Project-Database Normalization

Felipe Moreno

Facultad de Ciencias (FC)

Universidad Nacional de Ingeniería (UNI)

Introduction

We will discuss in this project about Informal Design Guidelines for Relation Schemas So That the Attributes is Semantics, Reducing the redundant values in tuples, Reducing the null values in tuples and Disallowing spurious tuples.
And what is meaning Functional dependency and how can export this function from the relation so this is very important to apply the normalization. and what is the normalization and Back ground to normalization that must be you know and how to apply the normalization on the relation Through.
1-First Normal Form
2- Second Normal Form
3-Third Normal Form
And What are the conditions that must be in the relation to say this relation in First Normal Form or Second Normal Form or Third Normal Form.
what is different between 1NF, 2NF, 3NF
and we will discuss some of algorithms Through them you can but the relation in 1NF, 2NF, 3NF and you can Through them make test on the relation to determine this relation in 1NF, 2NF, 3NF or no.

:Objectives

- Build generalizable data normalization pipeline
- Semantic normalization annotators
- Establish a globally available resource for value sets
- Establish and expand modular library of normalization algorithms

- Consistent and standardized common model to support large-scale vocabulary use and adoption
- Support mapping into canonical value sets
- Normalize the data against HR.
- Normalize retrospective data and compare it to normalized data that already exists in our data warehouses.

relation database design is the grouping of attributes to form "good " relation schemas. two levels of relation schemas
- the logical "user view" level
- the storage "base relation" level

We first discuss formal guidelines for good relation design then we discuss formal concepts of functional dependencies and normal forms.
- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
-BCNF (Boyce-Codd Normal Form)

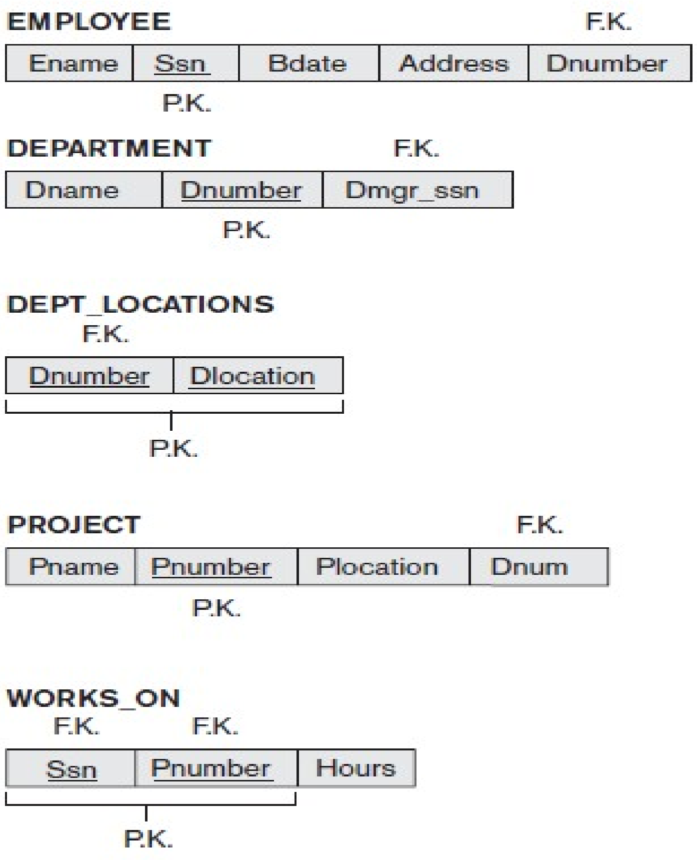
1- Informal Design Guidelines for Relation Schemas

Before discussing the formal theory of relational database design, we discuss four informal guidelines that may be used as measures to determine the quality of relation schema design:
- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

These measures are not always independent of one another, as we will see

1.1 Imparting Clear Semantics to Attributes in Relations

Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them. The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.



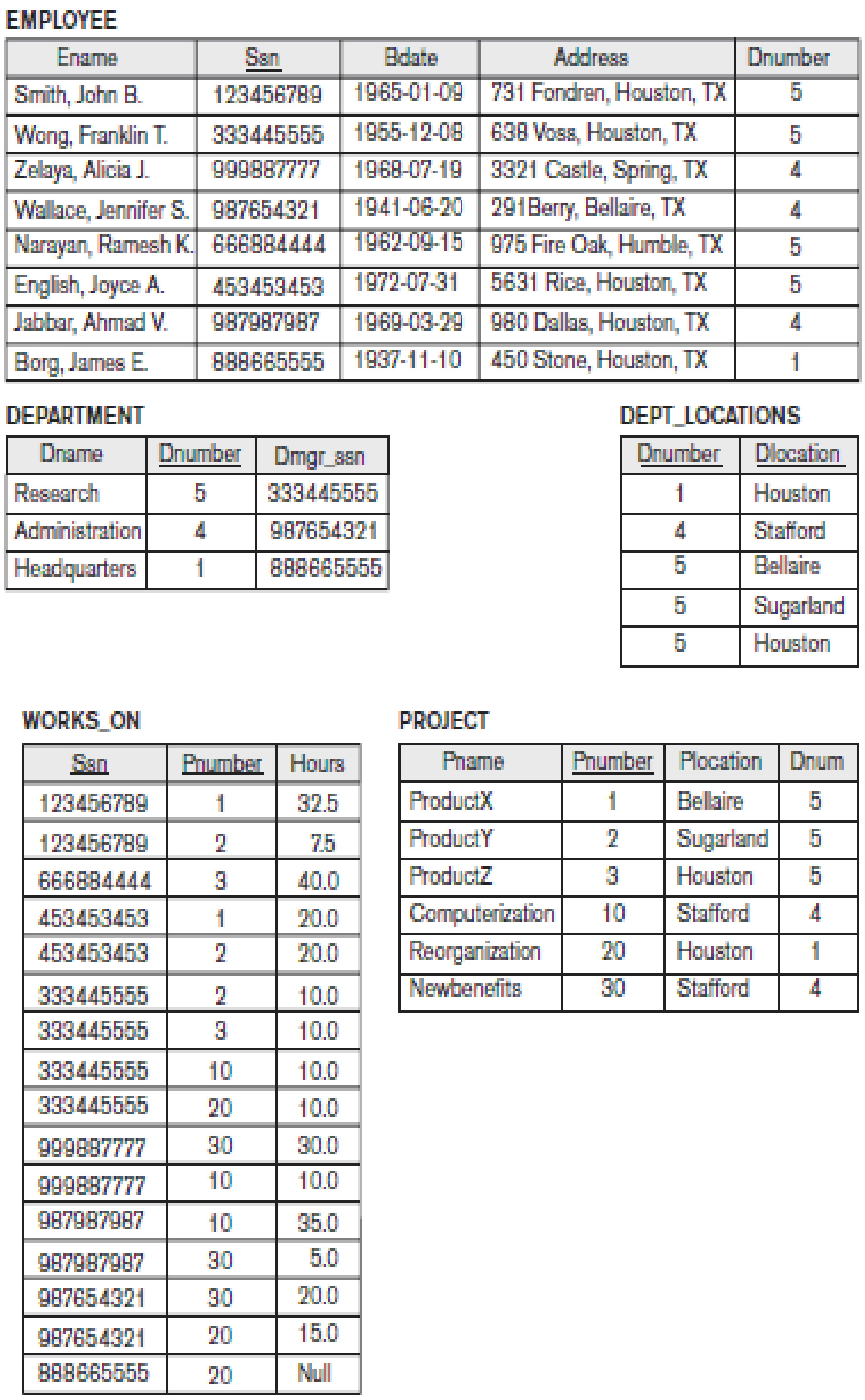
((Figure 1.1

in Figure 1.1 The meaning of the EMPLOYEE relation schema is quite simple: Each tuple represents an employee, with values for the employee's name (Ename), Social Security number (Ssn), birth date (Bdate), and address (Address), and the number of the department that the employee works for (Dnumber). The Dnumber attribute is a foreign key that represents an implicit relationship between EMPLOYEE and DEPARTMENT.

The semantics of the DEPARTMENT and PROJECT schemas are also straightforward Each DEPARTMENT tuple represents a department entity, and each PROJECT tuple represents a project entity. The attribute Dmgr_ssn of DEPARTMENT relates a department to the employee who is its manager, while Dnum of PROJECT relates a project to its controlling department; both are foreign key attributes. The ease with which the meaning of a relation's attributes can be explained is an informal measure of how well the relation is designed.

1.2 Redundant Information in Tuples and Update Anomalies

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has schemas has a significant effect on storage space. For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in Figure 15.2 with that for an EMP_DEPT base relation in Figure 15.4, which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT. In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department. In contrast, each department's information appears only once in the DEPARTMENT relation in Figure 15.2. Only the department number (Dnumber) is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key. Similar comments apply to the EMP_PROJ relation (see Figure 15.4), which augments the WORKS_ON relation with additional attributes from EMPLOYEE and PROJECT.



EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

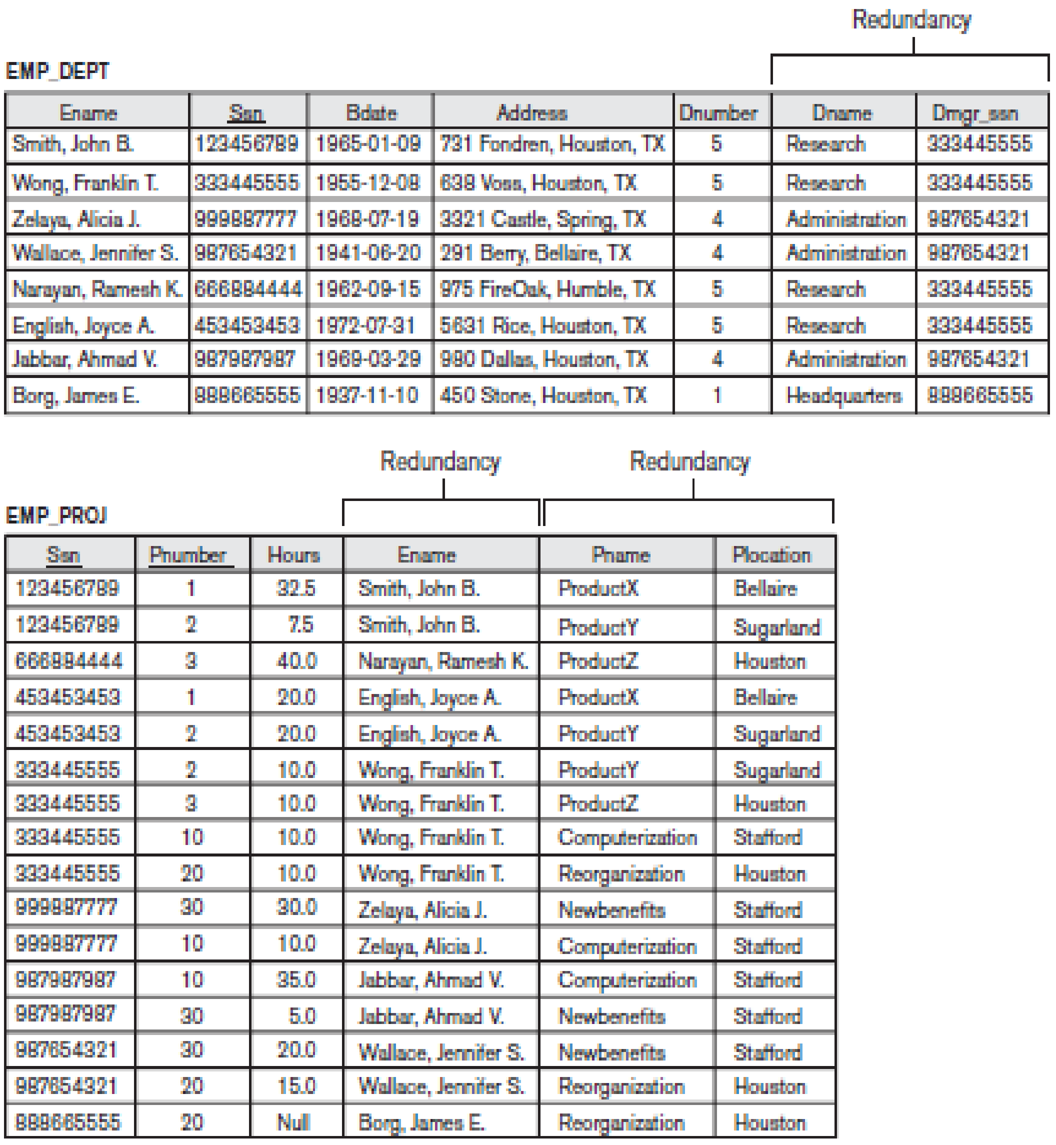
WORKS_ON

Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

(Figure 1.2)


Redundancy
Redundancy
Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

(Figure 1.3)

Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for

a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT. In the design of Figure 15.2, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.
- It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the

Deletion Anomalies. The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database. This problem does not occur in the database of Figure 15.2 because DEPARTMENT tuples are stored separately.

Modification Anomalies. In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong.

1.3 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 and with specifying JOIN operations at the logical level. Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied.
SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable. Moreover, NULLs can have multiple interpretations,

such as the following:
- The attribute does not apply to this tuple. For example, Visa status may not apply to U.S. students.
- The attribute value for this tuple is unknown. For example, the Date_of_birth may be unknown for an employee.
- The value is known but absent; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

2- Functional Dependencies

So far we have dealt with the informal measures of database design. We now introduce a formal tool for analysis of relational schemas that enables us to detect and describe some of the above-mentioned problems in precise terms. The single most important concept in relational schema design theory is that of a functional dependency.

2.1 Definition of Functional Dependency

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, ..., An; let us think of the whole database as being described by a single universal relation schema R = {A1, A2, ..., An}. We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies.

Definition. 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 t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[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; 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. The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X value, they must necessarily agree on their Y value.

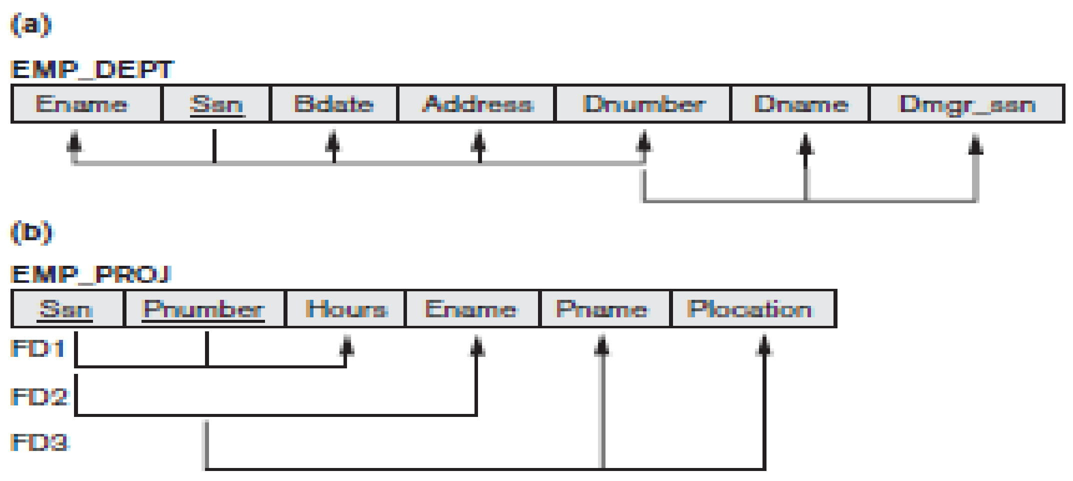
Note the following:

-If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R) that is, X is a candidate key of R this implies that X --> Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X). If X is a candidate key of R, then X--> R.

- If X--> Y in R, this does not say whether or not Y--> X in R.

A functional dependency is a property of the semantics or meaning of the attributes.

The database designers will use their understanding of the semantics of the attributes of R that is, how they relate to one another—to specify the functional dependencies that should hold on all relation states (extensions) r of R. Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint. Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R. Hence, the main use of functional dependencies is to describe further a relation schema R by specifying constraints on its attributes that must hold at all times. Certain FDs can be specified without referring to a specific relation, but as a property of those attributes given their commonly understood meaning. For example, {State, Driver_license_number} --> Ssn should hold for any adult in the United States and hence should hold whenever these attributes appear in a relation. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes. For example, the FD Zip_code --> Area_code used to exist as a relationship between postal codes and telephone number--> codes in the United States, but with the proliferation of telephone area codes it is no longer true.



(Figure 2.1)

Consider the relation schema EMP_PROJ in Figure 2.1(b); from the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

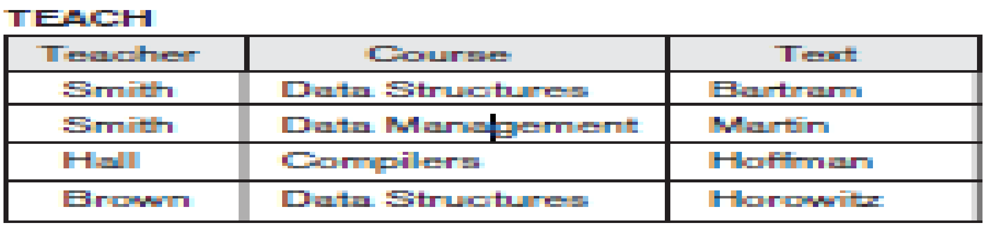
a. Ssn--> Ename

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 currently 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.

A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R. Therefore, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R. For example, Figure 15.7 shows a particular state of the TEACH relation schema. Although at first glance we may think that Text Course, we cannot confirm this unless we know that it is true for all possible legal states of TEACH. It is, however, sufficient to demonstrate a single counterexample to disprove a functional dependency. For example, because 'Smith' teaches both 'Data Structures' and 'Data Management,' we can conclude that Teacher does not functionally determine Course.



