Part 5: Relational Database Theory - Normalization

Relational Database Design

Relational Database Schema:

A set of relational schemas.

Relation Schema:

It consists of a number of attributes.

Relational Database Design:

It is a subject of schema design.

Schema Design:

a. Attributes are grouped to form a relation schema using the common sense of the database designer

b. By mapping a schema specified in the Entity-Relationship model into a relational schema.

Questions:

- Why one grouping of attributes into a relation schema may be better than another.
- Measurement of appropriateness or of the quality of the design except for the intuition of the designer.
- What is the theory to choose "good" relation schemas.

Measurement

Two Levels of Measurements:

1. **Logical Level:**

   It refers to how users interpret the relation schemas and the meaning of their attributes.

   Good relation schemas at this level helps users

   a. to clearly understand the meaning of the data tuples in the relations

   b. to formulate the correct queries.

   This level concerns both base relations and views (virtual relations).

2. **Manipulation (or storage) Level:**

- **It refers to how the tuples in a base relation are stored and updated.**
- **This level applies only to schemas of base relations - which are physically stored as files**
- **Relational database design theories developed mainly concern at this base relation level.**

Informal Design Guidelines for Relation Schemas

1. **Semantics of the Relation Attributes:**

- **Design a relation schema in such a way that it is easy to explain its meaning.**

2. **Reducing the Redundant Values in Tuples:**

- **Design the base relation schemas so that no insertion, deletion, or modification anomalies occur in the relations.**

Note:

- **Sometimes these guidelines have to be violated in order to improve the performance of certain queries, but the constraints have to be embedded in the application programs to avoid anomalies.**

Informal Design Guidelines for Relation Schemas

3. **Reducing the Null Values in Tuples:**

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

4. **Disallowing Spurious Tuples from Join Operations:**

- Design relation schemas so that they can be JOINed with equality conditions on attributes that either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

Simplified Company Database Schema

EMPLOYEE

				f.k.
ENAME	SSN	BDATE	ADDRESS	DNUMBER
	p.k.			

DEPARTMENT

		f.k.	
DNAME	DNUMBER	DMGRSSN	DLOCATIONS
	p.k.		

DEPT_LOCATIONS

f.k.	
DNUMBER	DLOCATION
p.k.	

PORJECT

			f.k.
PNAME	PNUMBER	PLOCATION	DNUM
	p.k.		

WORKS_ON

f.k.	f.k.	
SSN	PNUMBER	HOURS
p.k.		

Relations for Simplified Company Database

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	291 Berry,Bellaire,TX	4
Narayan,Remesh 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	DMGRSSN
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

Semantics of the Relation Attributes

Semantics:

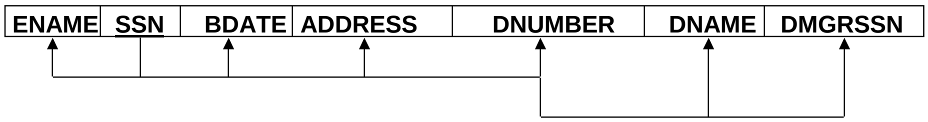
- **How to interpret the attribute values stored in a tuple of the relation.**
- **How the attribute value in a tuple are related to one another.**

Schema Design:

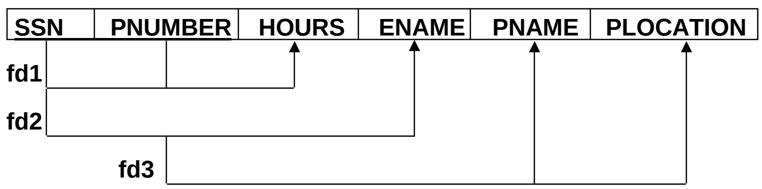
- **The better the relation schema design, the easier to explain the semantics of the relation.**
- **One heuristics is that we should not combine attributes from multiple entity types and relationship types into a single relation.**
- **A relation schema corresponds to one entity type or one relationship type.**
- **Only the attribute related to 1-1 or 1-m relationship can be included as a (foreign key) attribute in one of the participating entity types.**
- **A foreign key represent an implicit relationship between participating entity types.**
- **Mixing attributes from distinct real world entities tends to be semantically unclear, and causes problems when used as base relations.**

Two Relation Schema with Mixed Attributes

EMP_DEPT



EMP_PROJ



Relation EMP_DEPT

EMP_DEPT

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
John Smith	123456789	09-JAN-55	731 Fondren, Houston, TX	5	Research	333445555
Franklin Wong	333445555	08-DEC-45	638 Voss, Houston, TX	5	Research	333445555
Alicia Zelaya	999887777	19-JUL-58	3321 Castle, Spring TX	4	Administration	987654321
Jennifer Wallace	987654321	19-JUN-31	291 Berry, Bellaire, TX	4	Administration	987654321
Ramesh Narayan	666884444	15-SEP-52	975 FireOak, Humble, TX	5	Research	333445555
Joyce English	453453453	31-JUL-62	5631 Rice, Houston, TX	5	Research	333445555
Ahmad Jabbar	987987987	29-MAR-59	980 Dallas, Houston, TX	4	Administration	987654321
James Borg	888665555	10-NOV-27	450 Stone, Houston, TX	1	Headquarters	

