

# Design Theory for Relational Databases

August 7, 2022

## **Abstract**

These notes have been written and sourced from the textbook, "Database Systems - The Complete Book" (Garcia-Molina et al.) The details have just been compressed to the most important definitions and constituent, high-level descriptions for you.

# Contents

<b>1</b>	<b>Functional Dependencies</b>	<b>3</b>
1.1	Functional Dependencies . . . . .	3
1.2	The Splitting / Combining Rule . . . . .	3
1.3	Keys of Relations . . . . .	3
1.4	Superkeys . . . . .	3
1.5	Finding Keys . . . . .	3
1.6	Implied FDs . . . . .	4
1.6.1	Inference Test . . . . .	4
1.6.2	Closure Test . . . . .	4
1.6.3	Closure Test Algorithm . . . . .	4
1.7	Trivial FDs . . . . .	4
<b>2</b>	<b>Relation Schema Design</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Boyce-Codd Normal Form . . . . .	5
2.3	Decomposition into BCNF . . . . .	5
2.4	Projecting Functional Dependencies . . . . .	6
2.5	The Third Normal Form . . . . .	6
2.5.1	Benefits of 3NF and BCNF . . . . .	6
2.6	Putting Relations into 3NF (Synthesis Algorithm) . . . .	7

# 1 Functional Dependencies

## 1.1 Functional Dependencies

- A functional dependency (FD) on a relation R is a statement of the form, "If two tuples of R agree on all the attributes  $A_1, A_2, \dots, A_n$ , then they must also agree on all of another list of attributes  $B_1, B_2, \dots, B_m$ "

We write this FD formally as  $A_1A_2\dots A_n \rightarrow B_1B_2\dots B_m$  and say that  $A_1A_2\dots A_n$  functionally determine  $B_1B_2\dots B_m$ .

- **Short-Hand Conventions:**
  1. X, Y, Z represent sets of attributes
  2. A, B, C represent single attributes

## 1.2 The Splitting / Combining Rule

- **Splitting Rule:**  $A_1A_2\dots A_n \rightarrow B_1B_2\dots B_m \Leftrightarrow \{A_1A_2\dots A_n \rightarrow B_i\}$  for  $i = 1, 2, \dots, m$
- The **combining rule** is really just the opposite of the splitting rule.
- Do note that there does not exist any splitting rules for left sides. In addition, we'll generally express FD's with singleton right sides.

## 1.3 Keys of Relations

- We say that a set of one or more attributes  $\{A_1, A_2, \dots, A_n\}$  is a **key** for a relation R, if:
  1. The attributes of the key **functionally determine** all other attributes of the relation. That is, it is impossible for two distinct tuples of R to agree on all of  $A_1, A_2, \dots, A_n$
  2. No **proper subset** of  $\{A_1, A_2, \dots, A_n\}$  functionally determines all other attributes of R. That is, a key must be **minimal**.

## 1.4 Superkeys

- A set of attributes that contains a key is called a superkey, short for, "superset of a key."
- Every key is a superkey, but not every superset is a key.
- Superkeys satisfy condition (1) of a key, but not (2)

## 1.5 Finding Keys

We can generally find keys by:

1. Randomly asserting a key, k (i.e., Artificially Created Keys)
2. Asserting FD's, and using the principle of deduction to find the other key components through systematic exploration

## 1.6 Implied FDs

FD's often can be presented in several different ways, without changing the set of legal instances of the relation. We say:

1. Two sets of FD's  $S$  and  $T$  are **equivalent** if the set of relation instances satisfying  $S$  is exactly the same as the set of relation instances satisfying  $T$ .
2. More generally, a set of FD's **follows from** a set of FD's  $T$  if every relation instance that satisfies all the FD's in  $T$  also satisfies all the FD's in  $S$ .

TLDR - Transitivity Rule:  $((A \rightarrow B), (B \rightarrow C))$  implies  $(A \rightarrow C)$

### 1.6.1 Inference Test

1. To test if  $Y \rightarrow B$ , start by assuming two tuples agree in all attributes of  $Y$ .
2. Use the given FD's to infer that these tuples must also agree in certain other attributes.

### 1.6.2 Closure Test

- Suppose  $\{A_1, A_2, \dots, A_n\}$  is a set of attributes and  $S$  is a set of FD's. Then, the closure of  $\{A_1, A_2, \dots, A_n\}$  under the FD's in  $S$  is the set of attributes  $B$  such that every relation that satisfies all the FD's in set  $S$  also satisfies  $A_1A_2\dots A_n \rightarrow B$ . That is,  $A_1A_2\dots A_n \rightarrow B$  follows from the FD's of  $S$ .
- We denote the closure of a set of attributes  $A_1A_2\dots A_n$  by  $\{A_1, A_2, \dots, A_n\}^+$
- Do note that  $\{A_1A_2\dots A_n\} \subseteq \{A_1, A_2, \dots, A_n\}^+$

### 1.6.3 Closure Test Algorithm

1. If necessary, split the FD's of  $S$ , so each FD in  $S$  has a single attribute on the right.
2. Let  $X$  be a set of attributes that eventually will become the closure.
3. Repeatedly search for some FD  $B_1B_2\dots B_m \rightarrow C$  such that all of  $B_1B_2, \dots, B_m$  are in the set of attributes  $X$ , but  $C$  is not. Add  $C$  to the set  $X$  and repeat the search. Since  $X$  can only grow, and the number of attributes of any relation schema must be finite, eventually nothing more can be added to  $X$ , and this step ends.
4. The set  $X$ , after no more attributes can be added to it, is the correct value of  $\{A_1, A_2, \dots, A_n\}^+$

## 1.7 Trivial FDs

A constraint of any kind on a relation is said to be trivial if it holds for every instance of the relation, regardless of what other constraints are assumed.