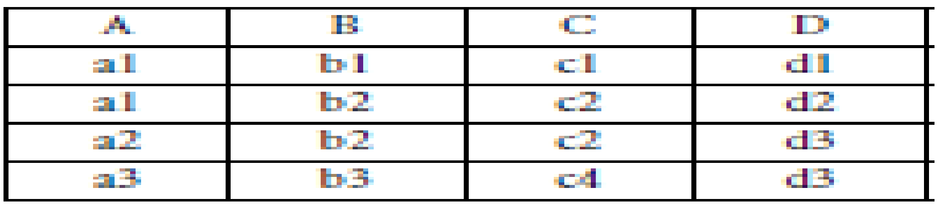
TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

(Figure 2.2)

Given a populated relation, one cannot determine which FDs hold and which do not unless the meaning of and the relationships among the attributes are known. All one can say is that a certain FD may exist if it holds in that particular extension. One cannot guarantee its existence until the meaning of the corresponding attributes is clearly understood. One can, however, emphatically state that a certain FD does not hold if there are tuples that show the violation of such an FD.

See the illustrative example relation in Figure 15.8. Here, the following FDs may hold because the four tuples in the current extension have no violation of these constraints:
B --> C; C --> B; {A, B} --> C; {A, B} --> D; and {C, D} --> B. However, the following do not hold because we already have violations of them in the given extension: A --> B
(tuples 1 and 2 violate this constraint); B--> A (tuples 2 and 3 violate this constraint); D--> C (tuples 3 and 4 violate it).



A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

(Figure 2.3)

3- Database normalization

3-1Back ground to normalization

Functional dependency

In a given table, an attribute Y is said to have a functional dependency on a set of attributes X (written X → Y) if and only if each X value is associated with precisely one Y value. For example , in an "Employee" table that includes the attributes "Employee ID" and "Employee Date of Birth", the functional dependency {Employee ID} → {Employee Date of Birth} would hold. It follows from the previous two sentences that each {Employee ID} is associated with precisely one {Employee Date of Birth}.

Full functional dependency

An attribute is fully functionally dependent on a set of attributes X if it is:

functionally dependent on X, and not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a full functional dependency, because it is also dependent on {Employee ID}. Even by the removal of {Skill} functional dependency still holds between {Employee Address} and {Employee ID}.

Transitive dependency

A transitive dependency is an indirect functional dependency, one in which X→Z only by virtue of X-->Y and Y-->Z.

Trivial functional dependency

A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} --> {Employee Address} is trivial, as is {Employee Address} --> {Employee Address}.

Multivalued dependency

A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

Join dependency

A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T.

Superkey

A superkey is a combination of attributes that can be used to uniquely identify a database record. A table might have many superkeys.

Candidate key

A candidate key is a special subset of superkeys that do not have any extraneous information in them: it is a minimal superkey.

Example:

A table with the fields <Name>, <Age>, <SSN> and <Phone Extension> has many possible superkeys. Three of these are <SSN>, <Phone Extension, Name> and <SSN, Name>. Of those, only <SSN> is a candidate key as the others contain information not necessary to uniquely identify records ('SSN' here refers to Social Security Number, which is unique to each person).

Non-prime attribute

A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.

Prime attribute

A prime attribute, conversely, is an attribute that does occur in some candidate key.

Primary key

One candidate key in a relation may be designated the primary key. While that may be a common practice (or even a required one in some environments), it is strictly notational and has no bearing on normalization. With respect to normalization, all candidate keys have equal standing and are treated the same.

3.2 Normalization of Relations

Definition: -

is the process of organizing the fields and tables of a relational database to minimize redundancy. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as relational design by analysis. Initially, Codd proposed three
normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies

process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions the normal form tests are decomposed into smaller relation schemas that meet the tests and hence possess the
desirable properties. Thus, the normalization procedure provides database designers
with the following:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree normal form

Definition

- The normal forms (NF) of relational database theory provide criteria for determining a table's degree of immunity against logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is. Each table has a "highest normal form" (HNF): by definition, a table always meets the requirements of its HNF and of all normal forms lower than its HNF; also by definition, a table fails to meet the requirements of any normal form higher than its HNF.
- The normal forms are applicable to individual tables; to say that an entire database is in normal form n is to say that all of its tables are in normal form n.
- Newcomers to database design sometimes suppose that normalization proceeds in an iterative fashion, i.e. a 1NF design is first normalized to 2NF, then to 3NF, and so on. This is not an accurate description of how normalization typically works. A sensibly designed table is likely to be in 3NF on the first attempt; furthermore, if it is 3NF, it is overwhelmingly likely to have an HNF of 5NF. Achieving the "higher" normal forms (above 3NF) does not usually

require an extra expenditure of effort on the part of the designer, because 3NF tables usually need no modification to meet the requirements of these higher normal forms.

- Normal forms, when considered in isolation from other factors, do not guarantee a
- good database design. It is generally not sufficient to check separately that each
- relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional
- properties that the relational schemas, taken together, should possess. These would
- include two properties:

- nonadditive join or lossless join property
- dependency preservation property

3-3 First Normal Form (1NF)

First normal form (1NF) sets the fundamental rules for database normalization and relates to a single table within a relational database system. Normalization follows three basic steps, each building on the last. The first of these is the first normal form.

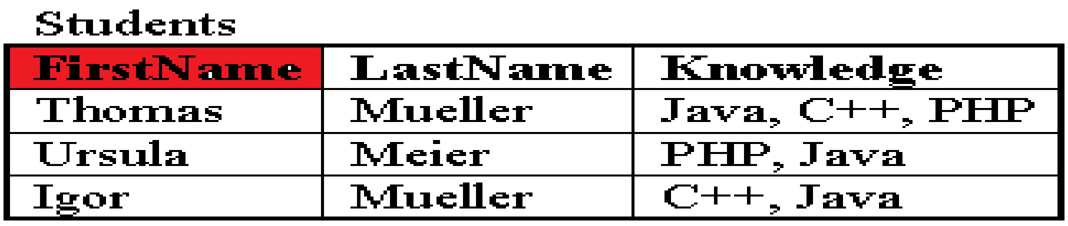
The first normal form states that:
- Every column in the table must be unique
- Separate tables must be created for each set of related data
- Each table must be identified with a unique column or concatenated columns called the primary key
- No rows may be duplicated
- no columns may be duplicated
- no row/column intersections contain a null value
- no row/column intersections contain multivalued fields

disallow multivalued attributes, composite attributes, and their combinations.

It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only
attribute values permitted by 1NF are single atomic (or indivisible) values.

First normal form example: -

This example not in 1NF because the attribute Knowledge have multivalued Not in First normal form befor normalization

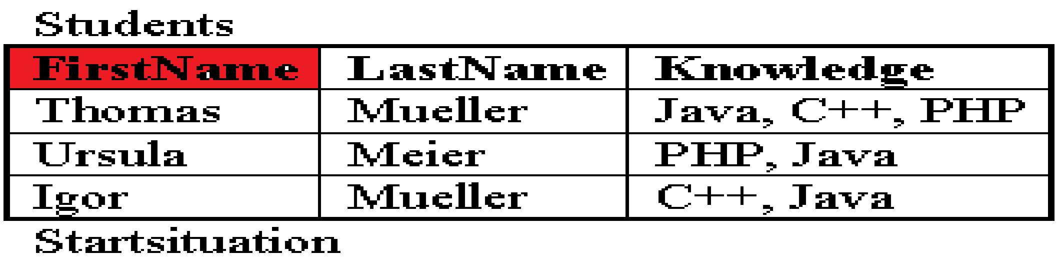


Students

FirstName	LastName	Knowledge
Thomas	Mueller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Mueller	C++, Java

Answer
In this example Remove the attribute Knowledge that violates 1NF and place it in a separate relation Students along with the primary key FirstName of Students. The primary key of this relation is the combination {FirstName, Knowledge}, this in 1NF

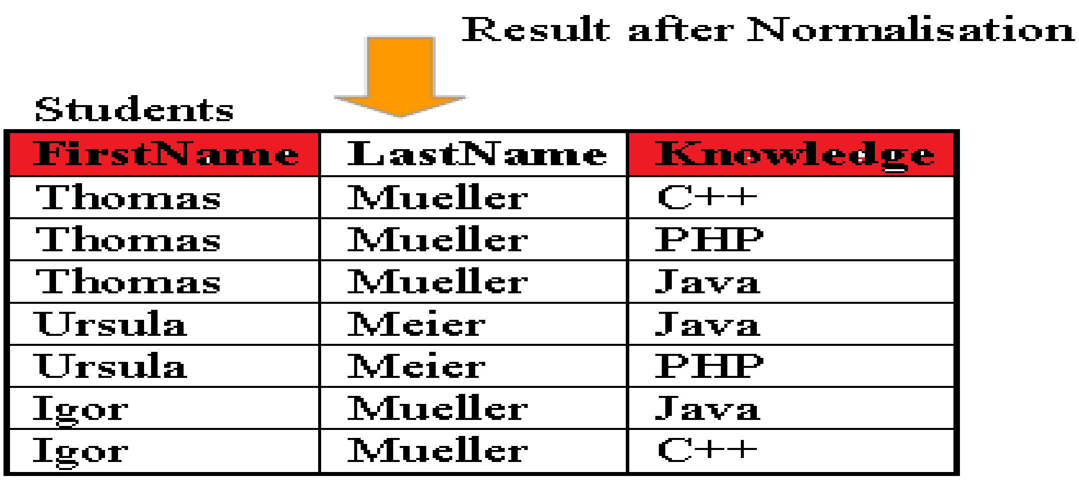
in First normal form after normalization



