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.

What is Normalization?

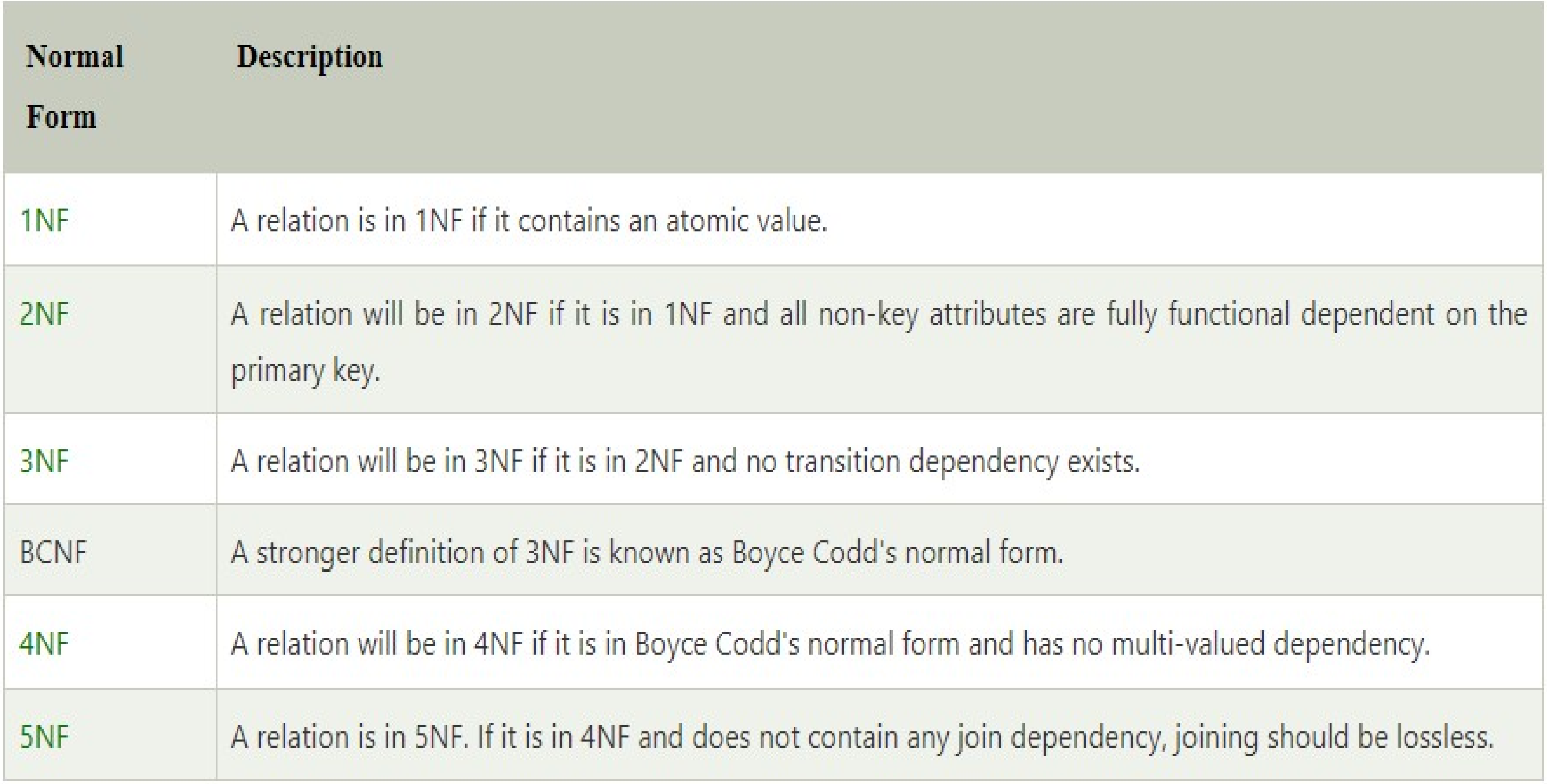
- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

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.

Data modification anomalies can be categorized into three types:

- Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- Updatation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.



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.

Advantages of Normalization

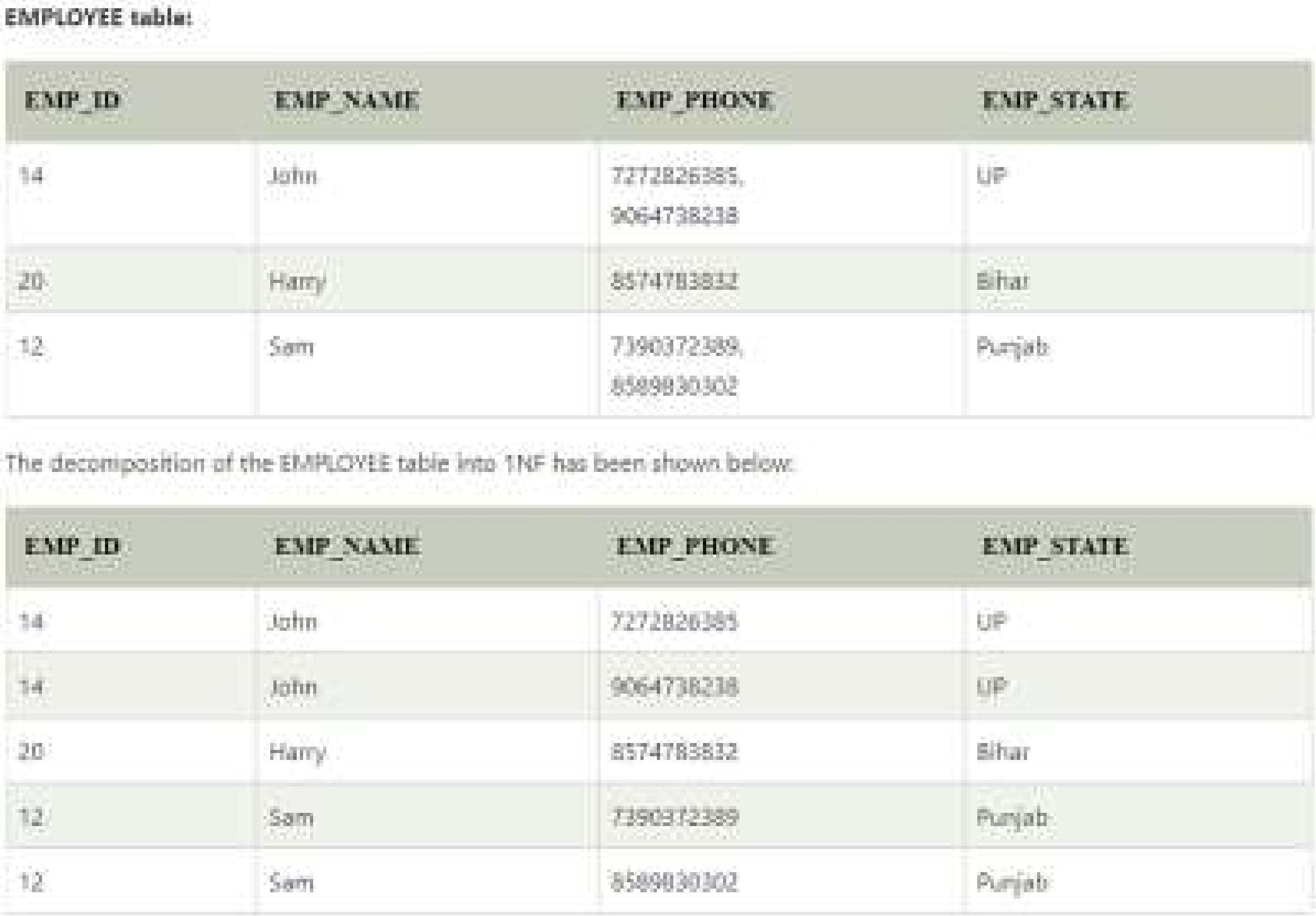
- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

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

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.



