼ 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