Students

FirstName	LastName	Knowledge
Thomas	Mueller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Mueller	C++, Java

Startsituation



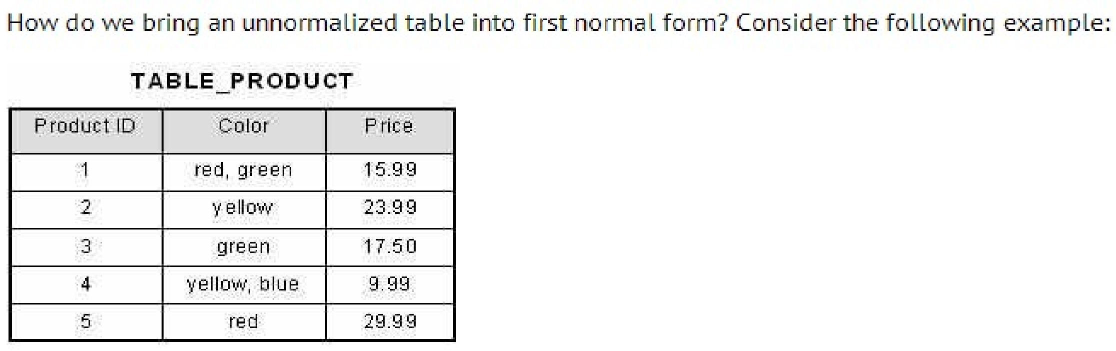
Result after Normalisation

Students

FirstName	LastName	Knowledge
Thomas	Mueller	C++
Thomas	Mueller	PHP
Thomas	Mueller	Java
Ursula	Meier	Java
Ursula	Meier	PHP
Igor	Mueller	Java
Igor	Mueller	C++

In this examble Remove the attribute Knowledge that violates 1NF and place it in a separate relation Students along with the primary key FirstName of Students. The primary key of this relation is the combination {FirstName, Knowledge}, this in 1NF

Another first normal form example: -

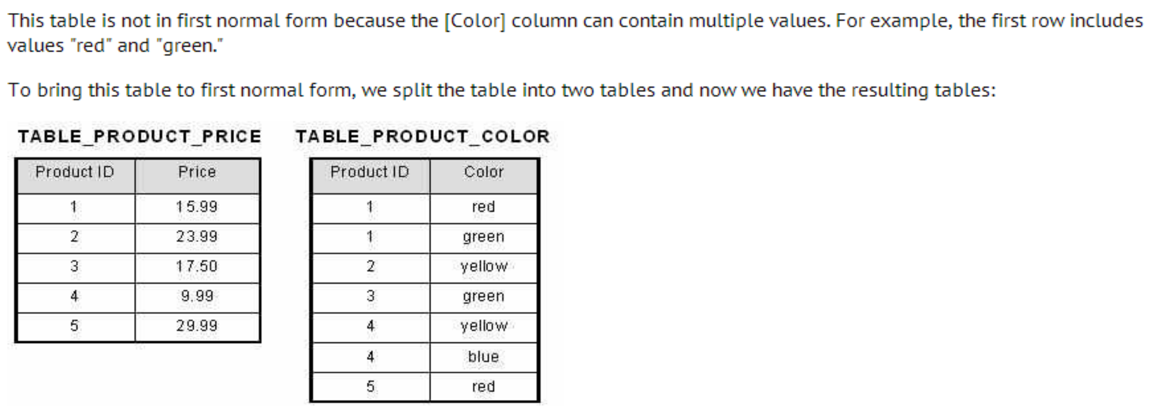


How do we bring an unnormalized table into first normal form? Consider the following example:

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

Answer



This table is not in first normal form because the [Color] column can contain multiple values. For example, the first row includes values "red" and "green."

To bring this table to first normal form, we split the table into two tables and now we have the resulting tables:

TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

3.4 Second Normal Form: -

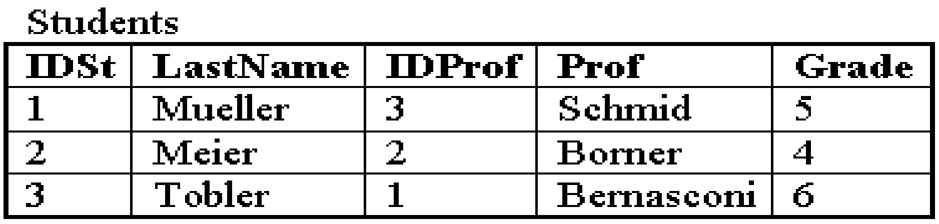
Definition: -

A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

-Second normal form (2NF) is the second step in normalizing a database. 2NF builds on the first normal form (1NF).
-A 1NF table is in 2NF form if and only if all of its non-prime attributes are functionally dependent on the whole of every candidate key.
-Second normal form (2NF) is based on the concept of full functional dependency functionally dependent on X, and not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a full functional dependency, because it is also dependent on {Employee ID}. Even by the removal of {Skill} functional dependency still holds between {Employee Address} and {Employee ID}.

Second normal form example: -

Student(IDSt, StudentName, IDProf, ProfessorName, Grade) The attributes IDSt and IDProf are the identification keys. All attributes a single valued (1NF).

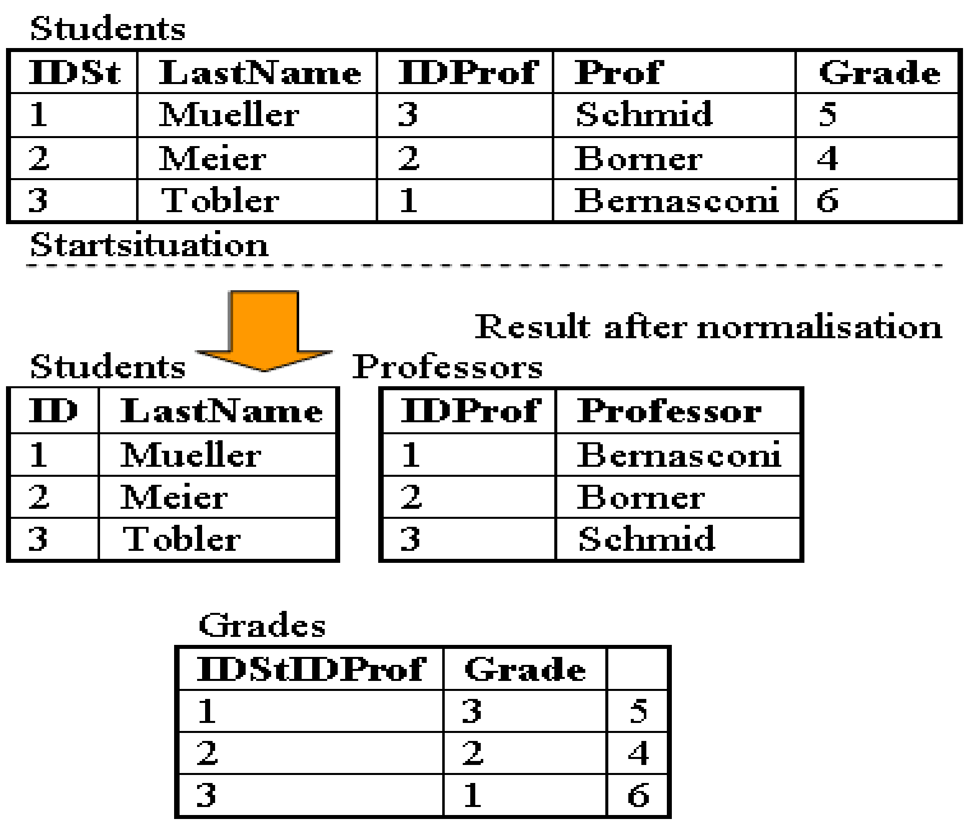


Students

IDSt	LastName	IDProf	Prof	Grade
1	Mueller	3	Schmid	5
2	Meier	2	Borner	4
3	Tobler	1	Bernasconi	6

The following functional dependencies exist:
1. The attribute ProfessorName is functionally dependent on attribute IDProf (IDProf --> ProfessorName)
2. The attribute StudentName is functionally dependent on IDSt (IDSt --> StudentName)
3. The attribute Grade is fully functional dependent on IDSt and IDProf (IDSt, IDProf --> Grade)