EMPLOYEE table:

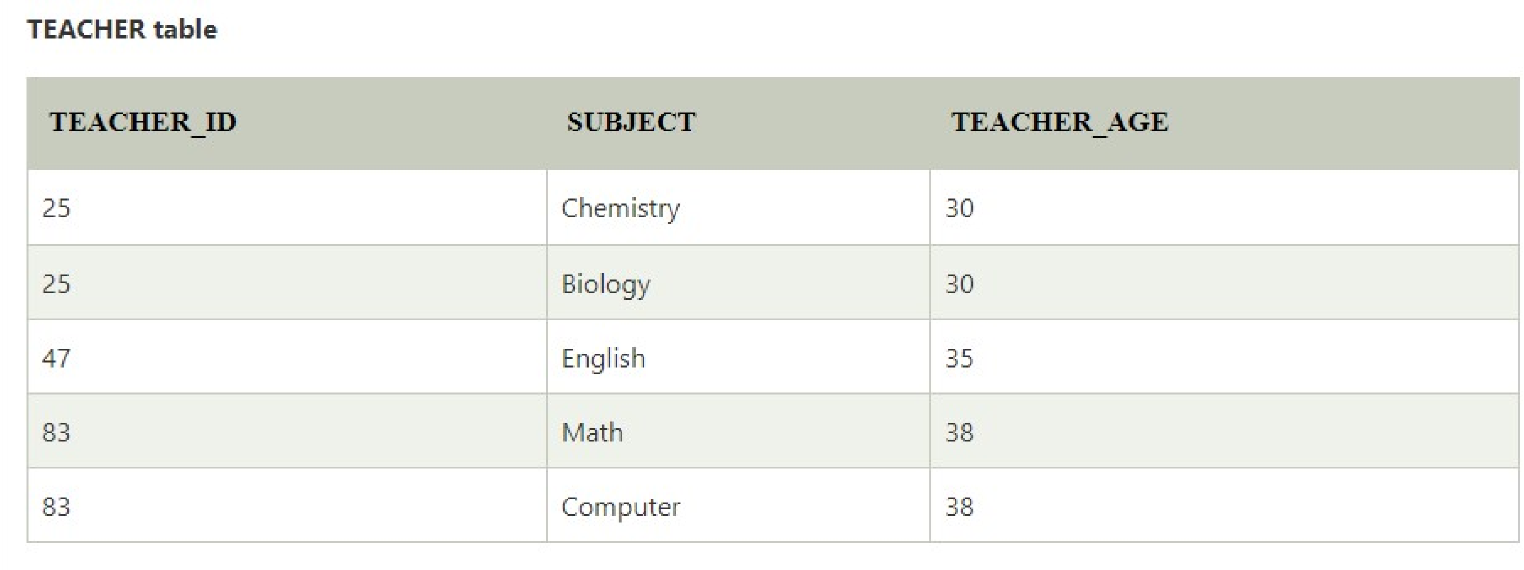
EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

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.

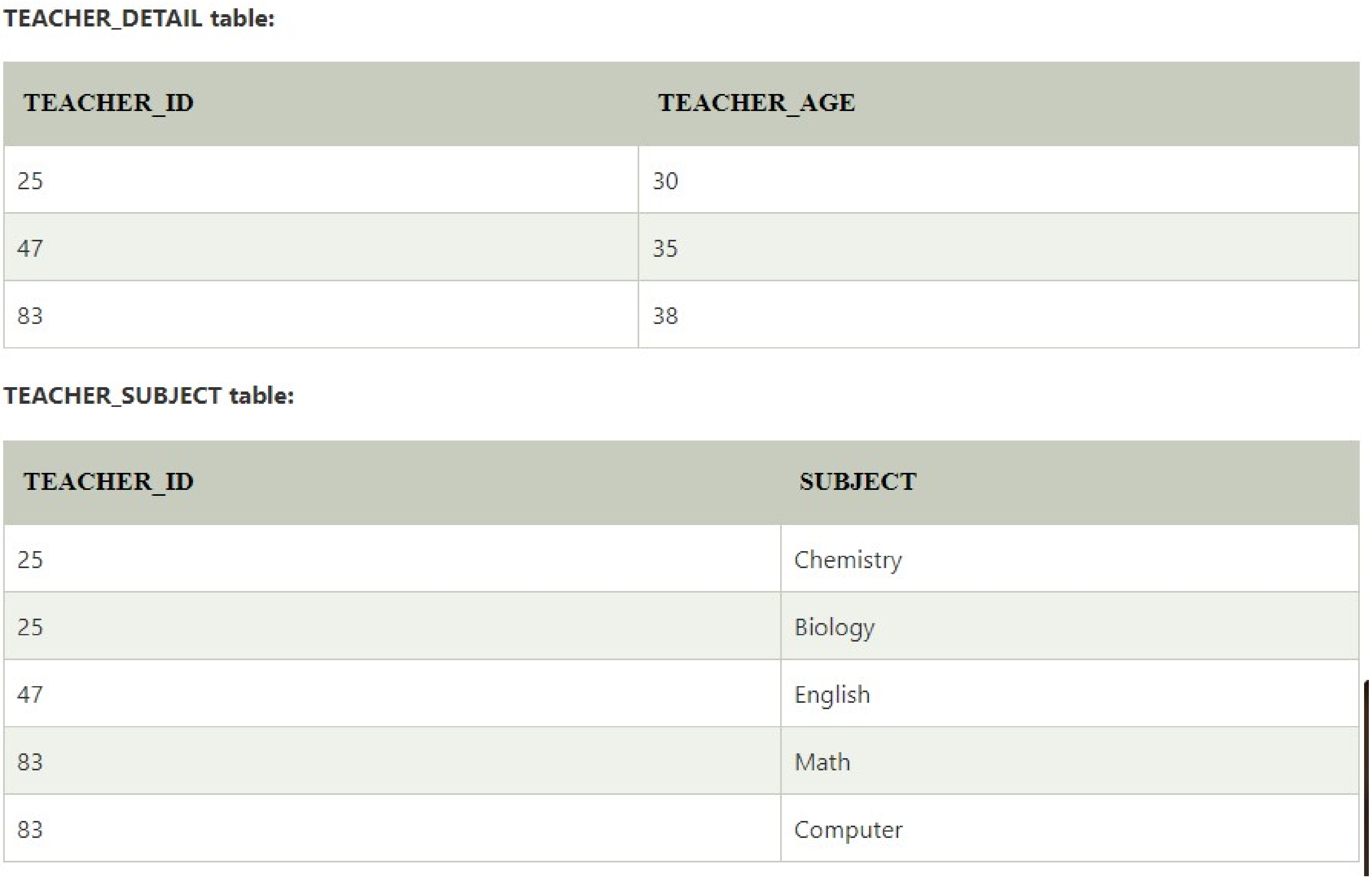


TEACHER table

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_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

