Chapter Four

Functional Dependencies and Normalization for Relational Databases

In this chapter we discuss some of the theory that has been developed in an attempt to choose "good" relation schemas—that is, to measure formally why one set of groupings of attributes into relation schemas is better than another. There are two levels at which we can discuss the "goodness" of relation schemas. The first is the **logical** (or **conceptual**) **level**—how users interpret the relation schemas and the meaning of their attributes. Having good relation schemas at this level enables users to understand clearly the meaning of the data in the relations, and hence to formulate their queries correctly. The second is the **implementation** (or **storage**) **level**—how the tuples in a base relation are stored and updated. This level applies only to schemas of base relations—which will be physically stored as files.

4.1. Informal Design Guidelines for Relation Schemas

We discuss four *informal measures* of quality for relation schema design in this section:

1. **Semantics of the attributes**

Whenever we group attributes to form a relation schema, we assume that a certain meaning is associated with the attributes. This meaning, or **semantics,** specifies how to interpret the attribute values stored in a tuple of the relation. Attributes of different entities should not be mixed in the same relation.

GUIDELINE 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, the meaning tends to be clear. Otherwise, the relation corresponds to a mixture of multiple entities and relationships and hence becomes semantically unclear.

2. **Reducing the redundant values in tuples**

One goal of schema design is to minimize the storage space that the base relations (files) occupy. Grouping attributes into relation schemas has a significant effect on storage space. Another serious problem with using the relations in table below as base relations is the problem of **update anomalies.** These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

EmpID	FName	LName	SkillID	Skill	SkillType	School	SchoolAdd	Skill Level
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
16	Lemma	Alemu	5	C++	Programming	Unity	Gerji	6
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8
94	Alem	Kebede	3	Cisco	Networking	AAU	Sidist_Kilo	7
18	Girma	Dereje	1	IP	Programming	Jimma	Jimma City	4
13	Yared	Gizaw	7	Java	Programming	AAU	Sidist_Kilo	6

Example of problems related with Anomalies

Deletion Anomalies:

If employee with ID 16is deleted then ever information about skill C++and the type of skill is deleted from the database. Then we will not have any information about C++and its skill type.

Insertion Anomalies:

What if we have a new employee with a skill called Pascal? We cannot decide weather Pascal is allowed as a value for skill and we have no clue about the type of skill that Pascal should be categorized as.

Modification Anomalies:

What if the address for Helico is changed from Piazza to Mexico? We need to look for every occurrence of Helico and change the value of School_Add from Piazza to Mexico, which is prone to error.

Based on the preceding three anomalies, we can state the guideline that follows.

GUIDELINE 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

3. Reducing the null values in tuples

In some schema designs we may group many attributes together into a "fat" relation. If many of the attributes do not apply to all tuples in the relation, we end up with many nulls in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes. Another problem with nulls is how to account for them when aggregate operations such as COUNT or SUM are applied. Moreover, nulls can have multiple interpretations, such as the following:

- The attribute *does not apply* to this tuple.
- The attribute value for this tuple is *unknown.*
- The value is *known but absent;* that is, it has not been recorded yet.

Having the same representation for all nulls compromises the different meanings they may have. Therefore, we may state another guideline.

GUIDELINE 3: As far as possible, avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

For example, if only 10 percent of employees have individual offices, there is little justification for including an attribute OFFICE_NUMBER in the EMPLOYEE relation; rather, a relation EMP_OFFICES (ESSN, OFFICE_NUMBER) can be created to include tuples for only the employees with individual offices.

4. **Disallowing the possibility of generating spurious tuples.**

Spurious tuples represent information that is not valid. Therefore avoid matching attributes that are not (foreign key, primary key) combination.

GUIDELINE 4: Design relation schemas so that they can be JOINed with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated. Do not have relations that contain matching attributes other than foreign key-primary key combinations. If such relations are unavoidable, do not join them on such attributes, because the join may produce spurious tuples.

4.2. Functional Dependency

A functional dependency is a constraint between two sets of attributes from the database. Two data items X and Y are said to be in a determinant or dependent relationship if certain values of data item Y always appears with certain values of data item X. if the data item X is the determinant data item and Y the dependent data item then the direction of the association is from X to Y and not vice versa.

A **functional dependency,** denoted by X→Y between two sets of attributes *X* and *Y* that are subsets of *R* specifies a *constraint* on the possible tuples that can form a relation state *r* of *R.* The constraint is that, for any two tuples and in r that have [*X*] = [*X*], we must also have [*Y*] = [*Y*]. This means that the values of the *Y* component of a tuple in *r* depend on, or are *determined by,*

the values of the *X* component; or alternatively, the values of the *X* component of a tuple uniquely (or **functionally**) **determine** the values of the *Y* component. We also say that there is a functional dependency from *X* to *Y* or that *Y* is **functionally dependent** on *X*.