Relation EMP_PROJ

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION
123456789	1	32.5	John B. Smith	ProductX	Bellaire
123456789	2	7.5	John B. Smith	ProductY	Sugarland
666884444	3	40.0	Ramesh Narayan	ProductZ	Houston
453453453	1	20.0	Joyce English	ProductX	Bellaire
453453453	2	20.0	Joyce English	ProductY	Sugarland
333445555	2	10.0	Franklin Wong	ProductY	Sugarland
333445555	3	10.0	Franklin Wong	ProductZ	Houston
333445555	10	10.0	Franklin Wong	Computerization	Stafford
333445555	20	10.0	Franklin Wong	Reorganization	Houston
999887777	30	30.0	Alicia Zelaya	Newbenefits	Stafford
999887777	10	10.0	Alicia Zelaya	Computerization	Stafford
987987987	10	35.0	Ahmad Jabbar	Computerization	Stafford
987987987	30	5.0	Ahmad Jabbar	Newbenefits	Stafford
987654321	30	20.0	Jennifer Wallace	Newbenefits	Stafford
987654321	20	15.0	Jennifer Wallace	Reorganization	Houston
888665555	20	null	James Borg	Reorganization	Houston

Update Anomalies

<u>Insertion Anomalies:</u>

- **To insert a new employee tuple into EMP_DEPT, we must include the attribute values for the department that the employee works for , or nulls if the employee not work for a department as yet.**
- **It is difficulty 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 attributes for employee.**
- **This causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supported to represent employee entity. - not a department entity.**

Update Anomalies

Deletion Anomalies:

- **If an employee tuple is deleted from EMP_DEPT that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.**

Modification Anomalies:

- **If a department change a new manager, the tuples of all employees who work in that department have to be updated, otherwise, the database will become inconsistent.**
- **If the updating of some tuples are overlooked, the same department will be shown to have two different values for manager in different employee tuples, which should not be the case.**

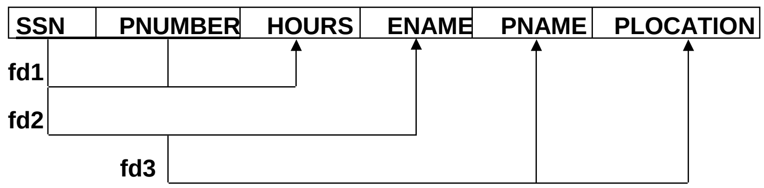
Spurious Tuples

Spurious Tuples:

- **Suppose to decompose EMP_PROJ relation into two base tables, EMP_PROJ1 and EMP_LOCS.**
- **This decomposition is very bad schema design, because we can not recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS.**
- **If we use NATURAL_JOIN operation on on EMP_PROJ1 and EMP_LOCS, we get many more tuples than EMP_PROJ had.**
- **The additional tuples that were not in EMP_PROJ are spurious tuples because they represent the spurious or wrong information that is not valid.**
- **The reason why the decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS is that the PLOCATION attribute is chosen as JOIN attribute, and PLOCATION is neither a primary key nor a foreign key in either EMP_LOCS and EMP_PROJ1.**

Decomposition of EMP_PROJ on Plocation

EMP_PROJ



EMP_LOCS

ENAME	PLOCATION

P.K.

EMP_PROJ1

SSN	PNUMBER	HOUR	PNAME	PLOCATION

P.K.

Relation EMP_LOCS

EMP_LOCS

ENAME	PLOCATION
John B. Smith	Bellaire
John B. Smith	Bellaire
Ramesh K. Narayan	Houston
Joyce A. English	Bellaire
Joyce A. English	Sugarland
Franklin T. Wong	Sugarland
Franklin T. Wong	Houston
Franklin T. Wong	Stafford
Alicia J. Zelaya	Stafford
Ahmad V. Jabbar	Stafford
Jennifer S. Wallace	Stafford
Jennifer S. Wallace	Houston
James E. Borg	Houston

EMP_PROJ1 Relation

EMP_PROJ1

SSN	PNUMBER	HOUR	PNAME	PLOCATION
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganizationl	Houston
888665555	20	nul	Reorganization	Houston

Result of Natural Join on EMP_PROJ1 and EMP_LOCS

SSN	PNUMBER	HOURS	PNAME	PLOCATION	ENAME
123456789	1	32.5	ProductX	Bellaire	John B. Smith
*123456789	1	32.5	ProductX	Bellaire	Joyce A. English
123456789	2	7.5	ProductY	Sugarland	John B. Smith
*123456789	2	7.5	ProductY	Sugarland	Joyce A. English
*123456789	2	7.5	ProductY	Sugarland	Franklin T. Wong
666884444	3	40.0	ProductZ	Houston	Ramesh K. Narayan
*666884444	3	40.0	ProductZ	Houston	Franklin T. Wong
*453453453	1	20.0	ProductX	Bellaire	John B. Smith
453453453	1	20.0	ProductX	Bellaire	Joyce A. English
*453453453	2	20.0	ProductY	Sugarland	John B. Smith
453453453	2	20.0	ProductY	Sugarland	Joyce A. English
*453453453	2	20.0	ProductY	Sugarland	Franklin T. Wong
*333445555	2	10.0	ProductY	Sugarland	John B. Smith
*333445555	2	10.0	ProductY	Sugarland	Joyce A. English
333445555	2	10.0	ProductY	Sugarlaand	Franklin T. Wong
*333445555	3	10.0	ProductZ	Houston	Ramesh K. Narayan
333445555	3	10.0	ProductZ	Houston	Franklin T. Wong
333445555	10	10.0	Computerization	Stafford	Franklin T. Wong
*333445555	20	10.0	Reorganization	Houston	Ramesh K.Narayan
333445555	20	10.0	Reorganization	Houston	Franklin T. Wong