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

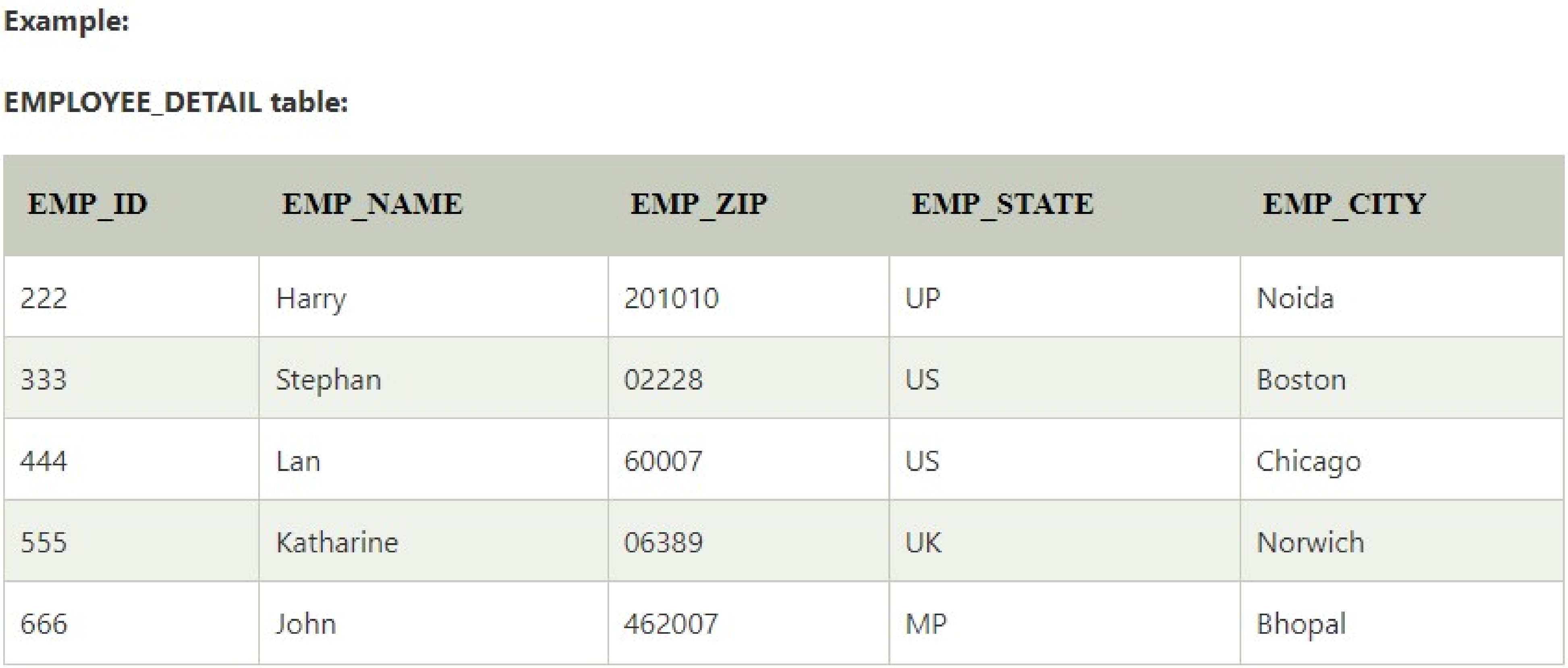
Third Normal Form (3NF)

- 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 → Y.

X is a super key.

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



Example:

EMPLOYEE_DETAIL table:

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

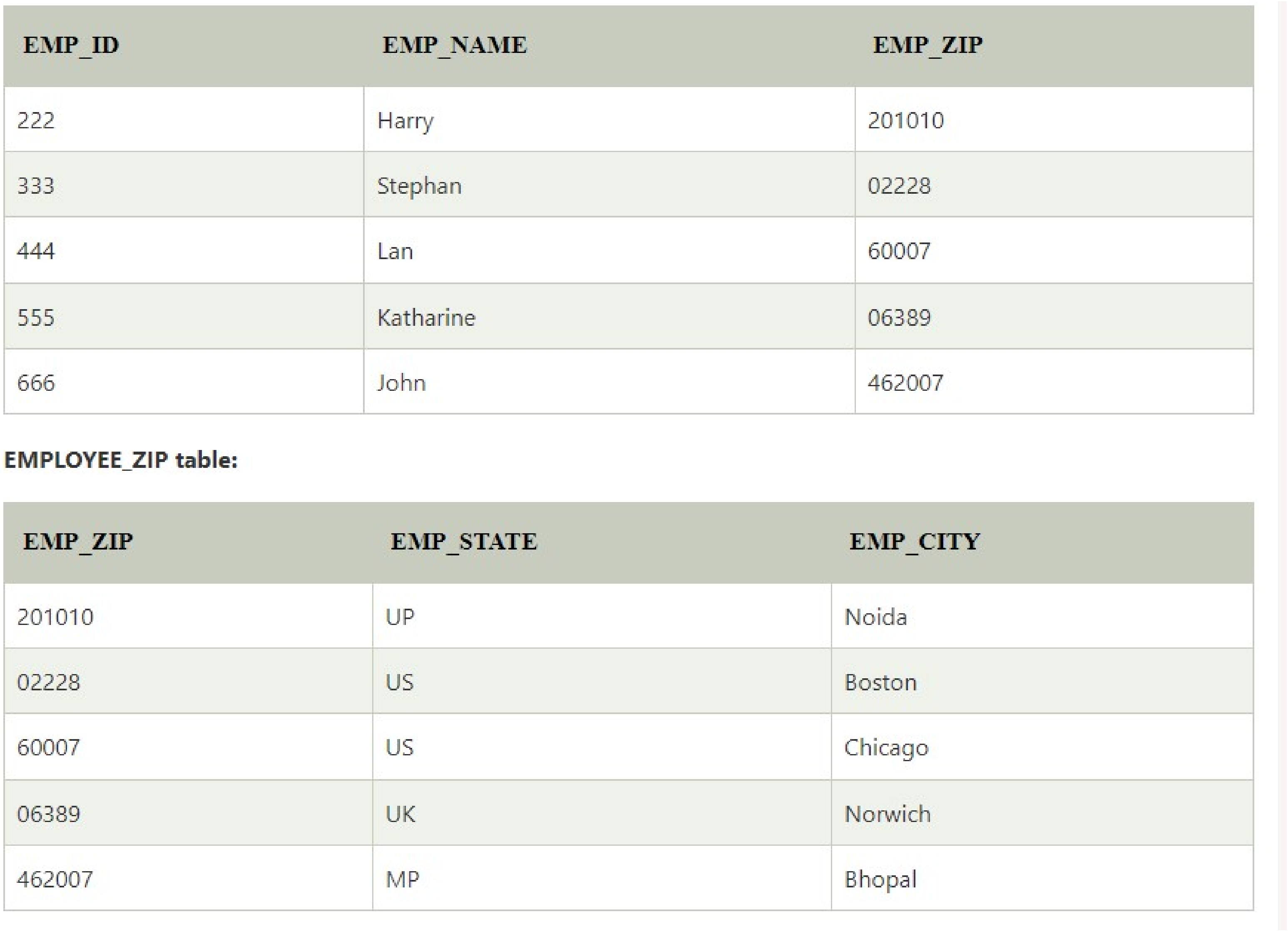
Super key in the table above:

{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.



