module

MODULE-4
NORMALIZATION DATABASE DESGIN THEORY

4.1 Informal design guidelines for relation schemas

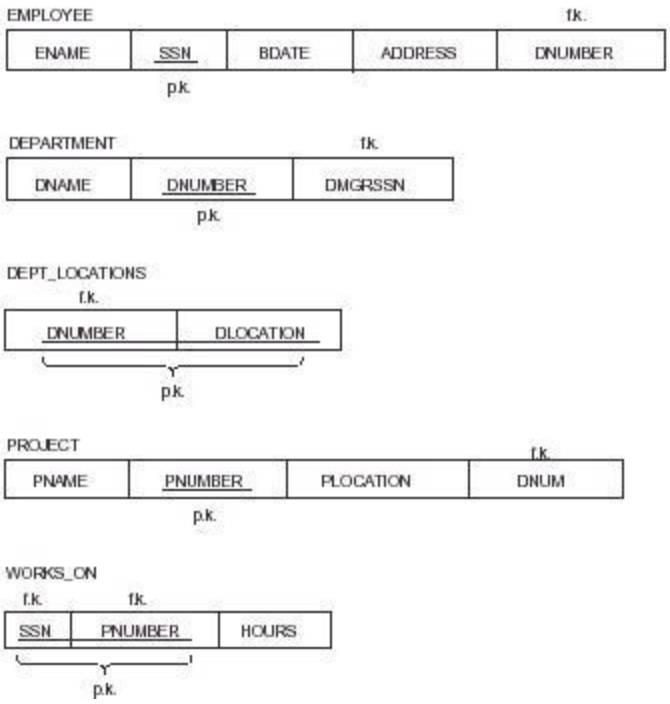
The four informal measures of quality for relation schema

- Semantics of the attributes
- Reducing the redundant values in tuples
- Reducing the null values in tuples
- Disallowing the possibility of generating spurious tuples

4.1.1 Semantics of relations attributes

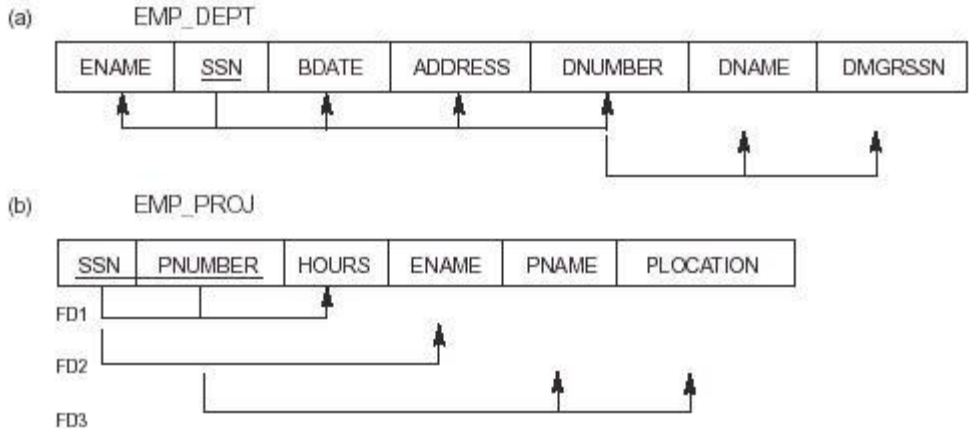
Specifies how to interpret the attributes values stored in a tuple of the relation. In other words, how the attribute value in a tuple relate to one another.

Figure 14.1 Simplified version of the COMPANY relational database schema.



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.
Save storage space and avoid update anomalies.
Insertion anomalies.
Deletion anomalies.
Modification anomalies

Figure 14.3 Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.



Insertion Anomalies

To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for that department that the employee works for, or nulls.

It's difficult to insert a new department that has no employee as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity - not a department entity.

Deletion Anomalies

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.

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.

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies occur. Reducing the null values in tuples. e.g., if 10% of employees have offices, it Is better to have a separate relation, EMP_OFFICE, rather than an attribute OFFICE_NUMBER in EMPLOYEE.

Guideline 3: Avoid placing attributes in a base relation whose values are mostly null. Disallowing spurious tuples.

Spurious tuples - tuples that are not in the original relation but generated by natural join of decomposed subrelations.