Answer



Students

IDSt	LastName	IDProf	Prof	Grade
1	Mueller	3	Schmid	5
2	Meier	2	Borner	4
3	Tobler	1	Bernasconi	6

Startsituation

Result after normalisation

Students

ID	LastName
1	Mueller
2	Meier
3	Tobler

Professors

IDProf	Professor
1	Bernasconi
2	Borner
3	Schmid

Grades

IDStIDProf	Grade	
1	3	5
2	2	4
3	1	6

Example Second normal form

The table in this example is in first normal form (1NF) since all attributes are single valued. But it is not yet in 2NF. If student 1 leaves university and the tuple is deleted, then we lose all information about professor Schmid, since this attribute is fully functional dependent on the primary key IDSt. To solve this problem, we must create a new table Professor with the attribute Professor (the name) and the key IDProf. The third table Grade is necessary for combining the two relations Student and Professor and to manage the grades. Besides the grade it contains only the two IDs of the student and the professor. If now a student is deleted, we do not lose the information about the professor

Another second normal form example: -

FIRST (supplier_no, status, city, part_no, quantity)
Functional Dependencies:

(supplier_no, part_no) → quantity

(supplier_no) → status

(supplier_no) → city

city → status (Supplier's status is determined by location)
Answer

Comments:

Non-key attributes are not mutually independent (city → status).
Non-key attributes are not fully functionally dependent on the primary key (i.e., status and city are dependent on just part of the key, namely supplier_no).

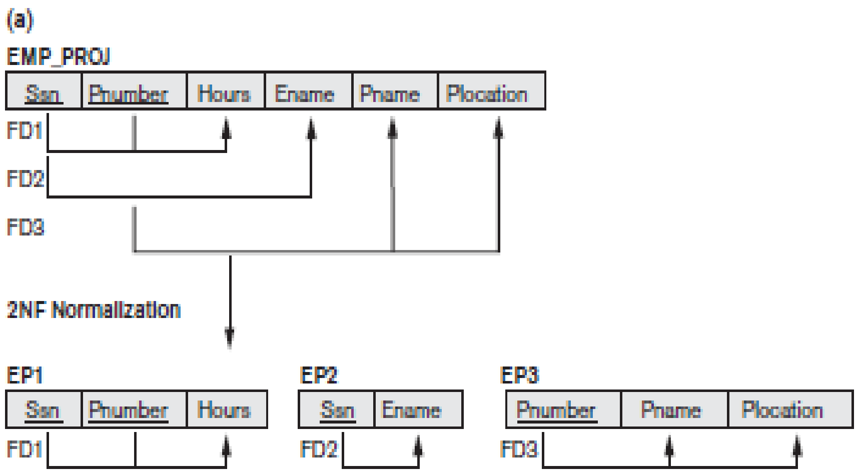
Anomalies:

INSERT: We cannot enter the fact that a given supplier is located in a given city until that supplier supplies at least one part (otherwise, we would have to enter a null value for a column participating in the primary key C a violation of the definition of a relation).
DELETE: If we delete the last (only) row for a given supplier, we lose the information that the supplier is located in a particular city.
UPDATE: The city value appears many times for the same supplier. This can lead to inconsistency or the need to change many values of city if a supplier moves.

Decomposition (into 2NF):

SECOND (supplier_no, status, city)
SUPPLIER_PART (supplier_no, part_no, quantity)

Another second normal form example: -



3.5 Third Normal Form: -

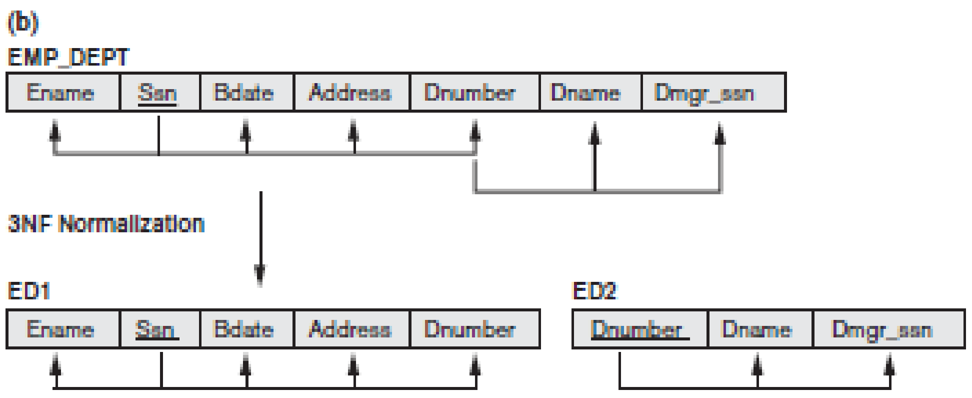
based on the concept of transitive dependency

Definition: -

According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

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. [(A --> B) and (B --> C)] --> (A --> C)

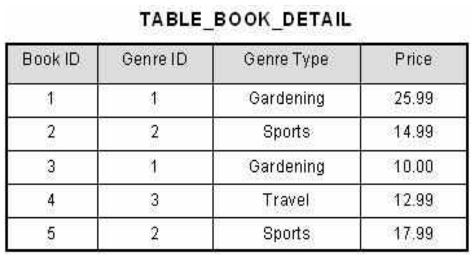
Third normal form example: -



Another Third normal form example: -

This relation in (2NF but not 3NF):
SECOND (supplier_no, status, city)
Functional Dependencies:
supplier_no → status
supplier_no → city
city → status
Answer
Comments:
Lacks mutual independence among non-key attributes.
Mutual dependence is reflected in the transitive dependencies:
supplier_no → city, city → status.

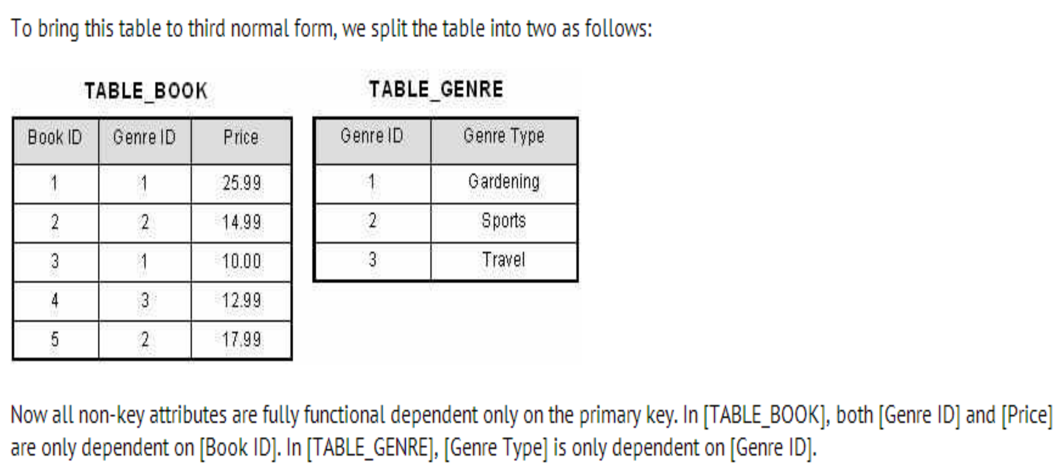
Anomalies:
INSERT: We cannot record that a particular city has a particular status until we have a supplier in that city.
DELETE: If we delete a supplier which happens to be the last row for a given city value, we lose the fact that the city has the given status.
UPDATE: The status for a given city occurs many times, therefore leading to multiple updates and possible loss of consistency.
Decomposition (into 3NF):
SUPPLIER_CITY (supplier_no, city)
CITY_STATUS (city, status)
Another Third normal form example: -


TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

Answer


To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

3.6 Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is one of the forms of database normalization. A database table is in BCNF if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key.

Definition: -
A relation schema R is in BCNF if whenever a nontrivial functional dependency X-->A holds in R, then X is a superkey of R.

Boyce-Codd normal form example: -

This relation in (3NF but not BCNF):
SUPPLIER_PART (supplier_no, supplier_name, part_no, quantity)
Functional Dependencies:
We assume that supplier_name's are always unique to each supplier. Thus we have two candidate keys:
(supplier_no, part_no) and (supplier_name, part_no)
Thus we have the following dependencies:

(supplier_no, part_no) → quantity

(supplier_no, part_no) → supplier_name

(supplier_name, part_no) → quantity

(supplier_name, part_no) → supplier_no

supplier_name → supplier_no

supplier_no → supplier_name

Comments:

Although supplier_name ☰ supplier_no (and vice versa), supplier_no is not a non-key column — it is part of the primary key! Hence this relation technically satisfies the definition(s) of 3NF (and likewise 2NF, again because supplier_no is not a non-key column).

Anomalies:

INSERT: We cannot record the name of a supplier until that supplier supplies at least one part.

DELETE: If a supplier temporarily stops supplying and we delete the last row for that supplier, we lose the supplier's name.