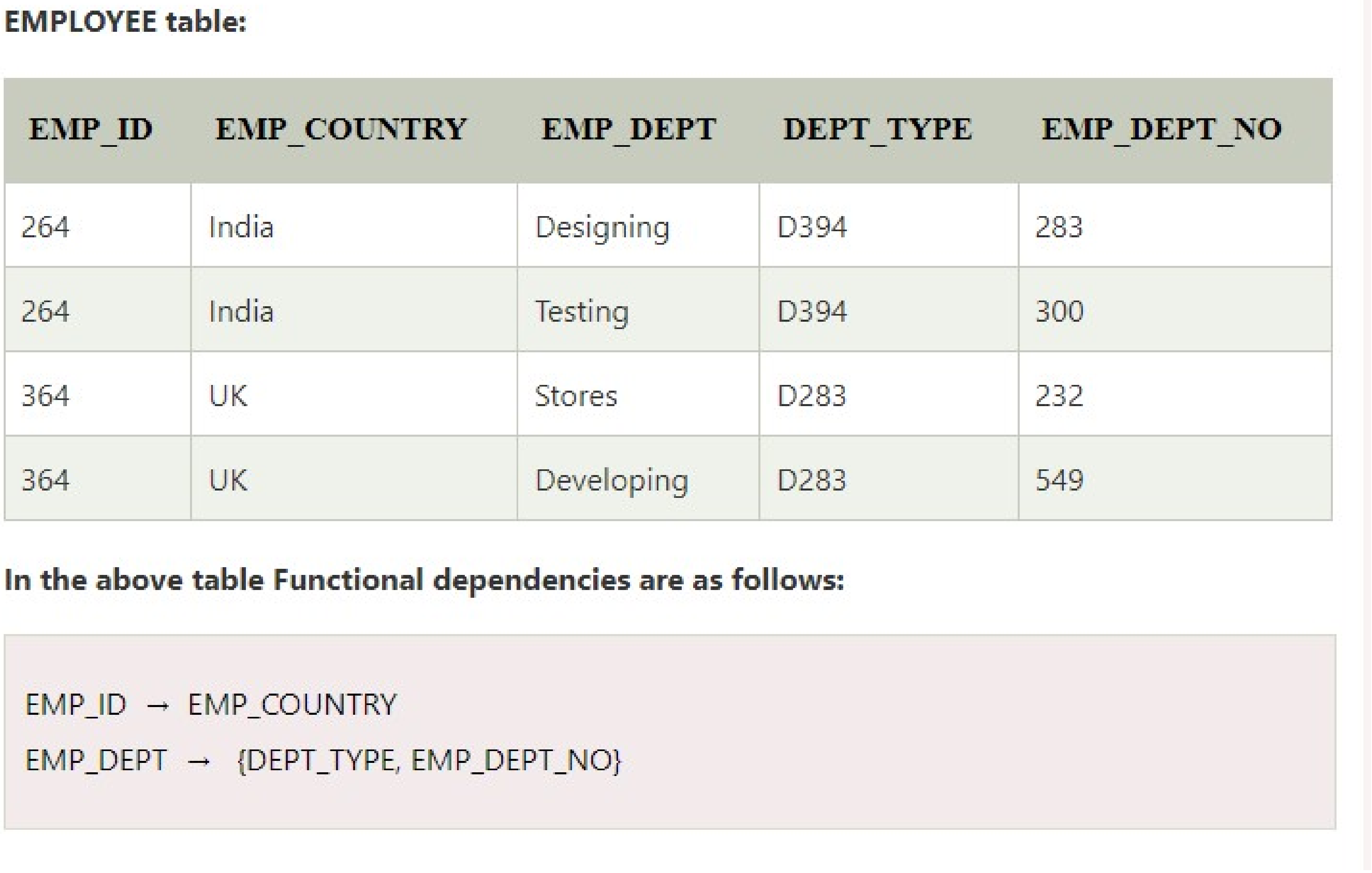
EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

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 → 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.



EMPLOYEE table:

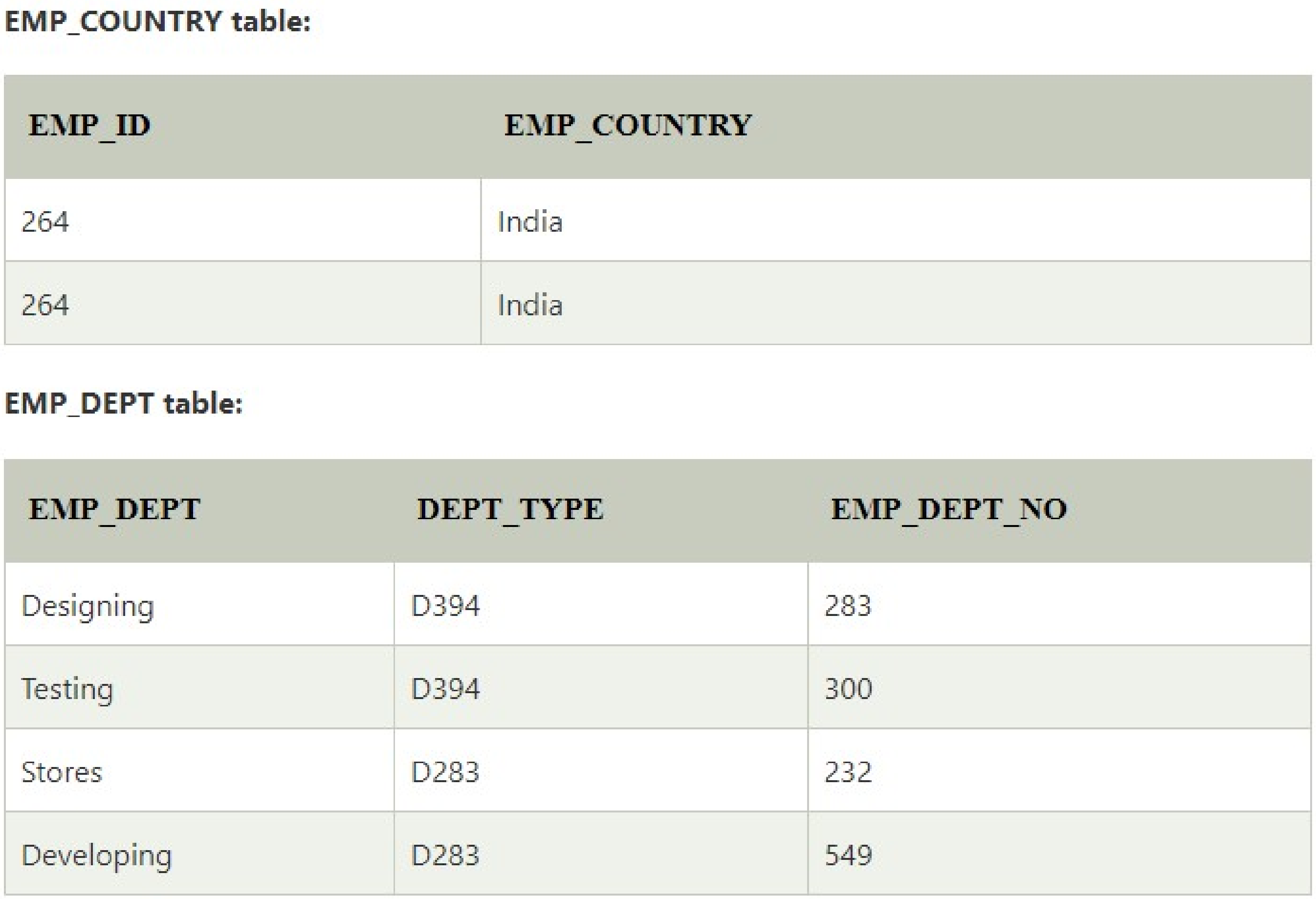
EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

```
EMP_ID  →  EMP_COUNTRY
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:

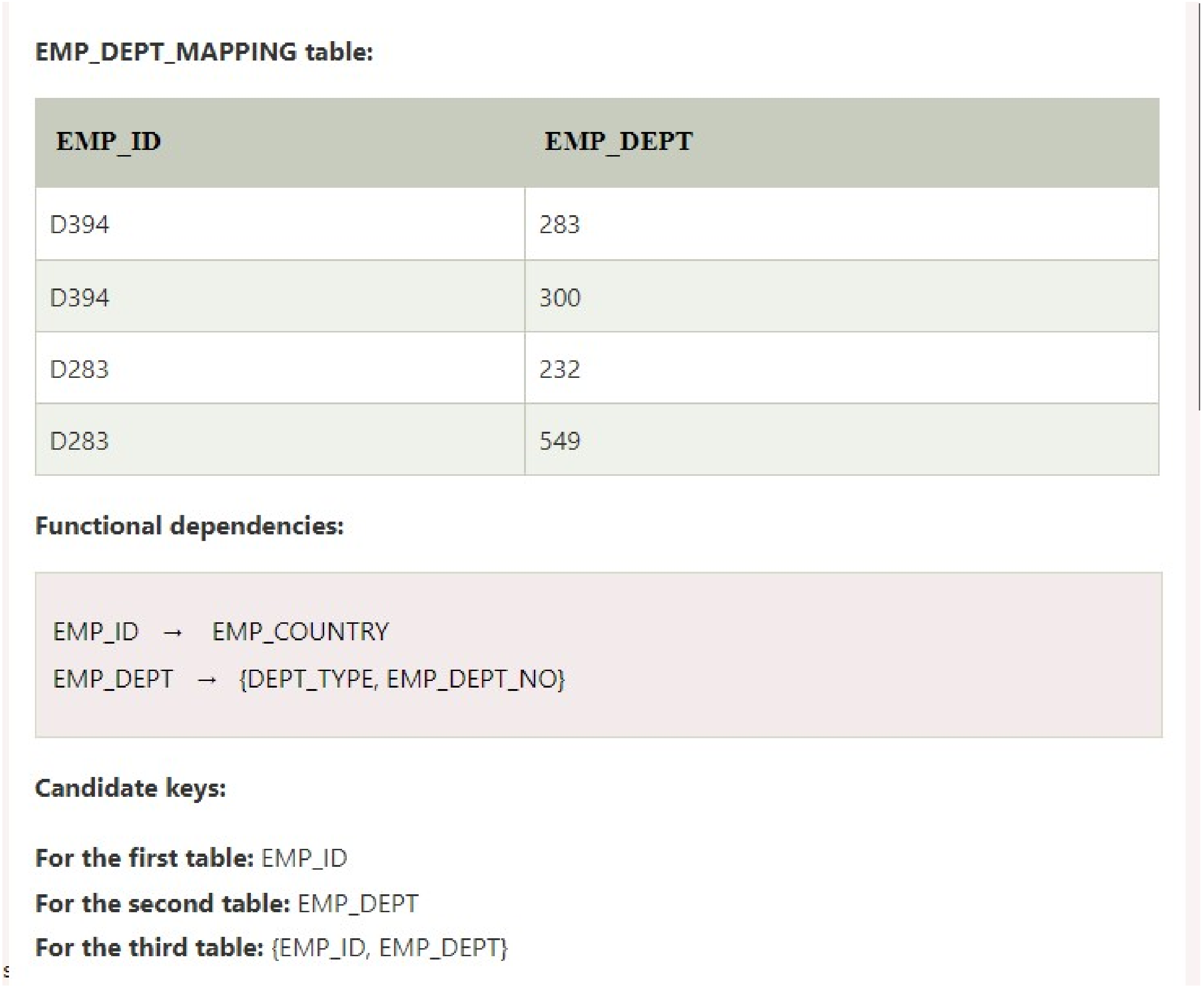


EMP_COUNTRY table:

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:

```
EMP_ID   →   EMP_COUNTRY
EMP_DEPT   →   {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate keys:

For the first table: EMP_ID

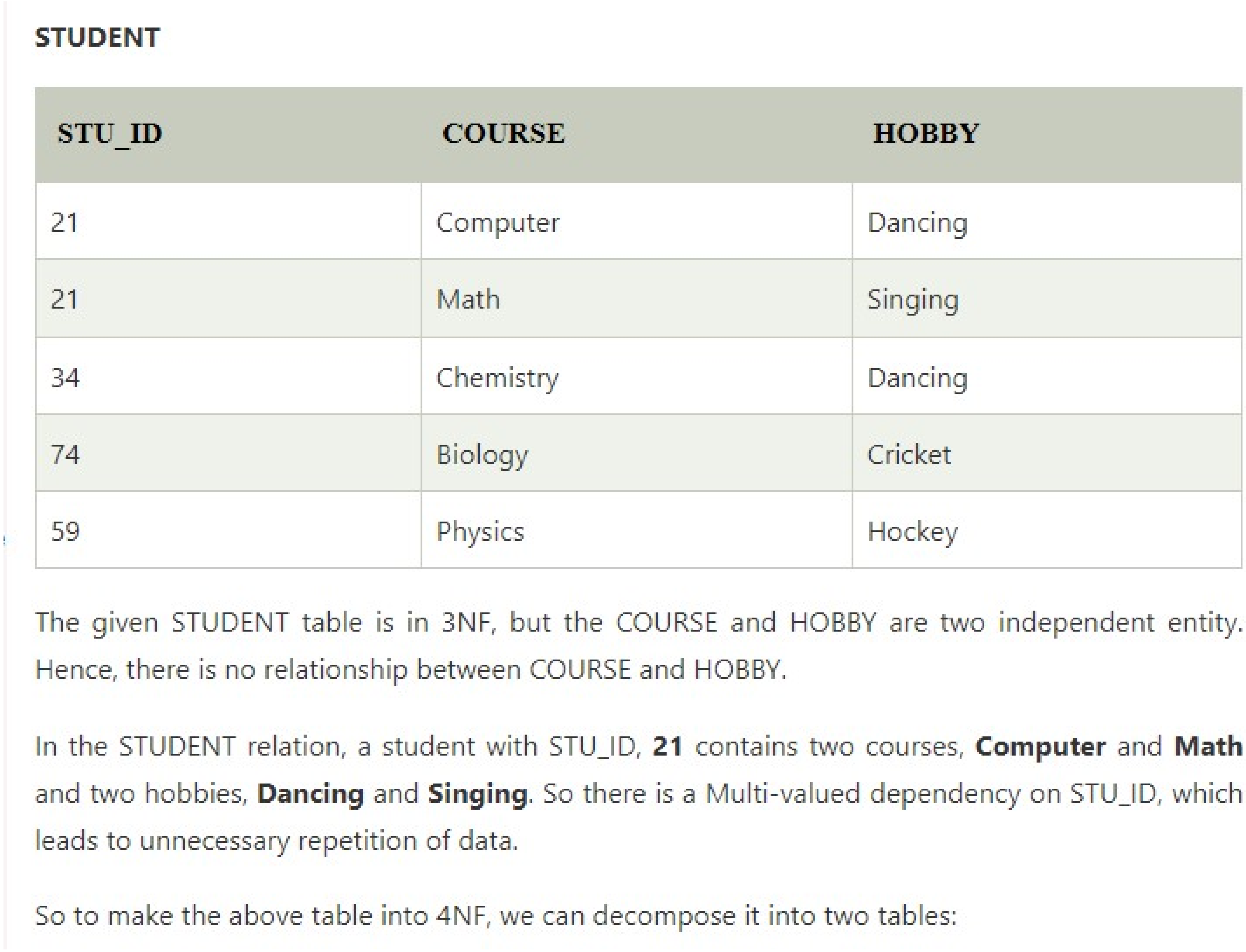
For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.