Example: decompose EMP_PROJ into EMP_LOCS and EMP_PROJ1.

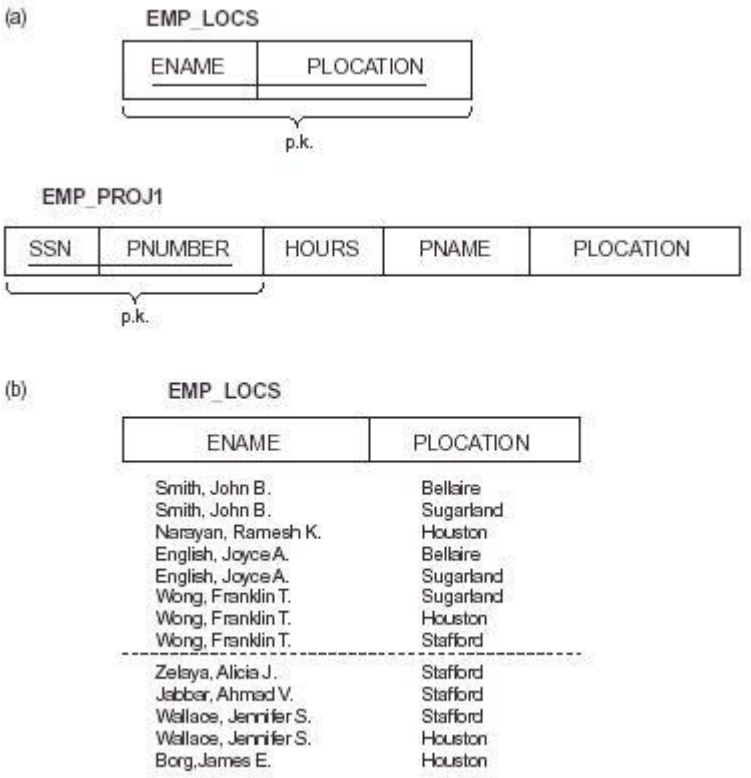


Fig. 14.5a

Guideline 4: Design relation schemas so that they can be naturally JOINed on primary keys or foreign keys in a way that guarantees no spurious tuples are generated.

6.2 A functional dependency (FD) is a constraint between two sets of attributes from the database. It is denoted by

$X \rightarrow Y$

We say that "*Y* is **functionally dependent** on *X*". Also, X is called the left-hand side of the FD. Y is called the right-hand side of the FD.

A functional dependency is a property of the semantics or meaning of the attributes, i.e., a property of the relation schema. They must hold on all relation states (extensions) of R. Relation extensions r(R). A FD X $\rightarrow$ *Y* is a full *functional dependency* if removal of any attribute from X *means that the dependency does not hold any more; otherwise, it is a partial functional dependency.*

Examples:

1. SSN $\rightarrow$ ENAME
2. PNUMBER $\rightarrow$ {PNAME, PLOCATION}
3. {SSN, PNUMBER} $\rightarrow$ HOURS

FD is property of the relation schema R, not of a particular relation state/instance

Let R be a relation schema, where X $\subseteq$ R and Y $\subseteq$ R

$t_1, t_2 \in r$, $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

The FD X $\rightarrow$ Y holds on R if and only if for all possible relations r(R), whenever two tuples of r agree on the attributes of X, they also agree on the attributes of Y.

- the single arrow $\rightarrow$ denotes "functional dependency"
- *X* $\rightarrow$ *Y* can also be read as "*X* determines *Y*"
- the double arrow $\Rightarrow$ denotes "logical implication"

4.2.1 Inference Rules

IR1. Reflexivity e.g. *X* $\rightarrow$ *X*

- a formal statement of *trivial dependencies*; useful for derivations
- if a dependency holds, then we can freely expand its left hand side
- the "most powerful" inference rule; useful in multi-step derivations

Armstrong inference rules

are sound

meaning that given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using IR1 through IR3 holds every relation state r of R that specifies the dependencies in F. In other words, rules can be used to derive precisely the closure or no additional FD can be derived.

complete

meaning that using IR1 through IR3 repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F. In other words, given a set of FDs, all implied FDs can be derived using these 3 rules.

