Normalization

CMSC 206 – Database Management Systems

Revision History
10 Oct 2022 : Examples update – Ria Borromeo
26 Sep 2019 : Main content – Thomas Laurent

Outline

1. Introduction
2. Functional Dependencies
3. Normal Forms
4. Normalization and Table Decomposition

Introduction

- We now have the tools to design DBs, but how do we make sure that our DB will be usable, i.e. easy to maintain and to query? Until now we have been jusing our common sense.
- How? Through Normalization. ***Normalization*** is the process of transforming a relational model into another equivalent model that follows a set of rules that make it avoid redundancy and thus make it easier to maintain and query.
- The rules of normalization rely on the concept of functional dependencies, which we will discuss in the next topic.
- First, let’s see an example of a database that is not normalized to get a feeling of why we want to normalize our database.

Example of a Non-Normalized DB Schema[1]

- **Delivery**(SupplierId, SupplierTown, ProductID, ProductType, Quantity)
- The supplier (*SupplierId)* from town *SupplierTown* delivers a *Quantity* of product (*ProductID)* of type *ProductType*.
- Let’s see intuitively why this schema is not optimal.

[1]Example from Professor Jean Yves Martin, Ecole Centrale de Nantes

Problems in the Previous Example

- The schema introduces redundancy (*SupplierTown* and *ProductType* are recorded for each delivery) thus it increases the risk of incoherence and problems.
- What happens if a supplier changes town and we want to insert a new tuple with the new town without updating the database first?
- What happens if we do not update all tuples in the database when there is a change?
- What happens if we want to record a supplier before he makes a delivery?

Problems in the Previous Example

SupplierId	SupplierTown	ProductID	ProductType	Quantity
3	Nantes	52	Furniture	12
22	Paris,	10	Computer	6
22	Paris	25	Paper	200
3	Nantes	25	Paper	500
3	Angers	10	Laptop	15

Table 1. A non-normalized DB with inconsistencies

- If we query the DB for the town of the supplier with SupplierId = 3, what should it return?
- The type of product 10 changed so we will not get the same statistic on laptop deliveries if we query using the ProductType column or the ProductID column.

[1]Example from Professor Jean Yves Martin, Ecole Centrale de Nantes

Functional Dependencies

Definition

- A functional dependency is a constraint between two sets of attributes X and Y of a database relational schema.
- We say that Y is functionally dependent on X, and we note X → Y when, for every valid tuples t_1 and t_2 for the database (whether they are in the database or not), if t_1 [X] = t_2 [X] then t_1 [Y] = t_2 [Y]. i.e. if two tuples have the same values for attributes in X then they have the same values for attributes in Y.

$$(X \rightarrow Y) \Leftrightarrow (\forall t1, t2; t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$$

Remarks

- Functional dependencies are part of the semantics of the data and must be declared by the database designer.
- They are not intrinsic to the DBMS but can be implemented through constraints such as keys.
- Functional dependencies are dependent on the schema of the database, not on a particular state of the database. It is not possible to determine functional relations from the tuples in the database.

Examples

In the previous example of a non-normalized DB schema, we could define the following functional dependencies:

- {DeliveryId} → {DeliveryId, SupplierId, SupplierTown, ProductId, ProductType, Quantity}, this is called a trivial functional dependency, as DeliveryId is on both the le�t and right side
- {SupplierId} → {SupplierTown}
- {DeliveryId} → {ProdcutId}
- {ProductId} → {ProducType}, ProductType is ***transitively dependent*** on DeliveryId via ProductId
- {DeliveryId} → {ProductType}
- ...

Summary

- A ***functional dependency*** is a relationship between two sets of attributes in relation.
- The ***determinant*** is a set of attributes, which, when their values are known, fully determine the values of the dependent set.
- In other words, given a relation, and given the values of a set of determinant attributes, you could look up tuples (rows) in the relation with matching determinant values, and they should all have the same set of values for the attributes of the dependent set. (You might find zero, one, or many matching tuples.)
- A ***candidate key*** is a special case of determinant which determines all attributes of a relation.
- Although not part of the fundamental relational model, functional dependencies are useful in query optimization and help reason database schema normalization.