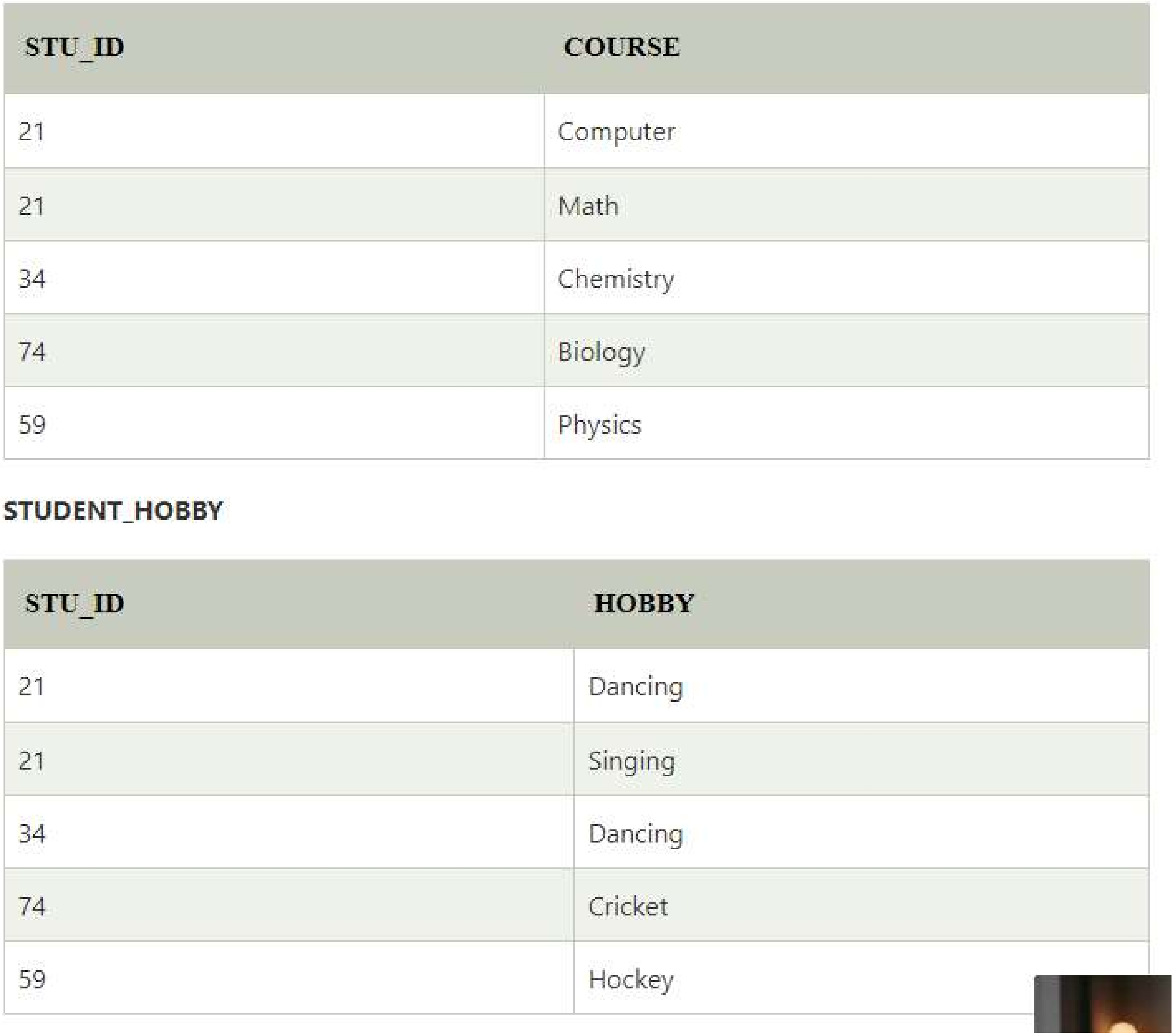
STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:



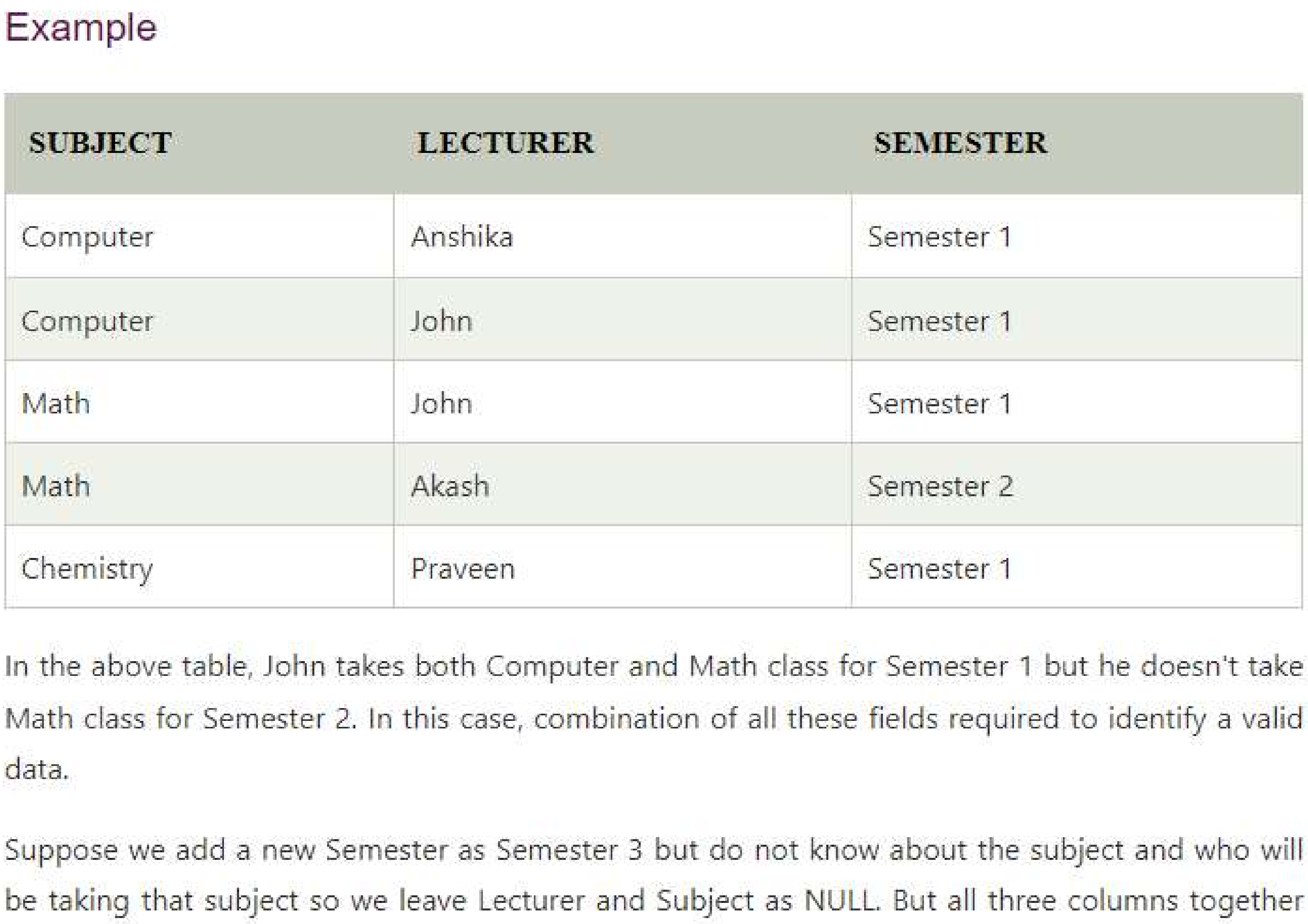
STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