Closure of a Set of Functional Dependencies

Given a set X of FDs in relation R, the set of all FDs that are implied by X is called the closure of X, and is denoted X^+.

Algorithms for determining X^+

$X^+ := X;$

repeat

$oldX^+ := X^+$

for each FD $Y \rightarrow Z$ *in F do*

if $Y \subseteq X^+$ *then* $X^+ := X^+ \cup Z;$

until $oldX^+ = X^+;$

Example:

A $\rightarrow$ BC

E $\rightarrow$ CF

B $\rightarrow$ E

CD $\rightarrow$ EF

Compute $\{A, B\}^+$ of the set of attributes under this set of FDs.

Solution:

Step1: $\{A, B\}^+ := \{A, B\}$.

Go round the inner loop 4 time, once for each of the given FDs. On the first iteration, for $A \subseteq BC$ A $\{A, B\}^+$ $\{A, B\}^+ := \{A, B, C\}$.

Step2: On the second iteration, for E

$\rightarrow$CF, $\notin$ {A, B, C}

Step3 :On the third iteration, for B $\rightarrow$E

B $\subseteq${A, B,C}+

{A, B}+ := {A, B, C, E}.

Step4: On the fourth iteration, for CD →EF remains unchanged.

Go round the inner loop 4 times again. On the first iteration result does not change; on the second it expands to {A,B,C,E,F}; On the third and forth it does not change.

Now go round the inner loop 4 times. Closure does not change and so the whole process terminates, with
$\{A,B\}_+ = \{A,B,C,E,F\}$

Example.

F = { SSN →ENAME, PNUMBER →{PNAME, PLOCATION}, {SSN,PNUMBER} →HOURS }

$\{SSN\}^+ = \{SSN, ENAME\}$

$\{PNUMBER\}^+ = ?$

$\{SSN,PNUMBER\}^+ = ?$

4.3 Normalization

The purpose of normalization.

- The problems associated with redundant data.
- The identification of various types of update anomalies such as insertion, deletion, and modification anomalies.
- How to recognize the appropriateness or quality of the design of relations.
- The concept of functional dependency, the main tool for measuring the appropriateness of attribute groupings in relations.
- How functional dependencies can be used to group attributes into relations that are in a known normal form.
- How to define normal forms for relations.
- How to undertake the process of normalization.
- How to identify the most commonly used normal forms, namely first (1NF), second (2NF), and third (3NF) normal forms, and Boyce-Codd normal form (BCNF).
- How to identify fourth (4NF), and fifth (5NF) normal forms

Main objective in developing a logical data model for relational database systems is to create an accurate representation of the data, its relationships, and constraints. To achieve this objective, we must identify a suitable set of relations. A technique for producing a set of relations with desirable properties, given the data requirements of an enterprise

NORMAL FORMS

A relation is defined as a *set of tuples*. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for *all* their attributes.

Any set of attributes of a relation schema is called a **superkey**. Every relation has at least one superkey—the set of all its attributes. A **key** is a *minimal superkey*, i.e., a superkey from which we cannot remove any attribute and still have the uniqueness constraint hold.

In general, a relation schema may have more than one key. In this case, each of the keys is called a **candidate key.** It is common to designate one of the candidate keys as the **primary key** of the relation. A **foreign key** is a key in a relation R but it's not a key (just an attribute) in other relation R' of the same schema.

Integrity Constraints

The **entity integrity constraint** states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values for the primary key implies that we cannot identify some tuples.

The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an *existing tuple* in that relation.

An attribute of a relation schema R is called a **prime attribute** of the relation R if it is a member of *any key* of the relation R. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key.

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the in the third normal form (3 NF).

Normalization of data can be looked on as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties. One objective of the original normalization process is to ensure that the update anomalies such as insertion, deletion, and modification anomalies do not occur

The most commonly used normal forms

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal

Form Other Normal Forms

- Fourth Normal Form
- Fifth Normal Form
- Domain Key Normal Form

4.3.1 First Normal Form (1NF)

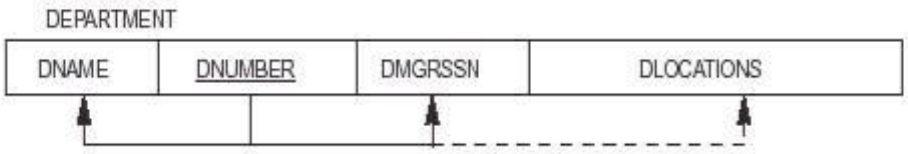
First normal form is now considered to be part of the formal definition of a relation; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domains of attributes must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.