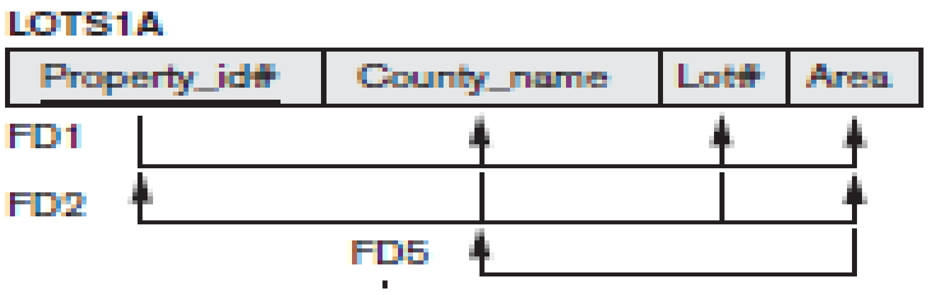
UPDATE: If a supplier changes name, that change will have to be made to multiple rows (wasting resources and risking loss of consistency).

Decomposition (into BCNF):

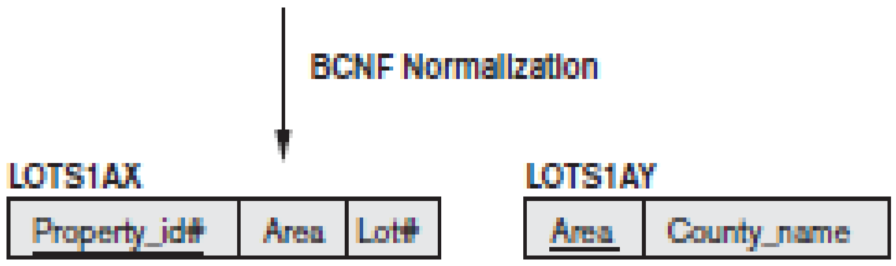
SUPPLIER_ID (supplier_no, supplier_name)

SUPPLIER_PARTS (supplier_no, part_no, quantity)

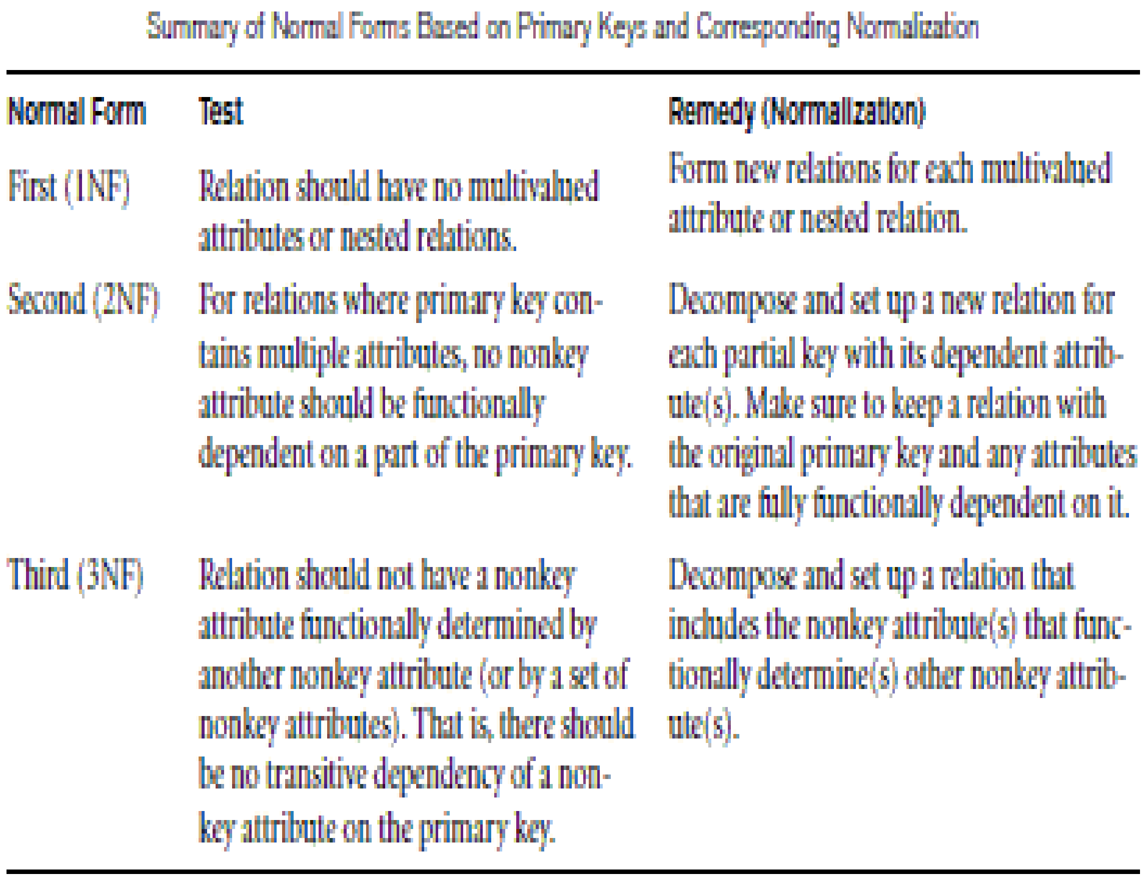
Another Boyce-Codd normal form example: -



Answer



Summary of Normal Forms Based on Primary Keys and Corresponding Normalization


Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

4- Inference Rules for Functional Dependencies

IR1 (reflective): If X (union) Y, then X->Y

IR2 (augmentation rule): {X->Y} |= XZ -> YZ

IR3 (transitive rule): {X->Y, Y->Z} |= X->Z

IR4 (decomposition): {X-> YZ} |= X->Y

IR5 (union): {X->Y, X->Z} |= X->YZ

IR6 (pseudo transitive): {X->Y, WY->Z} |= WX->Z

Proof of IR4 (Using IR1 through IR3).

1. X -->YZ (given).

2. YZ -->Y (using IR1 and knowing that YZ -->Y).

3. X -->Y (using IR3 on 1 and 2).

Proof of IR5 (using IR1 through IR3).

1. X -->Y (given).

2. X -->Z (given).

3. X -->XY (using IR2 on 1 by augmenting with X; notice that XX = X).

4. XY -->YZ (using IR2 on 2 by augmenting with Y).

5. X -->YZ (using IR3 on 3 and 4).

Proof of IR6 (using IR1 through IR3).

1. X -->Y (given).

2. WY -->Z (given).

3. WX -->WY (using IR2 on 1 by augmenting with W).

4. WX -->Z (using IR3 on 3 and 2).

5- Algorithms

Algorithm 1.
Determining X+, the Closure of X under F
Input: A set F of FDs on a relation schema R, and a set of attributes X, which is
a subset of R.

```
X+ := X;
repeat
oldX+ := X+;
for each functional dependency Y-->Z in F do
if X+ --> Y then X+ := X+ ∪ Z;
until (X+ = oldX+);
```

Algorithm 2.
Finding a Minimal Cover F for a Set of Functional
Dependencies E
Input: A set of functional dependencies E.

1. Set F := E.

2. Replace each functional dependency X-->{A1, A2, ..., An} in F by the n functional
dependencies X-->A1, X-->A2, ..., X-->An.

3. For each functional dependency X-->A in F
for each attribute B that is an element of X
if { {F – {X-->A} } ∪ { (X – {B}) -->A} } is equivalent to F
then replace X-->A with (X – {B}) -->A in F.

4. For each remaining functional dependency X-->A in F
if {F – {X-->A} } is equivalent to F,
then remove X-->A from F.

Algorithm 3.
Finding a Key K for R Given a set F of Functional
Dependencies
Input: A relation R and a set of functional dependencies F on the attributes of
R.

1. Set K := R.

2. For each attribute A in K
{compute (K – A)+ with respect to F;
if (K – A)+ contains all the attributes in R, then set K := K – {A} };

Algorithm 4
Testing for Nonadditive Join Property
Input: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a
set F of functional dependencies.
Note: Explanatory comments are given at the end of some of the steps. They follow
the format: (* comment *).
1. Create an initial matrix S with one row i for each relation Ri in D, and one
column j for each attribute Aj in R.
2. Set S(i, j):= bij for all matrix entries. (* each bij is a distinct symbol associated
with indices (i, j) *).
3. For each row i representing relation schema Ri
{for each column j representing attribute Aj
{if (relation Ri includes attribute Aj) then set S(i, j):= aj;};}; (* each aj is a
distinct symbol associated with index (j) *).
4. Repeat the following loop until a complete loop execution results in no
changes to S
{for each functional dependency X-->Y in F
{for all rows in S that have the same symbols in the columns corresponding
to attributes in X
{make the symbols in each column that correspond to an attribute in Y
be the same in all these rows as follows: If any of the rows has an a symbol
for the column, set the other rows to that same a symbol in the column.
If no a symbol exists for the attribute in any of the rows, choose
one of the b symbols that appears in one of the rows for the attribute
and set the other rows to that same b symbol in the column ;} ; } ;};
5. If a row is made up entirely of a symbols, then the decomposition has the

nonadditive join property; otherwise, it does not.