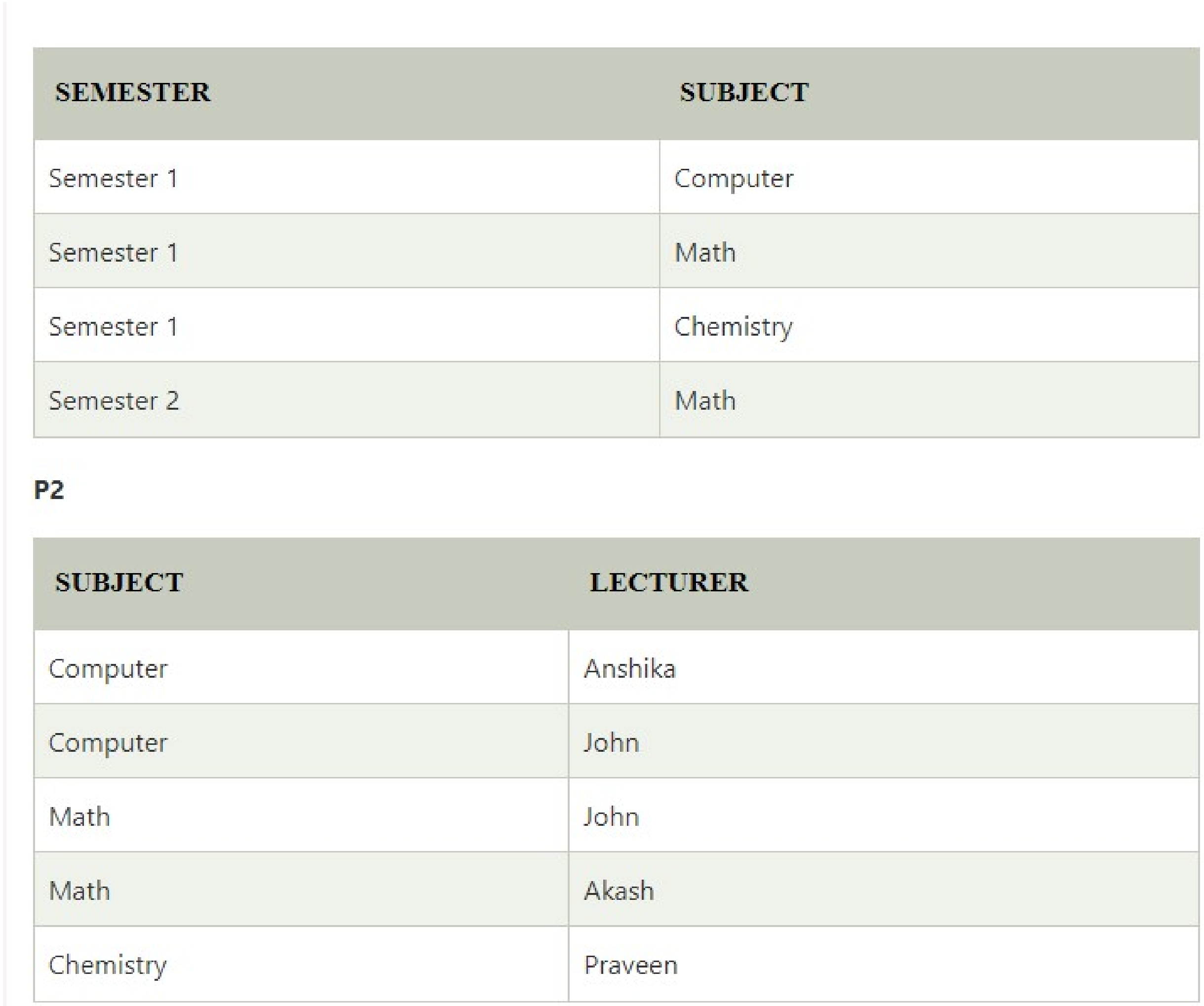


Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together

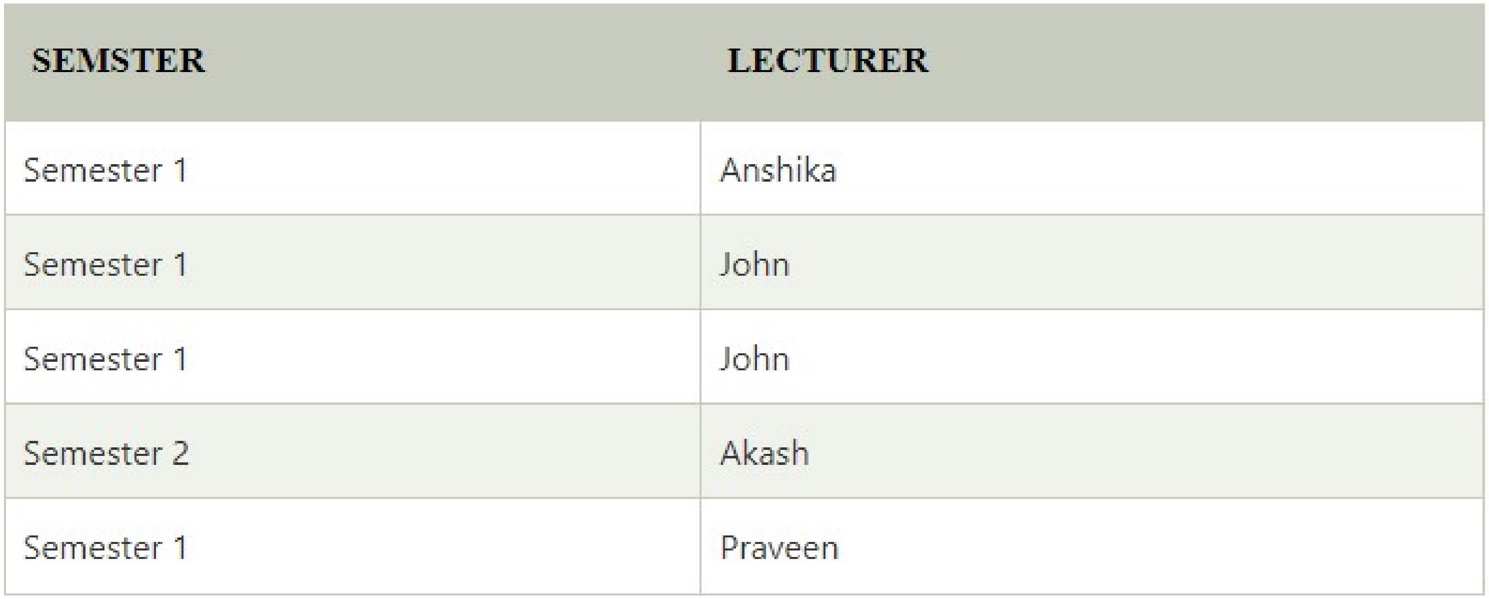


SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

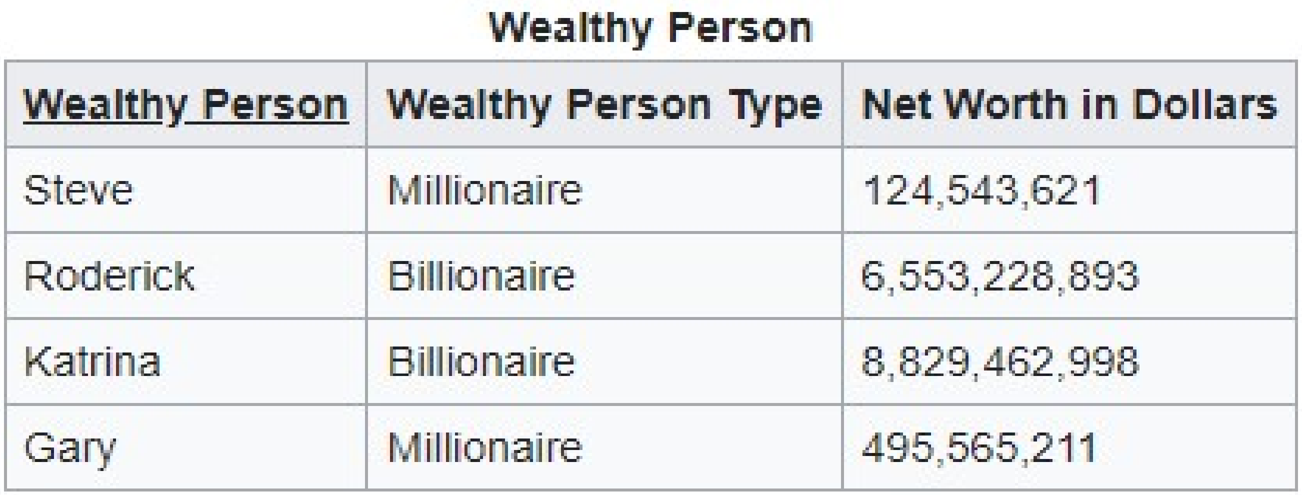


SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Domain-key normal form

- The domain/key normal form is achieved when every constraint on the relation is a logical consequence of the definition of keys and domains, and enforcing key and domain restraints and conditions causes all constraints to be met.
- Thus, it avoids all non-temporal anomalies.
- The reason to use domain/key normal form is to avoid having general constraints in the database that are not clear domain or key constraints. Most databases can easily test domain and key constraints on attributes.
- General constraints however would normally require special database programming in the form of stored procedures (often of the trigger variety) that are expensive to maintain and expensive for the database to execute.
- Therefore, general constraints are split into domain and key constraints.

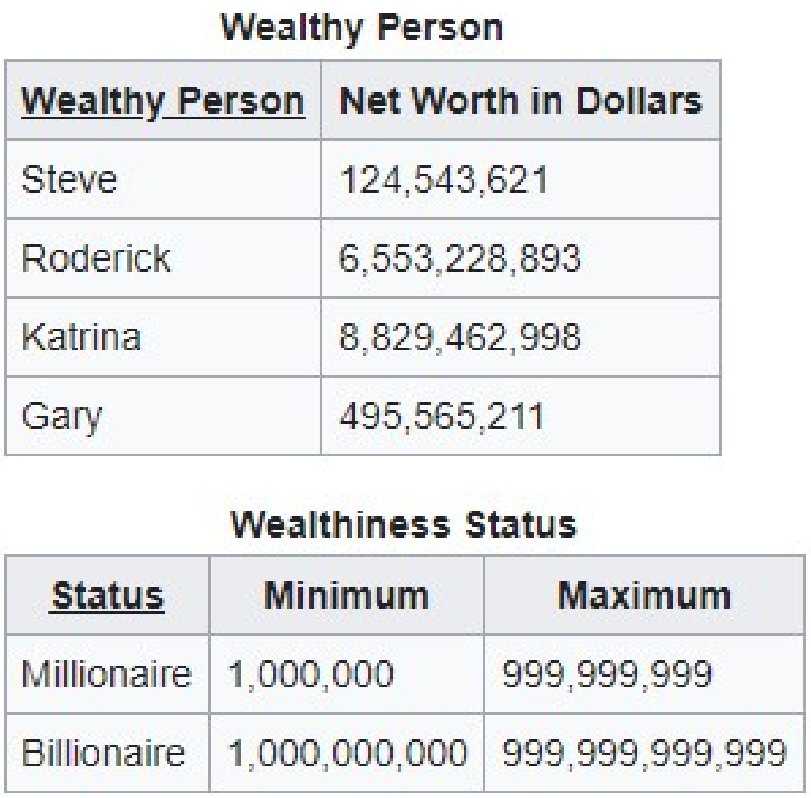
- It's much easier to build a database in domain/key normal form than it is to convert lesser databases which may contain numerous anomalies.
- However, successfully building a domain/key normal form database remains a difficult task, even for experienced database programmers.
- Thus, while the domain/key normal form eliminates the problems found in most databases, it tends to be the most costly normal form to achieve.
- However, failing to achieve the domain/key normal form may carry long-term, hidden costs due to anomalies which appear in databases adhering only to lower normal forms over time.