Functional Dependencies

Functional Dependencies:

- It is constraint between two sets of attributes from the database.
- The functional dependency constraint states that for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, we must have that $t_1[Y] = t_2[Y]$.
- This means that the values of the Y component of a tuple in r dependent on, or are determined by, the values of the X component, or

  The values of the X component of a tuple uniquely (or functionally) determine the values of the Y component.

Alternative definition of FD:

- In a relation schema R, X functionally determines Y if and only if whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value.
- If a constraint on R states that there can not 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 $X \rightarrow Y$ for any subset of attributes Y of R.
- If $X \rightarrow Y$ in R, this does not say whether or not $Y \rightarrow X$ in R.

Example

SSN $\rightarrow$ ENAME

The value of employee's social security number (SSN) uniquely determines the employee name (ENAME)

PNUMBER $\rightarrow$ {PNAME, PLOCATION}

The value of a project's number (PNUMBER) uniquely determines the project name (PNAME) and location (PLOCATION)

{SSN, PNUMBER} $\rightarrow$ {HOURS}

A combination of SSN and PNUMBER values uniquely determines the number of hours the employee works on the project per week (HOURS).

- FD can not be automatically inferred from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.

Inference Derivation of Functional Dependencies

Suppose there is a set of functional dependencies

F = {SSN $\rightarrow$ {ENAME, BDATE, ADDRESS, DNUMBER},
DNUMBER $\rightarrow$ {DNAME, DMGRSSN}}

The inferred set of functional dependencies from F as follows:
SSN $\rightarrow$ {DNAME, DMGRSSN},
SSN $\rightarrow$ SSN, (trivial functional dependencies)
DNUMBER $\rightarrow$ DNAME

<u>The Closure of Functional Dependencies:</u>

- **It is the set of all functional dependencies that can be inferred from F, denoted by F^+ .**
  The notation of F |= X $\rightarrow$ Y denotes that the functional dependencies X $\rightarrow$ Y is inferred from the set of functional dependencies F.

Inference Rules for Functional Dependencies

- **IR1 (Reflexive rule): if X $\supseteq$ Y, then X $\rightarrow$ Y.**
  It says that a set of attributes always determines itself.
- **IR2 (Augmentation rule): {X $\rightarrow$ Y} |= XZ $\rightarrow$ YZ or**
  { X $\rightarrow$ Y} |= XZ $\rightarrow$ Y
  Adding the same set of attributes to both left and right hand sides of a dependency results in another valid dependency.
- **IR3 (Transitive rule) {X $\rightarrow$ Y, Y $\rightarrow$ Z} |= { X $\rightarrow$ Z}**
  Dependencies are transitive.
- **IR4 (Decomposition rule) { X $\rightarrow$ YZ} |= X $\rightarrow$ Y, { X $\rightarrow$ YZ} |= X $\rightarrow$ Z**
  It allows to remove attributes from right hand side of dependencies.
- **IR5 (Union rule) { X $\rightarrow$ Y, X $\rightarrow$ Z} |= X $\rightarrow$ YZ**
  Combine the functional dependencies with same left hand sides.
- **IR6 (Pseudotransitive rule) { X $\rightarrow$ Y, WY $\rightarrow$ Z} |= WX $\rightarrow$ Z**

Proof of Inference Rules

Proof of IR1 (Reflexive rule):

if $X \supseteq Y$, then $X \rightarrow Y$.

Suppose that the $X \supseteq Y$ and that two tuples t_1 and t_2 exist in some relation instance r of R such that $t_1 [X] = t_2 [X]$. Then $t_1 [Y] = t_2 [Y]$ because $X \supseteq Y$; hence $X \rightarrow Y$ must hold in r.

Proof of IR2 (Augmentation rule):

$\{X \rightarrow Y\}$ |= $XZ \rightarrow YZ$ or $\{X \rightarrow Y\}$ |= $XZ \rightarrow Y$

Assume that $X \rightarrow Y$ holds in a relation instance r of R but that $XZ \rightarrow YZ$ does not holds.

Then there must exist two tuples t_1 and t_2 in r such that

(1) $t_1 [X] = t_2 [X]$, (2) $t_1 [Y] = t_2 [Y]$, (3) $t_1 [XZ] = t_2 [XZ]$ and

(4) $t_1 [YZ] \neq t_2 [YZ]$, this is not possible.

From (1) and (3) we deduce (5) $t_1 [Z] = t_2 [Z]$

From (2) and (5) we deduce (6) $t_1 [YZ] = t_2 [YZ]$, contradicting (4).

Proof of Inference Rules

Proof of IR3 (Transitive rule):

$\{X \rightarrow Y, Y \rightarrow Z\}$ |= $\{X \rightarrow Z\}$

Assume that (1) $X \rightarrow Y$ and (2) $Y \rightarrow Z$ both hold in a relation r.