Practical Rule: "Eliminate Repeating Groups," i.e., make a separate table for each set of related attributes, and give each table a primary key.

Formal Definition: A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only.

Figure 14.8 Normalization into 1NF. (a) Relation schema that is not in 1NF. (b) Example relation instance. (c) 1NF relation with redundancy.

(a)



(b) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

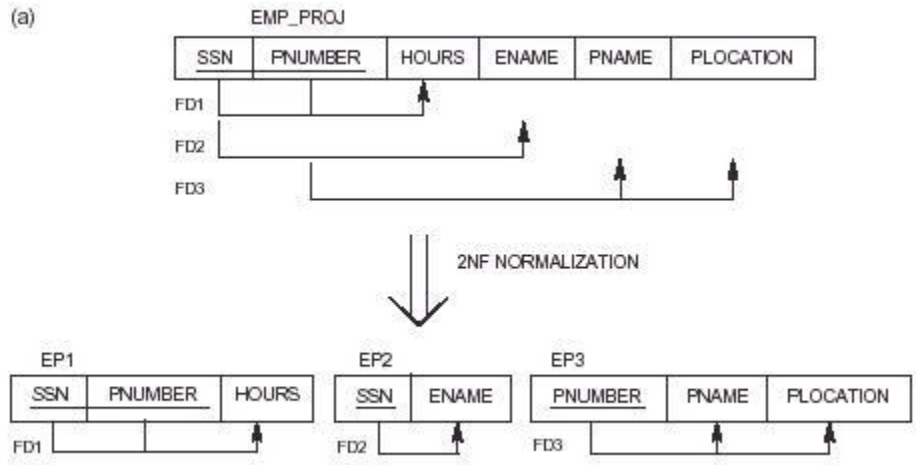
DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

4.3.2 Second Normal Form (2NF)

Second normal form is based on the concept of fully functional dependency. A functional X→ Y is a fully functional dependency is removal of any attribute A from X means that the dependency does not hold any more. A relation schema is in 2NF if every nonprime attribute in relation is fully functionally dependent on the primary key of the relation. It also can be restated as: a relation schema is in 2NF if every nonprime attribute in relation is not partially dependent on any key of the relation.

Practical Rule: "Eliminate Redundant Data," i.e., if an attribute depends on only part of a multivalued key, remove it to a separate table.

Formal Definition: A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.



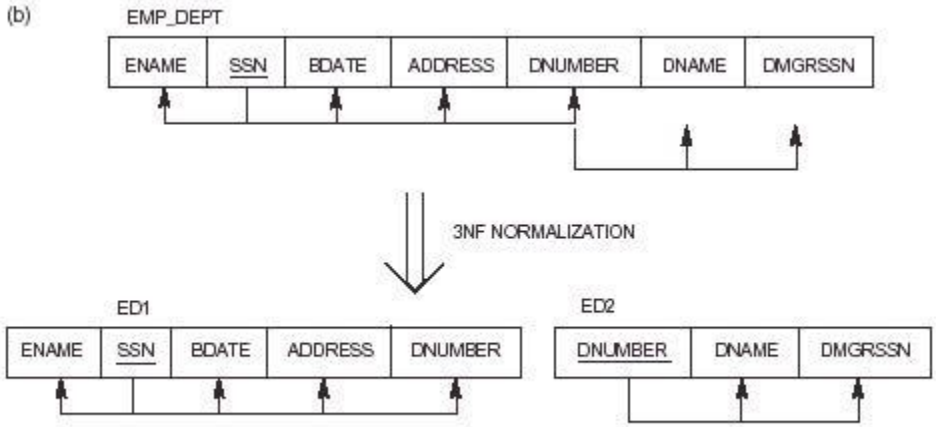
4.3.3 Third Normal Form (3NF)

Third normal form is based on the concept of transitive dependency. A functional dependency X → Y in a relation is a transitive dependency if there is a set of attributes Z that is not a subset of any key of the relation, and both X→ Z and Z →Y hold. In other words, a relation is in 3NF if, whenever a functional dependency

X →A holds in the relation, either (a) X is a superkey of the relation, or (b) A is a prime attribute of the relation.