Wealthy Person

Wealthy Person	Wealthy Person Type	Net Worth in Dollars
Steve	Millionaire	124,543,621
Roderick	Billionaire	6,553,228,893
Katrina	Billionaire	8,829,462,998
Gary	Millionaire	495,565,211

- (Assume that the domain for Wealthy Person consists of the names of all wealthy people in a pre-defined sample of wealthy people; the domain for Wealthy Person Type consists of the values 'Millionaire' and 'Billionaire'; and the domain for Net Worth in Dollars consists of all integers greater than or equal to 1,000,000.)
- There is a constraint linking Wealthy Person Type to Net Worth in Dollars, even though we cannot deduce one from the other.
- The constraint dictates that a Millionaire will have a net worth of 1,000,000 to 999,999,999 inclusive, whilst a Billionaire will have a net worth of 1,000,000,000 or higher.
- This constraint is neither a domain constraint nor a key constraint; therefore we cannot rely on domain constraints and key constraints to guarantee that an inconsistent Wealthy Person Type / Net Worth in Dollars combination does not make its way into the database.
- The DKNF violation could be eliminated by removing the Wealthy Person Type column. The wealthy person's status as a millionaire or billionaire is determined by their Net Worth in Dollars, as defined in the Wealthiness Status table, so no useful information is lost.



Wealthy Person

Wealthy Person	Net Worth in Dollars
Steve	124,543,621
Roderick	6,553,228,893
Katrina	8,829,462,998
Gary	495,565,211

Wealthiness Status

Status	Minimum	Maximum
Millionaire	1,000,000	999,999,999
Billionaire	1,000,000,000	999,999,999,999