E.g. Consider COMPANY database (see chapter two schema diagram of company)

a. SSN→ENAME

b. PNUMBER→ {PNAME, PLOCATION}

c. {SSN, PNUMBER} →HOURS

These functional dependencies specify that (a) the value of an employee's social security number (SSN) uniquely determines the employee name (ENAME), (b) the value of a project's number (PNUMBER) uniquely determines the project name (PNAME) and location (PLOCATION), and (c) a combination of SSN and PNUMBER values uniquely determines the number of hours the employee works on the project per week (HOURS). Alternatively, we say that ENAME is functionally determined by (or functionally dependent on) SSN, or "given a value of SSN, we know the value of ENAME," and so on.

Partial Dependency

If an attribute which is not a member of the primary key is dependent on some part of the primary key (if we have composite primary key) then that attribute is partially functionally dependent on the primary key.

Let {A, B} is the Primary Key and C is no key attribute.

Then if {A, B}➔C and B➔C

Then C is partially functionally dependent on {A, B}

e.g. {SSN, DNUMBER}➔ENAME, ENAME is partially functionally dependent since SSN alone can determine ENAME. Which means SSN➔ENAME hold

Full Dependency

If an attribute which is not a member of the primary key is not dependent on some part of the primary key but the whole key (if we have composite primary key) then that attribute is fully functionally dependent on the primary key.

Let {A, B} is the Primary Key and C is no key attribute

Then if {A, B}➔C and B➔C and A➔C does not hold

Then C Fully functionally dependent on {A, B}

e.g. {SSN, PNUMBER}➔Hours that means either SSN➔Hours or PNUMBER➔Hours , doesn't hold

Transitive Dependency

In mathematics and logic, a transitive relationship is a relationship of the following form:

"If A implies B, and if also B implies C, then A implies C."

Generalized way of describing transitive dependency is that:

If A functionally governs B, AND

If B functionally governs C

THENA functionally governs C

4.3. Normalization

NORMALIZATION is the process of identifying the logical associations between data items and designing a database that will represent such associations but without suffering the update anomalies which are;

1. Insertion Anomalies
2. Deletion Anomalies
3. Modification Anomalies

Normalization is formal technique for analyzing a relation based on its:

- ✓ Primary key and
- ✓ The functional dependencies between the attributes of that relation

A relation can be normalized to a specific form to prevent the possible occurrence of update anomalies. We will focus on the first three normal forms for relation schemas and the intuition behind them.

First Normal Form

- Disallow *multivalued* attributes, *composite* attributes, and their combinations.
- Domain of attribute must include only atomic and single value attribute
- The only attribute values permitted by 1NF are single atomic (or indivisible) values.

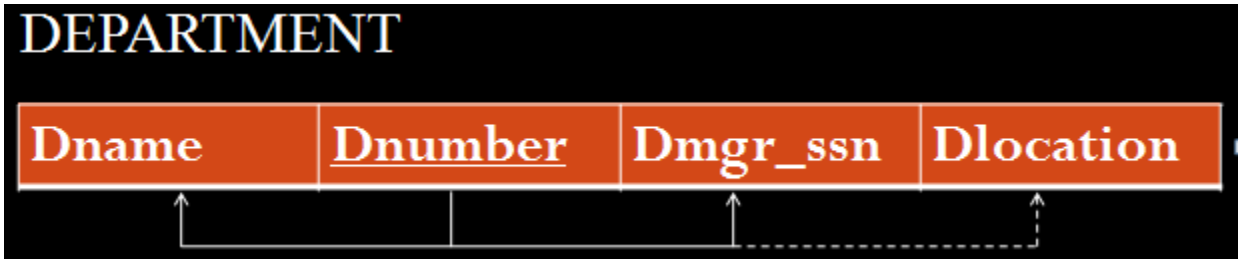

Figure 4.1: Department table with multivalued attribute

Consider the DEPARTMENT relation schema whose primary key is DNUMBER, and suppose that we extend it by including the DLOCATIONS attribute as shown in Figure. We assume that each department can have *a number of* locations. As we can see, this is not in 1NF because DLOCATIONS is not an atomic attribute. There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute DLOCATIONS that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT. The primary key of this relation is the combination {DNUMBER, DLOCATION}. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination {DNUMBER, DLOCATION}. This solution has the disadvantage of introducing *redundancy* in the relation.
3. If a *maximum number of values* is known for the attribute—for example, if it is known that *at most three locations* can exist for a department—replace the DLOCATIONS attribute by three atomic attributes: DLOCATION1, DLOCATION2, and DLOCATION3. This solution has the disadvantage of introducing *null values* if most departments have fewer than three locations.