Practical Rule: "Eliminate Columns not Dependent on Key," i.e., if attributes do not contribute to a description of a key, remove them to a separate table

Formal Definition: A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.



1NF: R is in 1NF iff all domain values are atomic.

2NF: R is in 2 NF iff R is in 1NF and every nonkey attribute is fully dependent on the key.

3NF: R is in 3NF iff R is 2NF and every nonkey attribute is non-transitively dependent on the key.

4.4 Boyce-Codd Normal Form (BCNF)

A relation schema R is in Boyce-Codd Normal Form (BCNF) if whenever a FD X -> A holds in R, then X is a superkey of R

- Each normal form is strictly stronger than the previous one:
- Every 2NF relation is in 1NF Every 3NF relation is in 2NF
- Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF

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

Additional criteria may be needed to ensure the the set of relations in a relational database are satisfactory.

Figure 14.12 Boyce-Codd normal form. (a) BCNF normalization with the dependency of FD2 being "lost" in the decomposition. (b) A relation *R* in 3NF but not in BCNF.

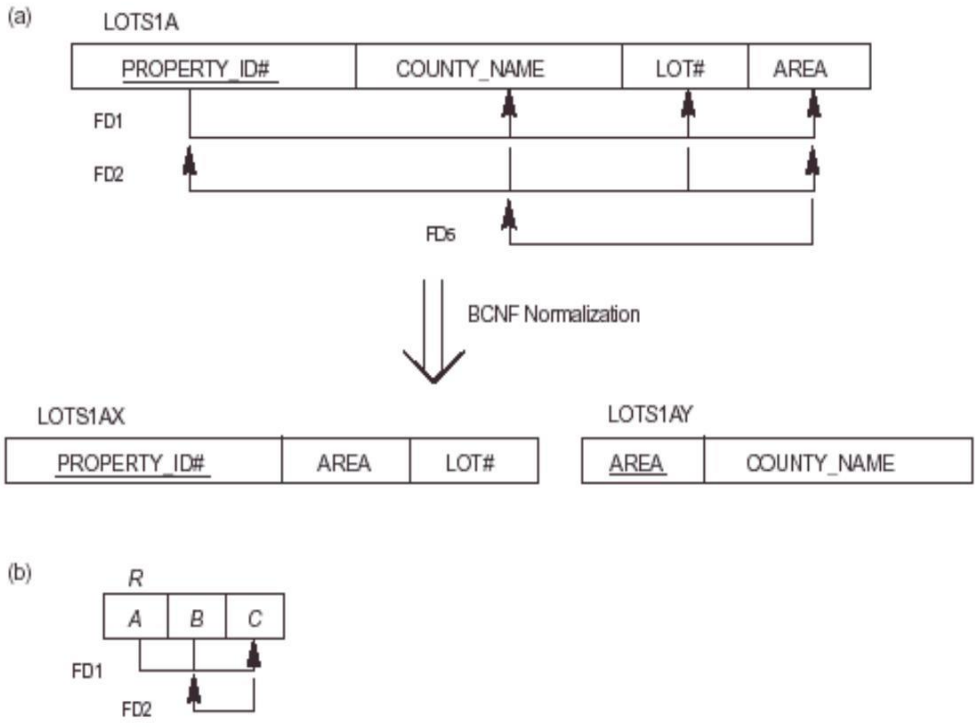


Figure 14.13 A relation TEACH that is in 3NF but not in BCNF.