One example of a trivial FD is where the right side is a subset of its left side.

Every trivial FD holds in every relation, since it says that "two tuples that agree in all of  $A_1, A_2, \dots, A_n$  agree in a subset of them.

## 2 Relation Schema Design

### 2.1 Introduction

- The goal of a relational schema design is to avoid **anomalies** and **redundancy**.
  1. **Update Anomaly:** One occurrence of a fact is changed, but not all occurrences.
  2. **Deletion Anomaly:** Valid facts are lost when a tuple is deleted.

### 2.2 Boyce-Codd Normal Form

- A feature that a relation may have. it is a condition under which the previously discussed anomalies can be guaranteed not to exist.
- A relation  $R$  is in BCNF if and only if whenever there is a non-trivial FD  $A_1A_2\dots A_n \rightarrow B_1B_2\dots B_m$  for  $R$ , it is the case that  $\{A_1, A_2, \dots, A_n\}$  is a superkey of  $R$ . **That is, the left side of every nontrivial FD must be a superkey.**
- If a relation is not in any form of BCNF, we must do something about it.

### 2.3 Decomposition into BCNF

- By repeatedly choosing suitable decompositions, we can break any relation schema into a collection of subsets of its attributes, with the following important properties:
  1. These subsets are the schemas of relations in BCNF.
  2. We need to be able to reconstruct the original relation instance exactly from the decomposed relation instances.
- **To decompose a relation into BCNF, we must:**
  1. Identify a given FD,  $X \rightarrow Y$ , which violates BCNF.
  2. Compute  $X^+$
  3. Replace  $R$  by relations with schemas:
    - a)  $R_1 = X^+$
    - b)  $R_2 = (R - X^+) \cup X$
  4. Recursively decompose  $R_1$  and  $R_2$  using this algorithm. Return the union of the results of these decompositions.

Do note that we may assume that every two-attribute relation is in BCNF. The proof is in the textbook. It's not necessary to read, but good for an overall understanding of the said mechanical processes.

## 2.4 Projecting Functional Dependencies

- Suppose we have a relation  $R$  with a set of FD's  $S$ , and we project  $R$  by computing  $R_1 = \pi_R(R)$ , for some list of attributes  $R$ .
- The FDs that will hold in  $R_1$  are:
  1. For all the attributes in  $R_1$ , compute their closure.
  2. Add to the set of FD's, all nontrivial FD's  $X \rightarrow A$ , such that  $A$  is in both  $X^+$  and an attribute of  $R_1$
- In general, we project FD's, because once we decompose a table into two sub-relations, we need to find which FDs hold for the new sub-tables. Looking at the original FDs may not be enough.

## 2.5 The Third Normal Form

- A more relaxed version of BCNF. It's used to decompose a troublesome relation where the lossless join property cannot be held accordingly.
- A relation  $R$  is in the third normal form (3NF) if:
  1.  $\{A_1, A_2, \dots, A_n\}$  is a superkey.
  2. Those of  $B_1, B_2, \dots, B_m$  that are not among the  $A$ 's, are each a member of some key (not necessarily the same key. Recall, An attribute that is a member of some key is often said to be prime.
- In general, we prefer 3NF, however, there are very minimal problems with redundancy in practice.

### 2.5.1 Benefits of 3NF and BCNF

- The following properties are important of a decomposition:
  1. **Lossless Join:** It should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.  
 The decomposition of  $R$  into  $X$  and  $Y$  is a lossless-join with relation to  $F$ , if and only if the closure of  $F$  contains  $X \cap Y \rightarrow X$  and  $X \cap Y \rightarrow Y$ . In particular, the decomposition of  $R$  into  $UV$  and  $R - V$  is lossless-join if  $U \rightarrow V$  holds over  $R$ .
  2. **Dependency Preservation:** It should be possible to check in the projected relations whether all the given FD's are satisfied.
- We **may** get a lossless-join decomposition with a BCNF decomposition.
- We **can** get both lossless-join and dependency preservation with 3NF.

## 2.6 Putting Relations into 3NF (Synthesis Algorithm)

- Prerequisite (Pre-Conditions):
  1. Split Right Sides
  2. Repeatedly try to remove an FD and see if the remaining FDs are equivalent to the original
  3. Repeatedly try to remove an attribute from a left side and see if the resulting FD's are equivalent to the original.
- We need a minimum basis for the FD's:
  1. One relation for each FD in the minimal basis.
    - a) Schema is the union of the left and right sides.
  - 2) If no key is contained in an FD, then add one relation whose schema is some key.