Of the three solutions above, the first is superior.

Second Normal Form

- Is based on the concept of *full functional dependency.*
- A relation R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all.

Solution:

Finally decompose and set up a new relation for full functionally dependences.

Example: Normalize the following table in to second normal form (it is already in first normal form)

ename	SSN	sex	Dname	Dnum	DMGRSSN
Sami	11	M	IT	1	66
Abel	22	M	CS	2	22
John	33	M	IT	1	66
Fasika	44	F	IS	3	44
Jemal	55	M	CS	2	22
Roza	66	F	IT	1	66

Solution: Given SSN and Dnum as a primary key So, let test by making the primary key to the left hand side and the non-key to the right hand side {SSN, Dnum}➔ ename: is not full functional dependency because SSN➔ename, so that take this full functional dependency alone in one relation. In this way test all and finally we will gate below table.

2 NF

Dname	Dnum	DMGRSSN
Research	1	E16
Administration	2	E24
Staff	5	E28

ename	SSN	Dno.
Sami	11	1
Abel	22	2
John	33	1
Fasika	44	3
Jemal	55	2
Roza	66	1

Third Normal Form

- Is based on the concept of *transitive dependency.*
- Relation should not have a non-key attribute functionally determined by another non-key attribute

Solution:

- Decompose and set up a relation that includes the non key attribute(s) that functionally determine(s) other non key attribute(s)

E.g. The dependency SSN→DMGRSSN is transitive through DNUMBER in EMP_DEPT because both the dependencies SSN→DNUMBER and DNUMBER→DMGRSSN hold *and* DNUMBER is neither a key itself nor a subset of the key of EMP_DEPT. Intuitively; we can see that the dependency of DMGRSSN on DNUMBER is undesirable in EMP_DEPT since DNUMBER is not a key of EMP_DEPT.

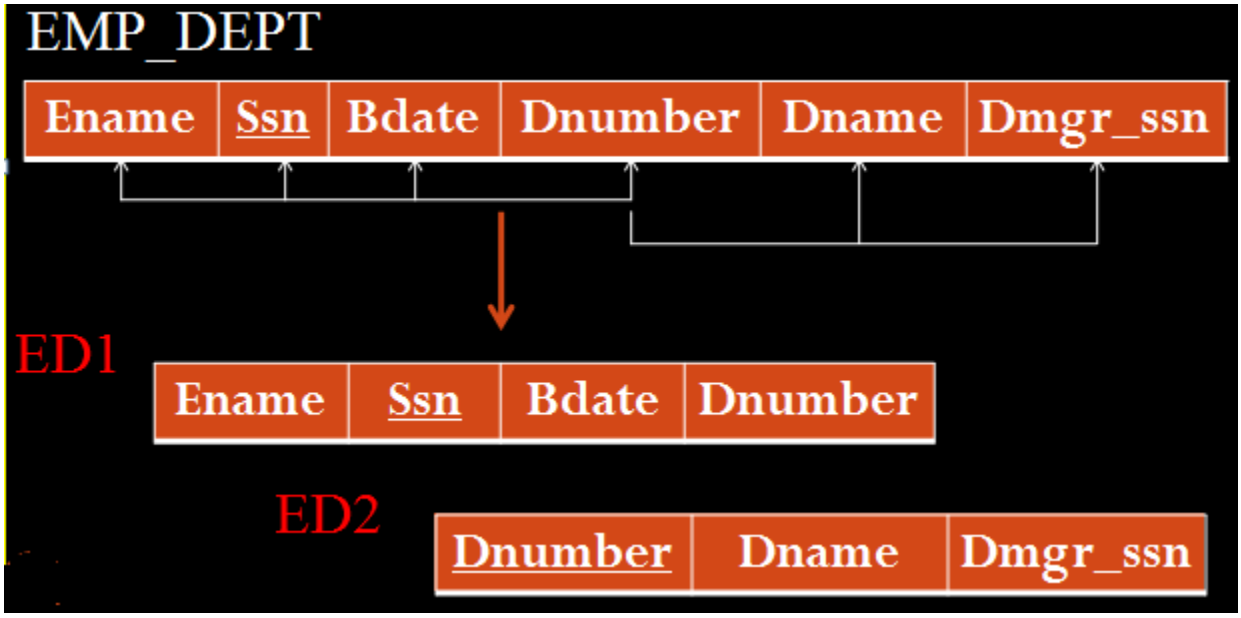

Figure 4.3: EMP_DEPT table