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

If X → Y is non-trivial then X is a super key

STREET	CITY	ZIP

{CITY,STREET } → ZIP

ZIP → CITY

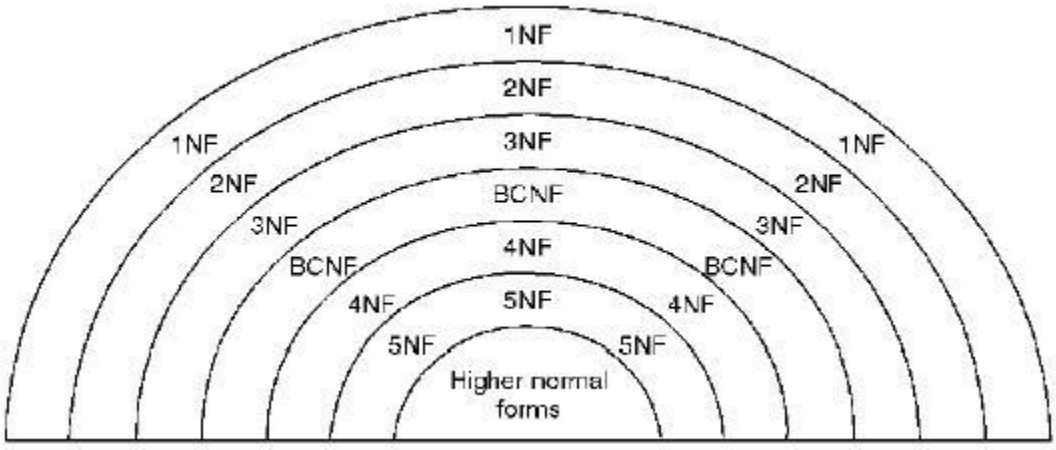
- Insertion anomaly: the city of a zip code can‘t be stored, if the street is not given

Normalization

STREET	ZIP

ZIP	CITY

Relationship Between Normal Forms



Questions

1. What is the need for normalization? Explain the first,second and third normal forms with examples.
2. Explain informal design guidelines for relation schemas.
3. A What is functional dependency?write an algorithm to find a minimal cover for a set of functional dependencies.
4. What is the need for normalization ?explain second normal form
5. Which normal form is based on the concept of transitive dependency? Explain with an example the decomposition into 3NF
6. Explain multivalued dependency. Explain 4NF with an example.
7. Explain any Two informal quality measures employed for a relation schema Design?
8. Consider the following relations: Car_sale(car_no,date-sold,salemanno,commission%,discount).assume a car can be sold by multiple salesman and hence primary key is {car-no,salesman} additional dependencies are: Date-sold → discount and salesmanno → commision Yes this relation is in 1NF
9. Discuss the minimal sets of FD‘S?

4.1 Properties of relational decomposition

Normalization Algorithms based on FDs to synthesize 3NF and BCNF describe two desirable properties (known as properties of decomposition).

- Dependency Preservation Property
- Lossless join property

Dependency Preservation Property enables us to enforce a constraint on the original relation from corresponding instances in the smaller relations.

Lossless join property enables us to find any instance of the original relation from corresponding instances in the smaller relations (Both used by the design algorithms to achieve desirable decompositions).
A property of decomposition, which ensures that no spurious rows are generated when relations are reunited through a natural join operation.

4.2 Algorithms for Relational Database Schema Design

Individual relations being in higher normal do not guarantee a good deign Database schema must posses additional properties to guarantee a good design.

Relation Decomposition and Insufficiency of Normal Forms

Suppose R = { A_1, A_2, …, A_n} that includes all the attributes of the database. R is a universal relation schema, which states that every attribute name is unique. Using FDs, the algorithms decomposes the universal relation schema R into a set of relation schemas

D = {R_1, R_2, …, R_n} that will become the relational database schema; D is called a decomposition of R. Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are lost; we have

$$\bigcup_{i=1}^{m} R_i = R$$

This is called attribute preservation condition of a decomposition.

4.2.1 Decomposition and Dependency Preservation

We want to preserve dependencies because each dependencies in F represents a constraint on the Database.

We would like to check easily that updates to the database do not result in illegal relations being created.

It would be nice if our design allowed us to check updates without having to compute natural joins. To know whether joins must be computed, we need to determine what functional dependencies may be tested by checking each relation individually.

Let F be a set of functional dependencies on schema R. Let D = {R1, R2, …, Rn} be a decomposition of R. Given a set of dependencies F on R, the projection of F on Ri, $\pi_{Ri}(F)$, where R_i is a subset of R, is the set of all functional dependencies $X \rightarrow Y$ such that attributes in $X \cup Y$ are all contained in Ri. Hence the projection of F on each relation schema Ri in the decomposition D is the set of FDs in F+, such that all their LHS and RHS attributes are in Ri. Hence, the projection of F on each relation schema Ri in the decomposition D is the set of functional dependencies in F+.