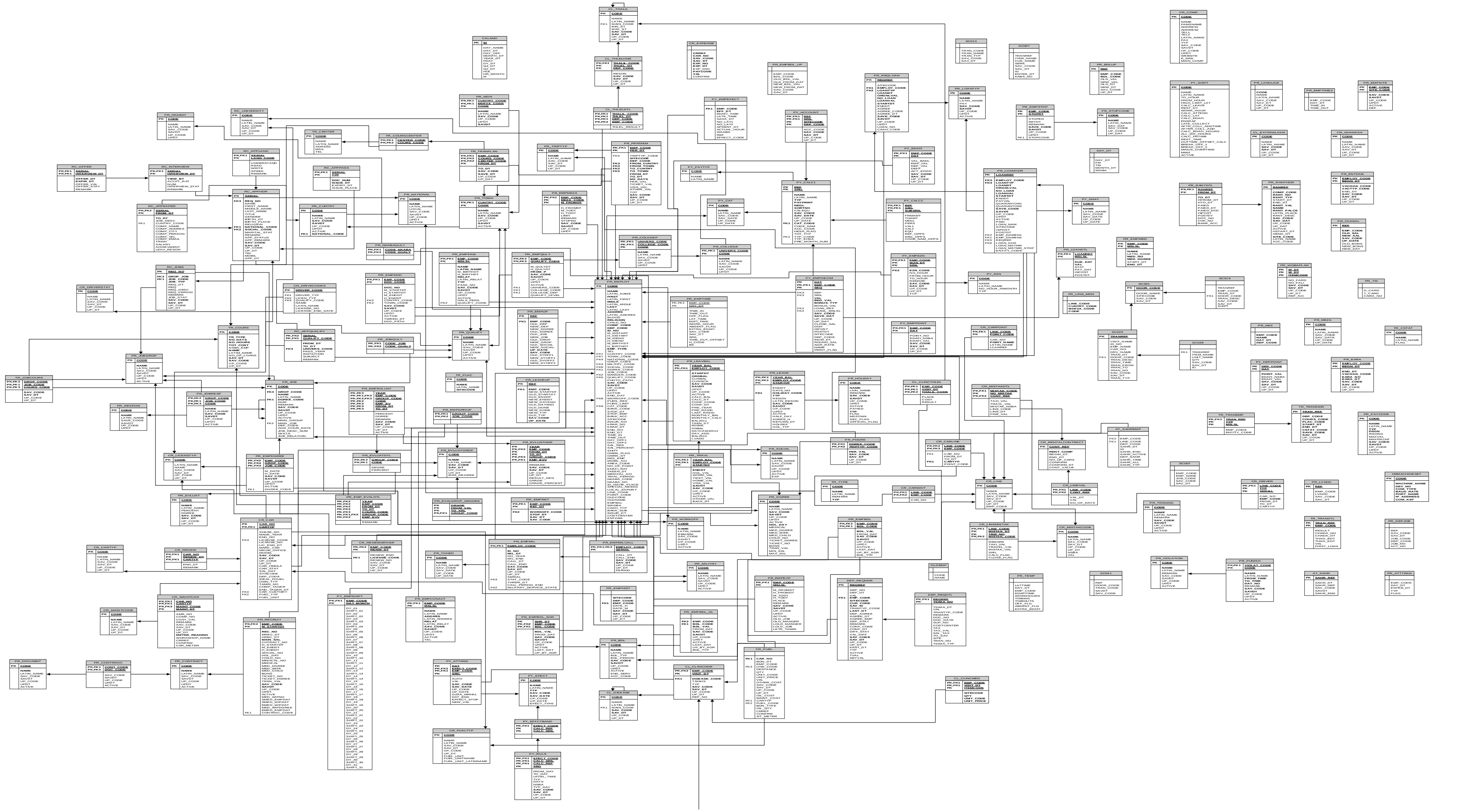
Algorithm 5
Relational Synthesis into 3NF with Dependency Preservation
Input: A universal relation R and a set of functional dependencies F on the
attributes of R.
1. Find a minimal cover G for F (use Algorithm 2);
2. For each left-hand-side X of a functional dependency that appears in G, create
a relation schema in D with attributes {X ∪ {A1} ∪ {A2} ... ∪ {Ak} },
where X-->A1, X-->A2, ..., X-->Ak are the only dependencies in G with X as
the left-hand-side (X is the key of this relation);
3. Place any remaining attributes (that have not been placed in any relation) in
a single relation schema to ensure the attribute preservation property.

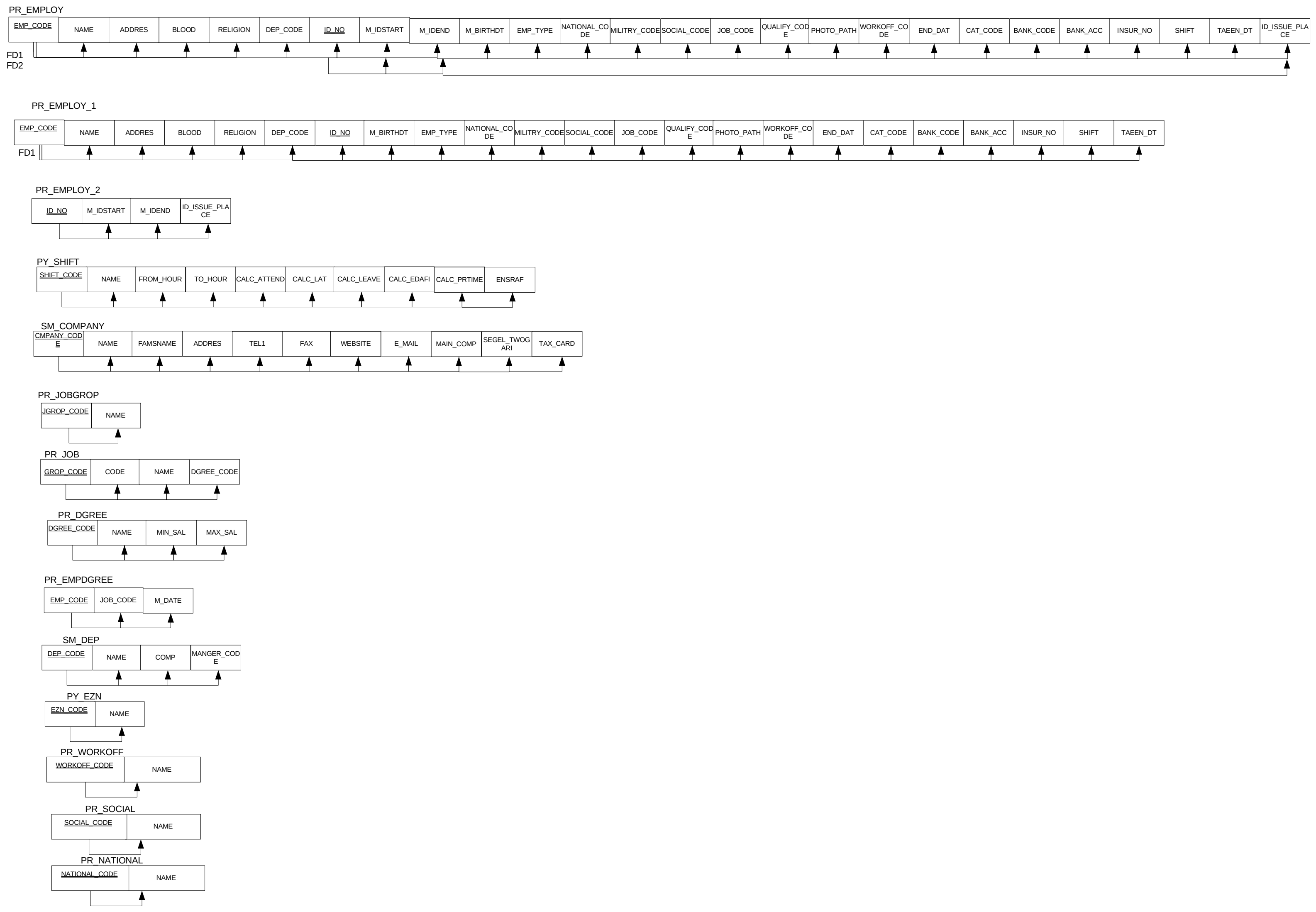
Algorithm 6
Relational Decomposition into BCNF with Nonadditive
Join Property
Input: A universal relation R and a set of functional dependencies F on the
attributes of R.
1. Set D := {R} ;
2. While there is a relation schema Q in D that is not in BCNF do
{
choose a relation schema Q in D that is not in BCNF;
find a functional dependency X-->Y in Q that violates BCNF;
replace Q in D by two relation schemas (Q – Y) and (X ∪ Y);
} ;

Algorithm 7
Relational Synthesis into 3NF with Dependency Preservation
and Nonadditive Join Property
Input: A universal relation R and a set of functional dependencies F on the
attributes of R.
1. Find a minimal cover G for F (use Algorithm 2).

