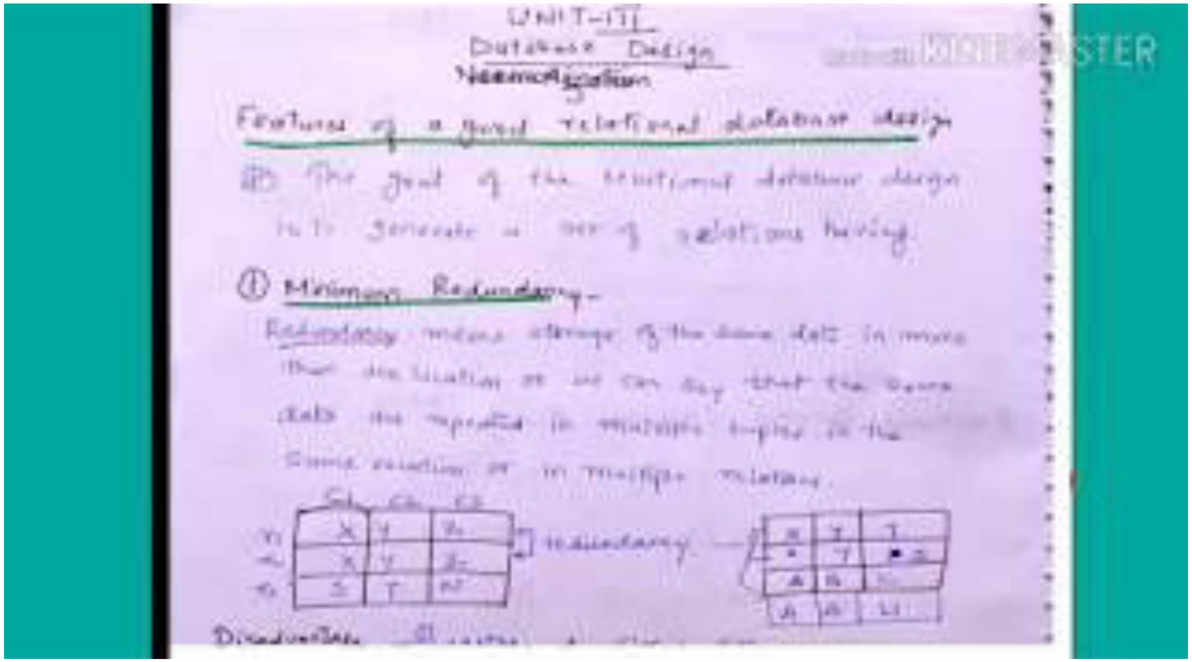
Database Design

Features of Good Relational Design

- Reflects real-world structure of the problem
- Can represent all expected data over time
- Avoids redundant storage of data items
- Provides efficient access to data
- Supports the maintenance of data integrity over time
- Clean, consistent, and easy to understand

Features of Good Relational Design



Normalization

Normalization is the process of reorganizing data in a database so that it meets two basic requirements:

- There is no redundancy of data, all data is stored in only one place.
- Data dependencies are logical, all related data items are stored together.

Normalization is important for many reasons, but chiefly because it allows databases to take up as little disk space as possible, resulting in increased performance.

Normalization is also known as data normalization.

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

Normalization

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

Anomaly

- **Insertion Anomaly**
- An *insertion anomaly* occurs when you are inserting inconsistent information into a table. When we insert a new record, such as account no. A-306 we need to check that the branch data is consistent with existing rows

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	800	1111111	Round Hill	Horseneck	8000800

Insertion anomaly - Insert account A-306 at Round Hill

Update Anomaly

- If a branch changes address, such as the Round Hill branch. we need to update all rows referring to that branch. Changing existing information incorrectly is called an *update anomaly*.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Palo Alto	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Manus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Update Anomaly - Round Hill branch address

Deletion Anomaly

- A *deletion anomaly* occurs when you delete a record that may contain attributes that shouldn't be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in Figure, all of the branch information disappears.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