$((\pi_{R1}(F)) \cup (\pi_{R2}(F)) \cup \ \ldots \cup (\pi_{Rm}(F)))^+ = F^+$

i.e., the union of the dependencies that hold on each Ri belongs to D be equivalent to closure of F (all possible FDs)

/*Decompose relation, R, with functional dependencies, into relations, R_1,..., R_n, with associated functional dependencies,

F_1,..., F_k.

The decomposition is **dependency preserving iff**:

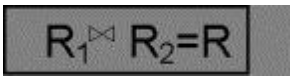
$F^+ = (F_1 \ \ldots \ F_k)^+$ */

If each functional dependency specified in F either appeared directly in one of the relation schema R in the decomposition D or could be inferred from the dependencies that appear in some R.

7.2.2 Lossless-join Dependency

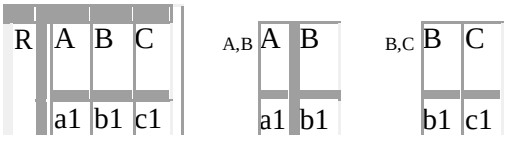
A property of decomposition, which ensures that no spurious rows are generated when relations are reunited through a natural join operation.

Lossless-join property refers to when we decompose a relation into two relations - we can rejoin the resulting relations to produce the original relation.



Decompose relation, R, with functional dependencies, F, into relations, R1 and R2, with attributes, A1 and A2, and associated functional dependencies, F1 and F

- Decompositions are projections of relational schemas



a2	b2	c2
a3	b1	c3

a2	b2
a3	b1

b2	c2
b1	c3

- Old tables should be derivable from the newer ones through the natural join operation

$\Pi_{A,B}(R) \bowtie \Pi_{B,C}(R)$

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c3
a1	b1	c3
a3	b1	c1

- Wrong!
- R_1, R_2 is a lossless join decomposition of R iff the attributes common to R_1 and R_2 contain a key for at least one of the involved relations

R

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c1

$\Pi_{A,B}$

A	B
a1	b1
a2	b2
a3	b1

$\Pi_{B,C}$

B	C
b1	c1
b2	c2

- $\Pi_{A,B}(R) \cap \Pi_{B,C}(R) = B$

The decomposition is **lossless iff**:

- $A_1 \quad A_2 \quad A_1 \backslash A_2$ is in F^+, or
- $A_1 \quad A_2 \quad A_2 \backslash A_1$ is in F^+

However, sometimes there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and 5NF.

4.3 Multivalued Dependencies and Fourth Normal Form (4NF)

4NF associated with a dependency called **multi-valued dependency** (MVD). MVDs in a relation are due to first normal form (1NF), which disallows an attribute in a row from having a set of values.

MVD represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B, and a set of values for C. However, the set of values for B and C are independent of each other.
MVD between attributes A, B, and C in a relation using the following notation

A B ***(A multidetermines B)***

A C

Formal Definition of Multivalued Dependency

A **multivalued dependency** (MVD) X Y specified on R, where X, and Y are both subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

4.3.1 Fourth Normal Form (4NF)

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s).

A Relation is in 4NF if it is in 3NF and there is no multivalued dependencies.