Then for any two tuple t_1 and t_2 such that $t_1[X] = t_2[X]$,

we must have (3) $t_1[Y] = t_2[Y]$ (from assumption (1)), hence we must also have (4) $t_1[Z] = t_2[Z]$, (from assumption (3) and (2));

hence $X \rightarrow Z$ must hold in r.

Proof of IR4 (Decomposition rule):

$\{X \rightarrow YZ\}$ |= $X \rightarrow Y$, $\{X \rightarrow YZ\}$ |= $X \rightarrow Z$

1. $X \rightarrow YZ$ (given)
2. $YZ \rightarrow Y$ (using IR1 and knowing $YZ \supseteq Y$) .
3. $X \rightarrow Y$ (using IR3 on 1 and 2)

Proof of Inference Rules

Proof of IR5 (Union rule):

{ $X \rightarrow Y$, $X \rightarrow Z$} |= $X \rightarrow YZ$
1. $X \rightarrow Y$ (given)
2. $X \rightarrow Z$ (given)
3. $X \rightarrow XY$ (using IR2 on 1 by augmenting with X, and XX = X)
4. $XY \rightarrow YZ$ (using IR2 on 2 by augmenting with Y)
5. $X \rightarrow YZ$ (using IR3 on 3 and 4)

Proof of IR6 (Pseudotransitive rule):

{ $X \rightarrow Y$, $WY \rightarrow Z$} |= $WX \rightarrow Z$
1. $X \rightarrow Y$ (given)
2. $WY \rightarrow Z$ (given)
3. $WX \rightarrow WY$ ((using IR2 on 1 by augmenting with W)
4. $WX \rightarrow Z$ (using IR3 on 3 and 2)

Sound and Complete of Armstrong's Inference Rules

Amstrong's Rule:

- **Inference rule IR1 to IR3 are known as Armstrong's Inference Rules.**
- **It has been shown by Armstrong (1974) that inference rules IR1 to IR3 are sound and complete.**

Sound:

Given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using IR1 to IR3 will hold in every relation instance r of R that satisfies the dependencies in F.

Complete:

It means that by using IR1 to IR3 repeatedly to infer dependencies until no more dependencies can be inferred will result in the complete set of all possible dependencies that can be inferred from F.

In other words, the set of dependencies F^+ , which is called closure of F, can be determined from F by using only inference rules IR1 to IR3.

Functional Dependency Set and Closure

Determination of Full Set of Relevant Functional Dependencies:

- **First specify the set of functional dependencies F that can easily be determined from the semantics of the attributes of R.**
- **Second using Armstrong's inference rules to infer additional functional dependencies that will also hold on R.**
- **Computation of F^+ for a set of dependencies F is a time-consuming.**

Systematic Way to Determine Additional Functional Dependencies:

- **First to determine each set of of of attributes X that appears as left hand side of some functional dependency in F and then use Armstrong's inference rules to determine the set of all attributes that are dependent on X.**
- **For each set of attributes X, we determine the set of X^+ of attributes that are functionally determined by X;**
- **X^+ is called the closure of X under F.**

Algorithm Determining X^+

Algorithm Closure
Input: A set of F of FDs and a set of X of Attributes
Output: $cl_F(X)$

```
X+ := X;
repeat
        old X+ := X+ ;
        for each functional dependencies  Y → Z  in F do
        if  Y ⊆  X+   then  X+ := X+ ∪ Z
until ( old X+ = X+ );
```

- **The algorithm starts by setting X^+ to all the attributes in X by IR1.**
- **It uses inference rules IR3 and IR4, adding attributes to X^+ using each functional dependency in F.**
- **It keeps going through all the dependencies in F until no more attributes are added to X^+ during a complete cycle through the dependencies in F.**

Example

Suppose the functional dependencies specified from the semantics of the attributes as following:

F = {SSN $\rightarrow$ {ENAME},
PNUMBER $\rightarrow$ {PNAME,PLOCATION},
{SSN,PNUMBER} $\rightarrow$ HOURS}

Using the algorithm, the following closure sets w.r.t. F is computed

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

$\{PNUMBER\}^+$ = {PNUMBER, PNAME, PLOCATION}

$\{SSN,PNUMBER\}^+$ = {SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS}

Equivalence of Sets of Functional Dependencies

- A set of functional dependencies E is said to be covered by a set of functional dependencies F, or alternatively F is said to cover E,

  if every FD in E is also in F^+, that is every dependency in E can be inferred from F.

- Two sets of functional dependencies E and F are said to be equivalent if $E^+ = F^+$, the equivalence means that every FD in E can be inferred from F, and FD in F can inferred from E. That is, E is equivalent to F if both E covers F and F covers E hold.

- We can determine whether F covers E by calculating X^+ with respect to F for each FD $X \rightarrow Y \in E$, then checking whether this X^+ includes the attributes in Y.

  If this is the case for every FD in E , then F covers E.

  So checking whether E and F are equivalent by checking that E covers F and F covers E.

Normalization

The Normalization Process:

- It takes a relation schema through a series of test to certify whether or not it belong to a certain normal form.
- E. F. Codd (1972) first proposed three normal forms, 1NF, 2NF, 3NF.
  Boyce and Codd proposed a strong definition of 3NF called BCNF.
- All these normal forms are based on the functional dependencies among the attributes of a relation.