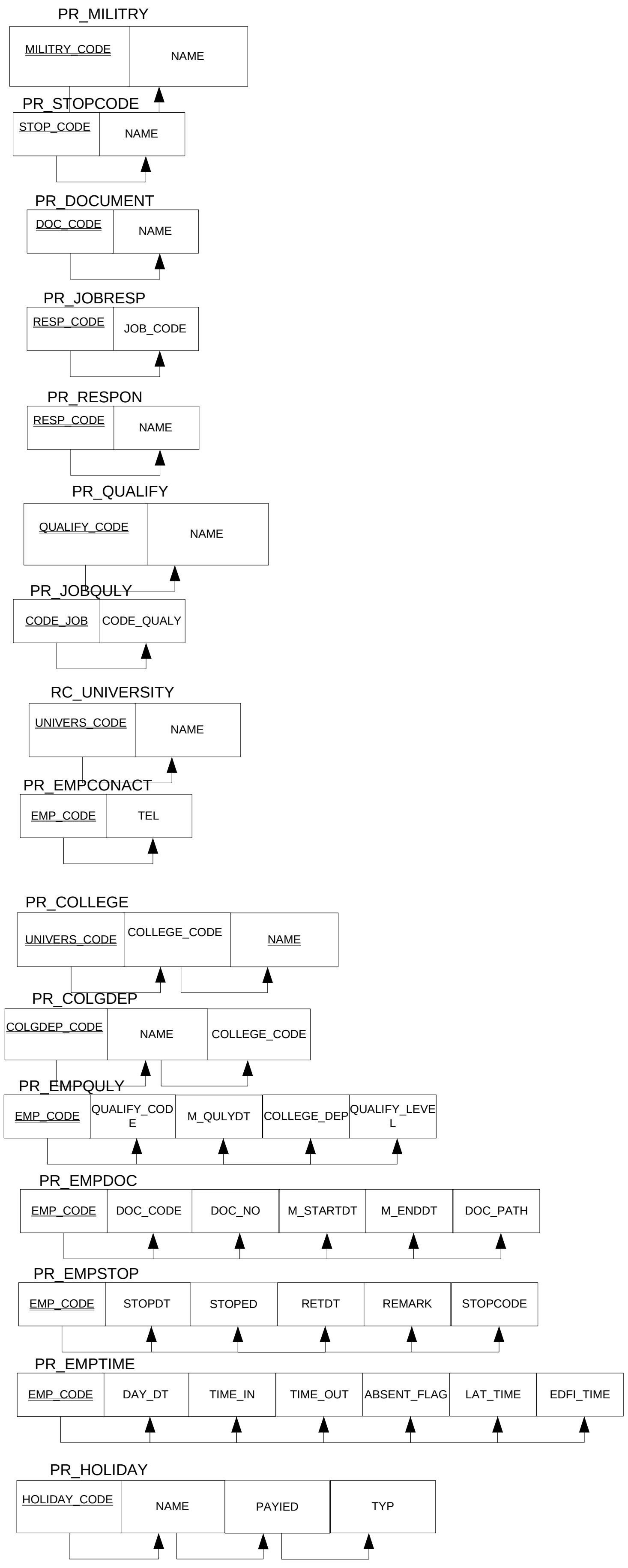
2. For each left-hand-side X of a functional dependency that appears in G, create
a relation schema in D with attributes {X ∪ {A1} ∪ {A2} ... ∪ {Ak} },
where X-->A1, X-->A2, ..., X-->Ak are the only dependencies in G with X as
left-hand-side (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one
more relation schema in D that contains attributes that form a key of R.7
(Algorithm 16.2(a) may be used to find a key.)
4. Eliminate redundant relations from the resulting set of relations in the relational
database schema. A relation R is considered redundant if R is a projection
of another relation S in the schema; alternately, R is subsumed by S.
Step 3 of Algorithm 16.6 involves identifying a key K of R. Algorithm 16.2(a) can be
used to identify a key K of R based on the set of given functional dependencies F.
Notice that the set of functional dependencies used to determine a key in Algorithm 3 could be either F or G, since they are equivalent.

<u>6- Case study</u>

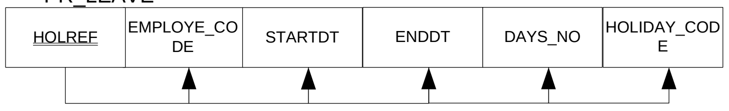




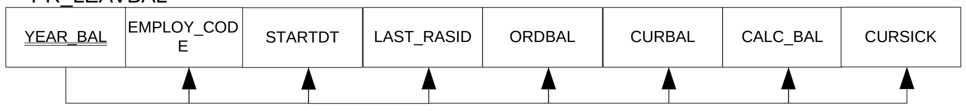
PR_EMPLOY
EMP_CODE
NAME
ADDRES
BLOOD
RELIGION
DEP_CODE
ID_NO
M_IDSTART
M_IDEND
M_BIRTHDT
EMP_TYPE
NATIONAL_CODE
MILITRY_CODE
SOCIAL_CODE
JOB_CODE
QUALIFY_CODE
PHOTO_PATH
WORKOFF_CODE
END_DAT
CAT_CODE
BANK_CODE
BANK_ACC
INSUR_NO
SHIFT
TAEEN_DT
ID_ISSUE_PLACE
FD1
FD2
PR_EMPLOY_1
EMP_CODE
NAME
ADDRES
BLOOD
RELIGION
DEP_CODE
ID_NO
M_BIRTHDT
EMP_TYPE
NATIONAL_CODE
MILITRY_CODE
SOCIAL_CODE
JOB_CODE
QUALIFY_CODE
PHOTO_PATH
WORKOFF_CODE
END_DAT
CAT_CODE
BANK_CODE
BANK_ACC
INSUR_NO
SHIFT
TAEEN_DT
FD1
PR_EMPLOY_2
ID_NO
M_IDSTART
M_IDEND
ID_ISSUE_PLACE
PY_SHIFT
SHIFT_CODE
NAME
FROM_HOUR
TO_HOUR
CALC_ATTEND
CALC_LAT
CALC_LEAVE
CALC_EDAFI
CALC_PRTIME
ENSRAF
SM_COMPANY
CMPANY_CODE
NAME
FAMSNAME
ADDRES
TEL1
FAX
WEBSITE
E_MAIL
MAIN_COMP
SEGEL_TWOGARI
TAX_CARD
PR_JOBGROP
JGROP_CODE
NAME
PR_JOB
GROP_CODE
CODE
NAME
DGREE_CODE
PR_DGREE
DGREE_CODE
NAME
MIN_SAL
MAX_SAL
PR_EMPDGREE
EMP_CODE
JOB_CODE
M_DATE
SM_DEP
DEP_CODE
NAME
COMP
MANGER_CODE
PY_EZN
EZN_CODE
NAME
PR_WORKOFF
WORKOFF_CODE
NAME
PR_SOCIAL
SOCIAL_CODE
NAME
PR_NATIONAL
NATIONAL_CODE
NAME


PR_MILITRY
MILITRY_CODE
NAME
PR_STOPCODE
STOP_CODE
NAME
PR_DOCUMENT
DOC_CODE
NAME
PR_JOBRESP
RESP_CODE
JOB_CODE
PR_RESPON
RESP_CODE
NAME
PR_QUALIFY
QUALIFY_CODE
NAME
PR_JOBQULY
CODE_JOB
CODE_QUALY
RC_UNIVERSITY
UNIVERS_CODE
NAME
PR_EMPCONACT
EMP_CODE
TEL
PR_COLLEGE
UNIVERS_CODE
COLLEGE_CODE
NAME
PR_COLGDEP
COLGDEP_CODE
NAME
COLLEGE_CODE
PR_EMPQULY
EMP_CODE
QUALIFY_CODE
M_QULYDT
COLLEGE_DEP
QUALIFY_LEVEL
PR_EMPDOC
EMP_CODE
DOC_CODE
DOC_NO
M_STARTDT
M_ENDDT
DOC_PATH
PR_EMPSTOP
EMP_CODE
STOPDT
STOPED
RETDT
REMARK
STOPCODE
PR_EMPTIME
EMP_CODE
DAY_DT
TIME_IN
TIME_OUT
ABSENT_FLAG
LAT_TIME
EDFI_TIME
PR_HOLIDAY
HOLIDAY_CODE
NAME
PAYIED
TYP

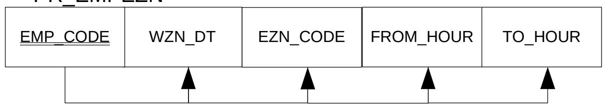
PR_LEAVE



PR_LEAVBAL



PR_EMPEZN



Overview

We got one of the companies of the relation schema (HR) and we choose some relation and we have identified Functional dependency and applied the normalization on this relations 1NF (First Normal Form), 2NF (Second Normal Form), 3NF (Third Normal Form)

Index PG. NO