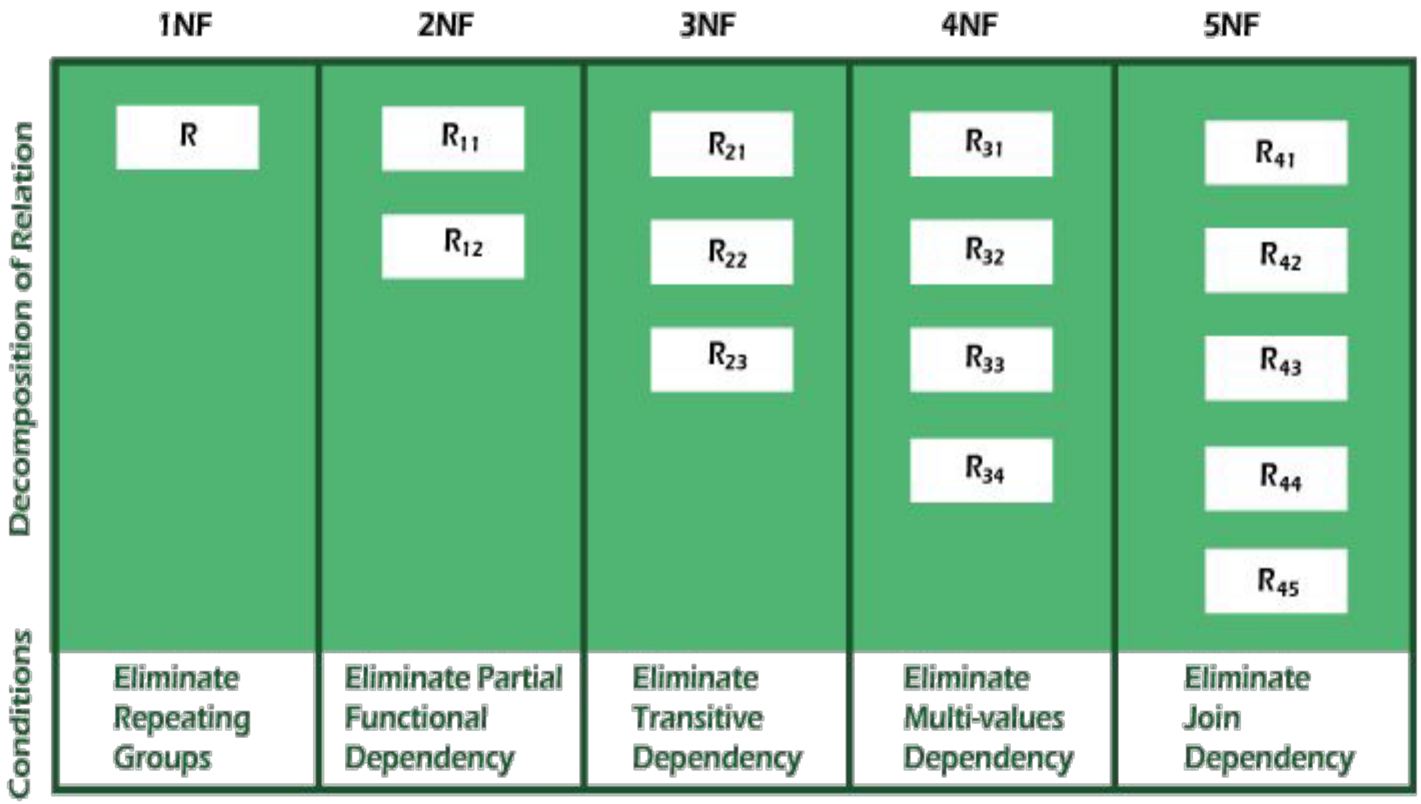
Deletion anomaly - Bank Account

- The problem with deleting the A-101 row is we don’t know where the Downtown branch is located and we lose all information regarding customer 1313131. To avoid these kinds of update or deletion problems, we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Types of Normal Forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.
- An equivalent definition is that each attribute is non decomposable and is functionally dependent on the key.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form(2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key.