Normalization of Data:

- It is a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties.
- One of the objectives of the original normalization process is to ensure that relation schemas have a good design by disallowing the update anomalies.

Normal Forms and Normalization

Normal Forms Provide Database Designers:

- A formal framework for analyzing relation schemas based on their keys and the functional dependencies among their attributes.

- A series of tests that can be carried out on individual relation schemas so that relational database can be normalized to any degree. When a test fails, the relation violating that test must be decomposed into relations that will individually meet the normalization tests.

Normal Forms

First Normal Form:

- **It was defined to disallow multivalued attributes, composite attributes, and their combinations.**
- **It states that the domains of attributes must include only atomic (single, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.**

Full Functional Dependency:

- **A functional dependency $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more, that is for any attribute $A \in X$, $(X - \{A\}) \nrightarrow Y$.**

Partial Functional Dependency:

- **A functional dependency $X \rightarrow Y$ is a partial functional dependency if there is some attribute $A \in X$ that can be removed from X and the dependency will still hold; that is for some $A \in X$, $(X - \{A\}) \rightarrow Y$.**

  Example:

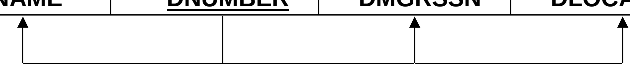
  {SSN, PNUMBER} $\rightarrow$ HOURS is a full functional dependency.

  {SSN, PNUMBER} $\rightarrow$ NAME is partial functional dependency.

Normalization to 1NF

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS



DEPARTMENT

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

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarter			

Normalize Nest Relation to 1NF

EMP_PROJ

SSN	ENAME	PROJS	
		PNUMBER	HOURS

⟹

EMP_PROJ1

<u>SSN</u>	ENAME

EMP_PROJ2

<u>SSN</u>	<u>PNUMBER</u>	HOURS

EMP_PROJ

SSN	ENAME	PNUBMER	HOURS
123456789	Smith, John B	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifier	30	20.0
		20	15.0
888665555	Borg, James E.	20	null

Second Normal Form

<u>Prime Attribute</u>

- **An attribute of relation schema R is called a prime attribute of R if it is a member of any key of R.**

<u>Nonprime Attribute</u>

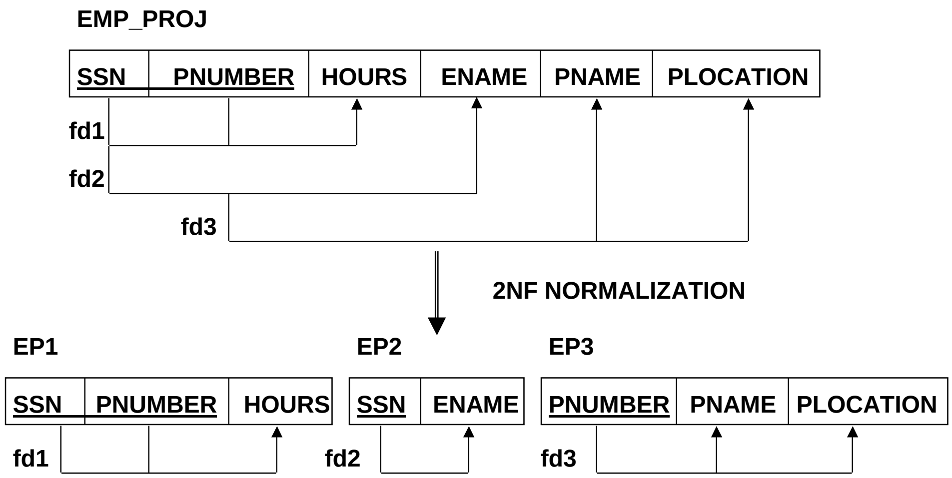
- **An attribute is called nonprime if it is not a prime attribute, i.e., it is not a member of any key of R.**

<u>Second Normal Form:</u>

- **A relation schema is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.**

- **If a relation is not in 2NF, it can be further normalized into a number of 2NF relations so that the nonprime attributes are associated only with the part of primary key on which they are fully functionally dependent.**

Normalization to 2NF



fd1: SSN, PNUMER → HOURS
fd2: SSN → ENAME
fd3: PNUMBER → PNMAE, PLOCATION

Normalization to 3NF

Transitive Dependency:

- A functional dependency X → Y in a relation schema R is a set of attributes Z that is not a subset of any key, and both X → Z and Z → Y hold.

  Example:

  SSN → DNUMBER, DNUMBER → DMGRSSN

  SSN → DMGRSSN is a transitive functional dependency.

Third Normal Form (3NF):

- A relation schema is in 3NF if it is in 2NF and no nonprime attribute of R is transitively dependent on the primary key.
- A relation schema is in 3NF if whenever a functional dependency X → A holds in R, either

  (a) X is a superkey of R, or

  (b) A is prime attribute of R.
- A relation schema R is in 3NF if every nonprime attribute of R is
  - Fully functionally dependent on every key of R, and
  - Nontransitively dependent on every key of R.