Source: "Functional Dependency." Wiki.c2.Com, https://wiki.c2.com/?FunctionalDependency

Normal Forms

Introduction

Normal forms are a set of rules that a DB schema should follow to ensure it does not facilitate update anomalies so that the DB is easy to maintain.

There are several normal forms (e.g. 1NF, 2NF, 3NF, BCNF, 4NF, ETNF, …), each stricter than the previous one. We will discuss Normal Forms 1 – 3 and BCNF in this lecture.

Although normal forms are the best practice, they are not always followed in the industry, which can make your life harder when dealing with DBs. After this class, I hope you strive to implement databases that are 3NF/BCNF.

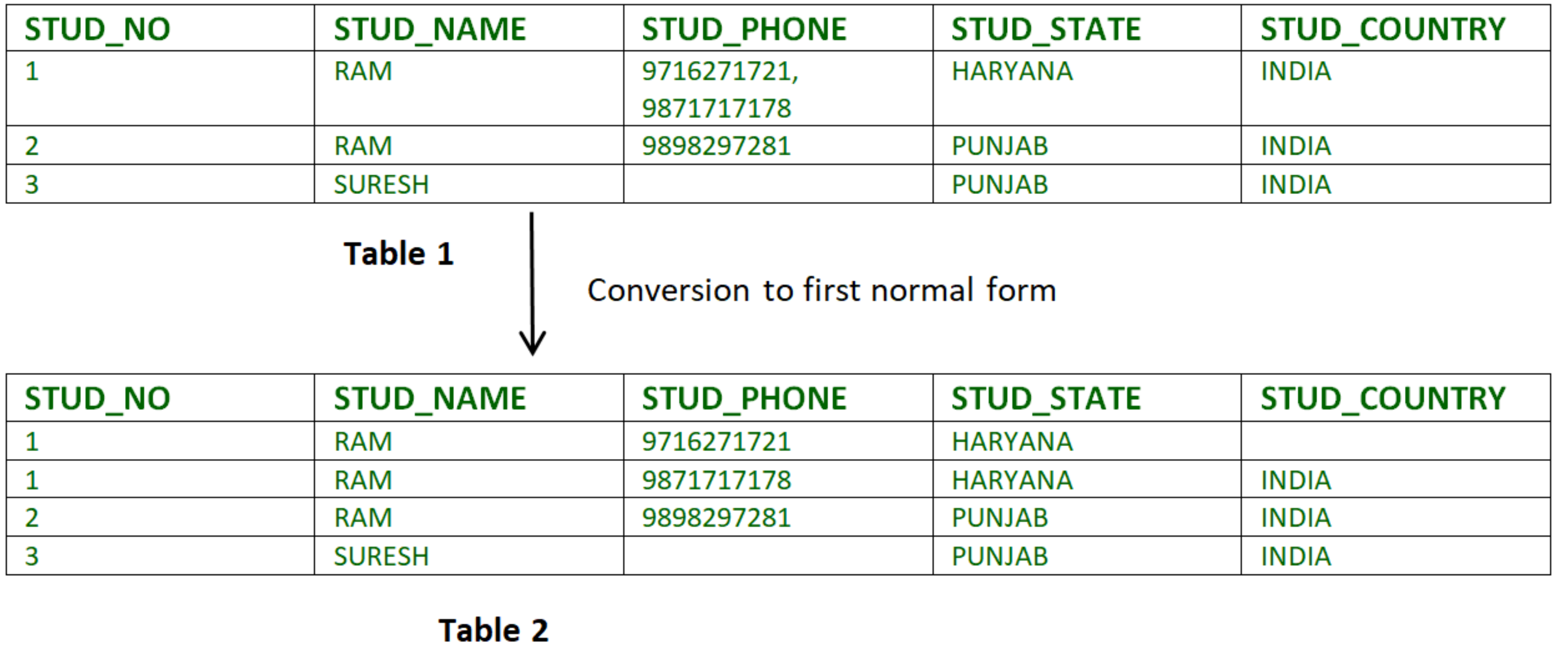
First Normal Form (1NF)

A table is in 1NF iff:

1. There are only single valued attributes.
2. Attribute domain does not change.
3. There is a unique name for every attribute/column.
4. The order in which data is stored does not matter.