- **Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form(3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

- X is a super key.
- Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Transitive dependency

Transitive dependency

Consider attributes A, B, and C, and where

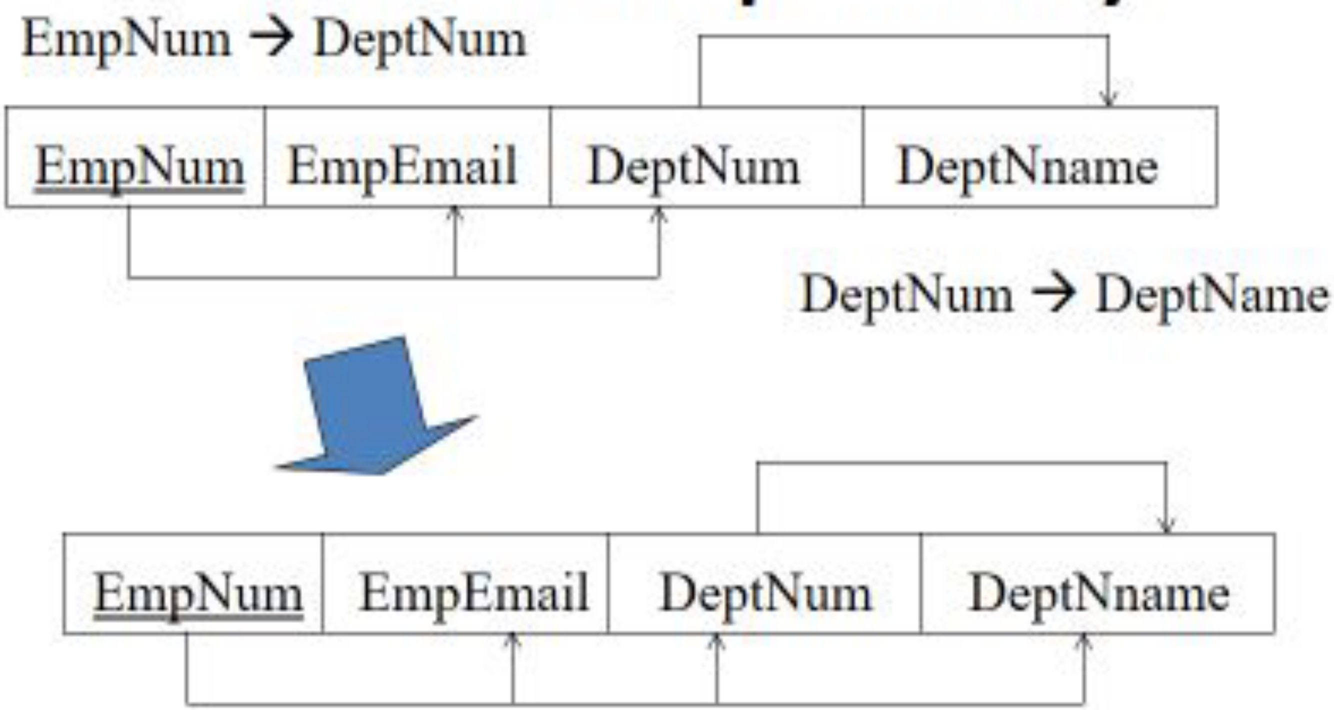
$A \rightarrow B$ and $B \rightarrow C$.

Functional dependencies are transitive, which means that we also have the functional dependency

$A \rightarrow C$

We say that C is transitively dependent on A through B.

Transitive dependency



DeptName is *transitively dependent* on EmpNum via DeptNum

EmpNum → DeptName

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:
{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}. ...so on
Candidate key: {EMP_ID}

- **Non-prime attributes:** In the given table, all attributes except EMP_ID are non-prime.
- Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.
- That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: **Let's assume there is a company where employees work in more than one department.**

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

- The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
- To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1.EMP_ID → EMP_COUNTRY
2.EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

- A superkey is **a combination of columns that uniquely identifies any row within a relational database management system (RDBMS) table**.

- A candidate key is obtained from a super key only.

- **CANDIDATE KEY** in SQL is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.