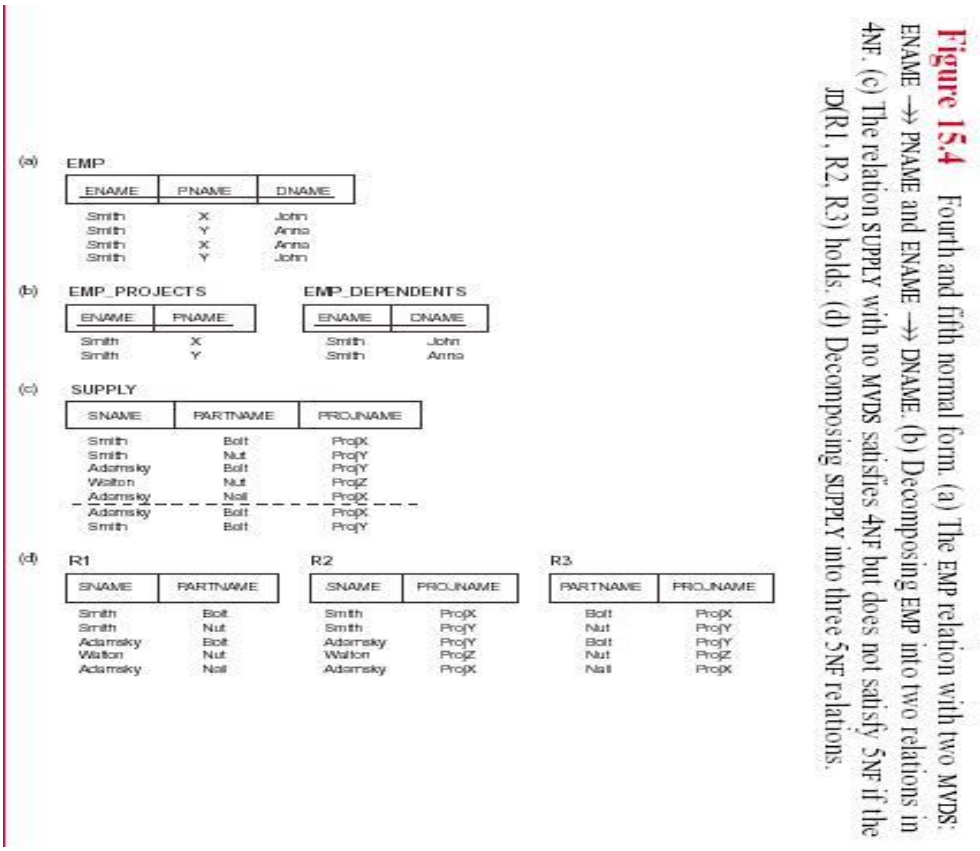


Figure 15.4 Fourth and fifth normal form. (a) The EMP relation with two MVDs: ENAME →→ PNAME and ENAME →→ DNAME. (b) Decomposing EMP into two relations in 4NF. (c) The relation SUPPLY with no MVDS satisfies 4NF but does not satisfy 5NF if the JD(R1, R2, R3) holds. (d) Decomposing SUPPLY into three 5NF relations.

4.4 Join Dependencies and 5 NF

A **join dependency** (**JD**), denoted by JD{R_1, R_2, …, R_n}, specified on relation schema *R*, specifies a constraint on the states *r* of *R*. The constraint states that every legal state *r* of *R* should have a lossless join decomposition into R_1, R_2, …, R_n; that is, for every such *r* we have

* (π_{R1}(r), (π_{R2}(r), … (π_{Rn}(r)) = r

Lossless-join property refers to when we decompose a relation into two relations - we can rejoin the resulting relations to produce the original relation. However, sometimes there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and 5NF.

5NF (or project-join normal form (PJNF))

A relation that has no join dependency.

4.5 Other dependencies:

4.5.1 Template Dependencies

The idea behind template dependencies is to specify a template—or example—that defines each constraint or dependency. There are two types of templates: tuple-generating templates and constraint-generating templates. A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the template conclusion. For tuple-generating templates, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there. For constraint-generating templates, the template conclusion is a condition that must hold on the hypothesis tuples.

4.5.2 Domain Key Normal Form

The idea behind **domain-key normal form (DKNF)** is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints.

A relation is said to be in **DKNF** if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and key constraints on the relation.

However, because of the difficulty of including complex constraints in a DKNF relation, its practical utility is limited, since it may be quite difficult to specify general integrity constraints.
For example, consider a relation (where VIN# is the vehicle identification

number) and another relation MANUFACTURE(VIN#, COUNTRY) (where COUNTRY is the country of manufacture). A general constraint may be of the following form: "If the MAKE is either Toyota or Lexus, then the first character of the VIN# is a "*J*" if the country of manufacture is Japan; if the MAKE is Honda or Acura, the second character of the VIN# is a "*J*" if the country of manufacture is Japan." There is no simplified way to represent such constraints short of writing a procedure (or general assertions) to test them.

Questions

1. Explain
    i. Inclusion dependency
    ii. ii) Domain Key Normal Form
2. Explain multivolume dependency and fourth normal form, with an example
3. Explain lossless join property
4. what are the ACID Properties? Explain any One?
5. What is Serializibility?How can seriaizability?Justify your answer?