Interpreting the General Definition of 3NF

- **If a relation R is in the third normal form, it has the following properties.**
  1. **The relation is in 2NF**
  2. **The nonprime attributes are mutually independent; that is no nonprime attribute is functionally dependent on another nonprime attribute.**

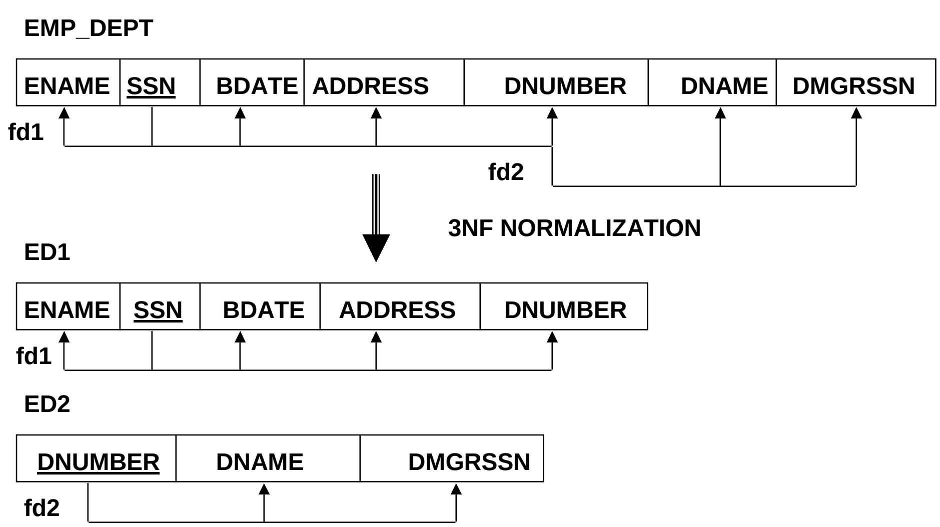
- **A relation schema violates the general definition of 3NF if a functional dependency X $\rightarrow$ Y holds in R that violates both (a) and (b).**
  - **Violating (a) implies that X is not a superset of any key of R, hence X could be nonprime or it could be a proper subset of a key of R.**

    X could be nonprime, it will typically cause transitive dependency

    X could be a proper subset of a key of R, it will cause a partial functional dependency that violates 3NF (and also 2NF)
  - **Violating (b) implies that A is nonprime attribute.**

- **A non 3NF relation can be further normalized to a number of 3NF relations and no nonprime attribute of R is transitively dependent on the primary key.**

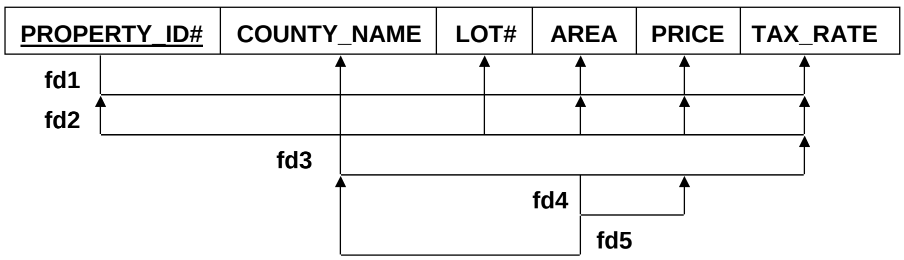
3NF Normalization



fd1: SSN $\rightarrow$ ENAME, BDATE, ADDRESS, DNUMBER
fd2: DNUMBER $\rightarrow$ DNAME, DMGRSSN