Source: "First Normal Form (1NF)." GeeksforGeeks, 15 July 2022, https://www.geeksforgeeks.org/first-normal-form-1nf/

First Normal Form (1NF) – Example 1



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

Source: "First Normal Form (1NF)." GeeksforGeeks, 15 July 2022, https://www.geeksforgeeks.org/first-normal-form-1nf/

First Normal Form (1NF) – Example 2

In the following table, Courses is a multi-valued attribute so it is not in 1NF.

```
ID    Name    Courses
--------------------
1     A       c1, c2
2     E       c3
3     M       C2, c3
```

The table below is in 1NF as there are no multi-valued attributes.

```
ID    Name    Course
--------------------
1     A         c1
1     A         c2
2     E         c3
3     M         c2
3     M         c3
```

NOTE: A database design is considered as bad if it is not even in the First Normal Form (1NF).

Second Normal Form (2NF)

Relations with a primary key composed of two or more attributes.

Second Normal Form (2NF) is based on the concept of full functional dependency. It applies to relations with composite keys. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.

To be in second normal form, a relation **must be in first normal form** and **relation must not contain any partial dependency**.

A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Second Normal Form (2NF)

In other words,

A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF).

Second Normal Form (2NF) – Example 1

Consider the following table:

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

Note that there are many courses having the same course fee.

Here,

- COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
- COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
- COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

However, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,
we need to split the table into two tables such as :
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE

Second Normal Form (2NF) – Example 2

Consider the following functional dependencies in relation R (A, B, C, D)

```
AB -> C  [A and B together determine C]
BC -> D  [B and C together determine D]
```

In the above relation, AB is the only candidate key and there is a no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

Third Normal Form (3NF)

A relation is in 3NF if there is no transitive dependency for non-prime attributes as well as it is in the second normal form.

A relation is in 3NF if at least one of the following conditions hold in every non-trivial functional dependency X→Y:

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Third Normal Form (3NF)

In other words,

A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

NOTE: If A->B and B->C are two FDs then A->C is called transitive dependency.

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

Source: "Third Normal Form (3NF)." GeeksforGeeks, 31 Jul. 2019, https://www.geeksforgeeks.org/third-normal-form-3nf/.

Third Normal Form (3NF) – Example 1

In relation STUDENT given in the table below,

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

FD set:
{STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}

Candidate Key:
{STUD_NO}

For this relation, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

```
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)
```

Third Normal Form (3NF) – Example 2

Consider relation R(A, B, C, D, E)

```
A -> BC,
CD -> E,
B -> D,
E -> A
```

All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

Note: Third Normal Form (3NF) is considered adequate for normal relational database design because most of the 3NF tables are free of insertion, update, and deletion anomalies. Moreover, 3NF always ensures functional dependency preserving and lossless.

Boyce Codd Normal Form (BCNF)

The Boyce-Codd Normal Form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation.

A relation is in BCNF if and only if, every determinant is a Form (BCNF) candidate key.

To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.

Source: "Boyce-Codd Normal Form (BCNF)." GeeksforGeeks, 27 Jan. 2020, https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/.

Fourth and Fifth Normal Form

These normal forms are less used.

You can read on them here:

“Introduction of 4th and 5th Normal Form in DBMS.” *GeeksforGeeks*, 2 Feb. 2022, https://www.geeksforgeeks.org/introduction-of-4th-and-5th-normal-form-in-dbms/.

Normalization and Table Decomposition

Introduction

Now you know what normal forms are and hopefully the examples have given you an idea of how to transform schemas in order to make them respect the normal forms.

Let us then introduce the concept of decomposition of a relation without loss of information to formalize this intuition.

Heath Theorem

Relation R(X, Y, Z) can be decomposed without loss of information into:

R1 =π[X,Y]

R2 =π[X,Z]

if the functional* dependency X → Y exists.

Source: “Boyce-Codd Normal Form (BCNF).” GeeksforGeeks, 27 Jan. 2020, https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/.