Multiple Normalization

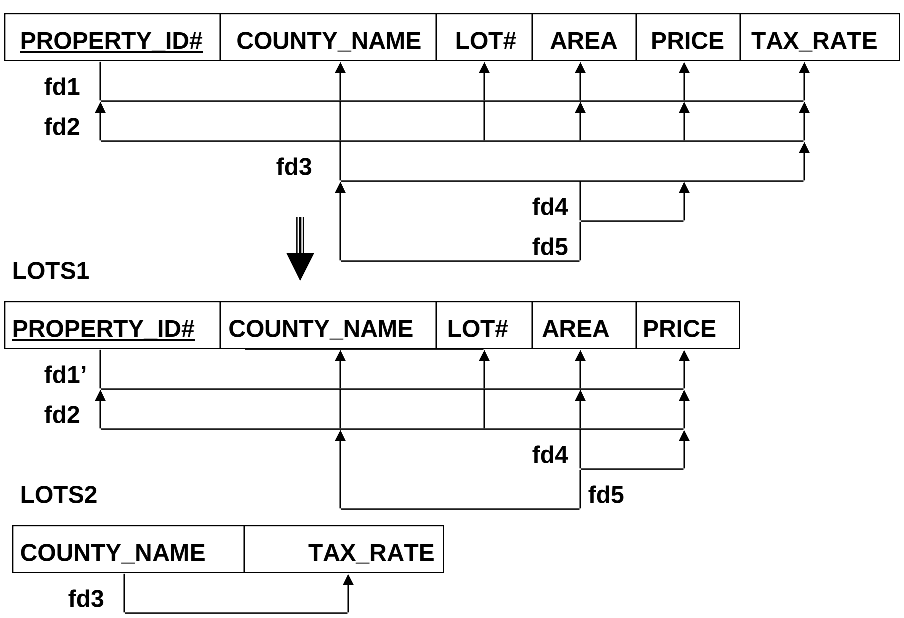
LOTS



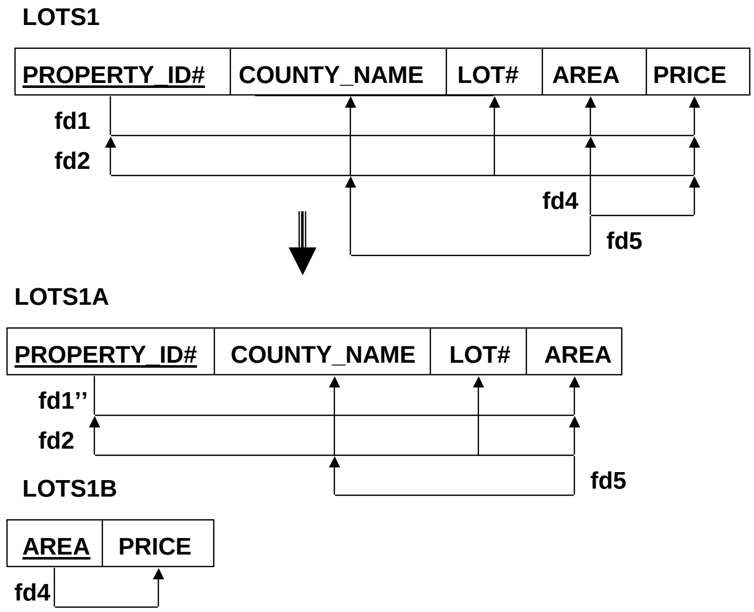
fd1: PROPERTY_ID# → COUNTY_NAME, LOT# , AREA, PRICE, TAX_RATE
fd2: COUNTY_NAME, LOT# → PROPERTY_ID#, AREA, PRICE, TAX_RATE
fd3: COUNTY_NAME → TAX_RATE
fd4: AREA → PRICE
fd5: AREA → COUNTY_NAME
Keys: PROPERTY_ID#
COUNTY_NAME, LOT#

Normalization to 2NF

LOTS



Normalization to 3NF

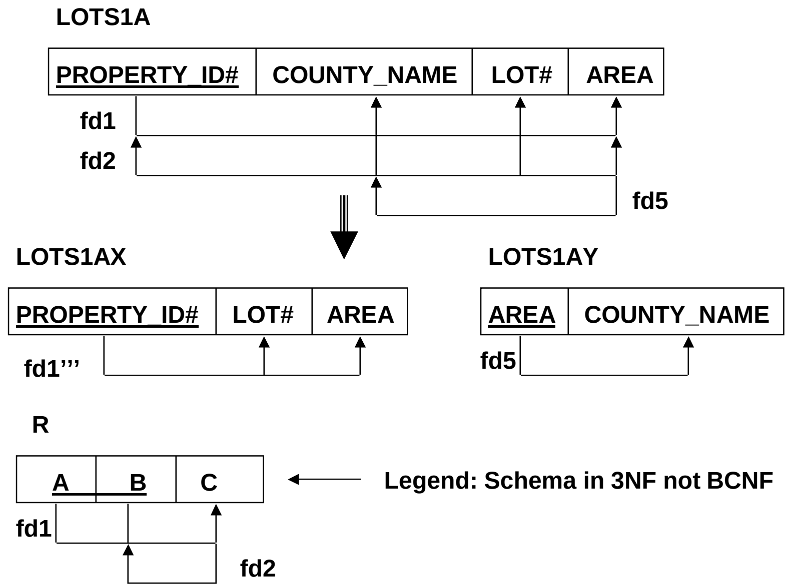


Boyce-Codd Normal Form (BCNF)

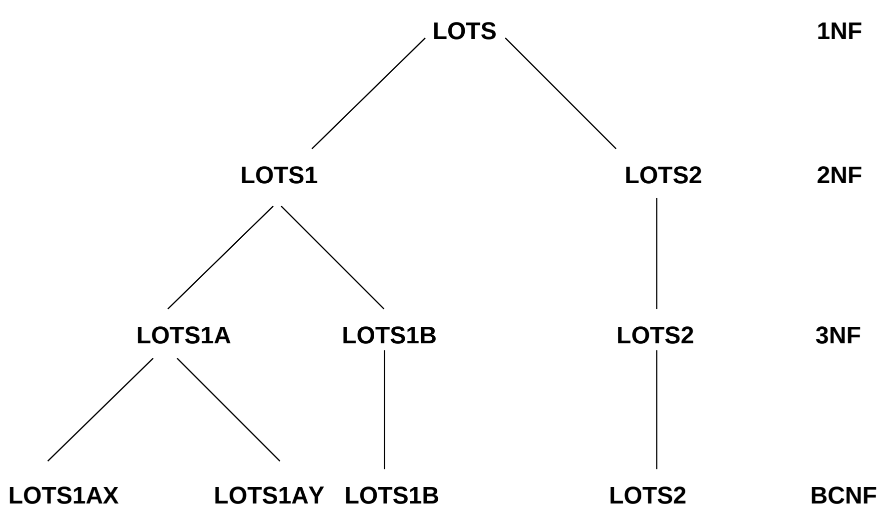
Boyce-Codd Normal Form:

- **A relation schema R is in Boyce-Codd Normal form if whenever a functional dependency X → A holds in R, then X is a superkey of R.**
- **The only difference between BCNF and 3NF is that condition (b) of the definition for 3NF, which allows A to be nonprime if X is not a superkey, is removed.**
- **BCNF is stronger (more restrictive) than 3NF, it means that every relation is in BCNF is automatically in 3NF.**
- **It is best to have relation schemas in BCNF. If that is not possible, 3NF will do.**
- **However, 2NF and 1NF are considered as good relation schema designs. These normal forms were developed historically as stepping stones to 3NF and BCNF.**

Normalization to BCNF



Schema Decomposition Hierarchy



Relational Database Design Method

Techniques for Relational Schema Design:

1. **Top-Down Design Method:**

   Step 1: design a conceptual schema in high level data model, such as E-R model, then map the conceptual schema into a set of relations using mapping procedure.

   Step 2: Informally apply the normalization principles, such as avoiding or detecting transitive or partial functional dependency (dependencies).

2. **Relational Synthesis Method:**

   Relational schemas in 3NF or BCNF are synthesized by grouping the appropriate attribute together.

   a. each individual relation schema should represent a logically coherent grouping of attributes.

   b. each individual relation schema posses the measures of goodness associated with normalization.

Schema Design Algorithm and Criteria

3. **Strict Decomposition:**

   a. start by synthesizing one giant relation schema, called universal schema including all database attributes.

   b. perform the decomposition of the universal schema repeatedly for normalization as long as it is feasible and desirable.

Criteria for a Good Relational Database Schema Design:

1. Each individual relational schema is in either BCNF or 3NF.

2. Decomposition of a relational schema guarantees

   a. dependency preservation.

   b. lossless or non additive join.

Universal Schema and Its Decomposition

Universal Relation Schema:

- **$R = \{A_1, A_2, \ldots, A_n\}$ is a universal relation schema**
- **R includes all the attributes of the database**

Assumptions on Universal Relation Schema:

- **It is assumed that $A_i \neq A_j, \forall\ 1 \leq i, j \leq n$.**
- **Set of functional dependencies that should hold on the attributes of R is specified and made available to the algorithms.**

Decomposition of Universal Schema:

$D = \{R_1, R_2, \ldots, R_m\}$

Attribute Preservation Condition for a Decomposition:

- **each attribute in R will appear in at least one relation schema in the decomposition so that no attributes are lost in the final database schema.**

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

General Forms of Design Algorithms

Input: **1. Universal Relation Schema R**
2. Set of Functional Dependencies F

Output: **Set of Relational schemas $D = \{R_1, R_2, \ldots, R_m\}$,**
D is the decomposition of R.

- **The algorithms will repeatedly decompose the universal relation schema and its decompositions based on the functional dependencies till all relation schemas in BCNF or 3NF.**

Decomposition and Dependency Preservation

Condition of Dependency Preservation:

Each functional dependency $X \rightarrow Y$ specified in F either appear directly in one of the relation schemas R_i in the decomposition D or alternatively be inferred from the dependencies that appear in some individual relation R_i in D.

- It is not necessary that exact functional dependencies specified in F appear themselves in individual relations of the decomposition D.

  It is sufficient that the union of the dependencies that hold on the individual relations in D be equivalent to F.
- If a decomposition is not dependency preserving, some dependency is lost in the decomposition.
- The objective to preserve the dependencies because each dependency in F represents a constraint on database.

  if one of the dependencies is not represented by the dependencies on some individual relation R_i of the decomposition, we will not be able to enforce this constraint on the database by looking at an individual relation.

  In order to enforce the constraint, we will have to join two or more of relations in the decomposition and then check that the functional dependency holds in the result of join operation.

Formal Definition of Dependency Preservation

The Projection of F on R_i:

- Given a set of functional dependencies F on R, the projection of F on R_i, denoted by $\pi_F(R_i)$ where R_i is a subset of R_i, is the set of functional dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i.
- The projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left-hand and right-hand side attributes are in R_i.

Dependency Preservation:

- A decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is dependency preserving with respect to F if the union of the projections of F on each R_i in D is equivalent to F; that is

  $$((\pi_F(R_1)) \cup (\pi_F(R_2)) \cup \ldots \cup (\pi_F(R_m)))^+ = F^+$$

Testing Dependency Preservation

compute F^+ ;
for each schema R_i in D do
  begin
    F_i := the restriction of F^+ to R_i;
  end;
for each restriction F_i do
  begin
    $F' = F' \cup F_i$;
  end
compute F'^+;
if ($F'^+ = F^+$) then return (true)
    else return (false);

- The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include only attributes of R_i.

Dependency-preservation Decomposition into 3NF

Algorithm: (relational synthesis)

Input: set of FD's F and universal schema R
Output: $D = \{R_1, R_2, \ldots, R_m\}$ with dependency preservation, each R_i in 3NF.

1. find a minimal cover of G for F;

2. for each left-hand side of X of a functional dependency that appears in G create a relation schema $\{X \cup A_1 \cup A_2 \cup \ldots \cup A_m\}$ in D where $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_m$ are only dependencies in G with X as left-hand side (X is the candidate key);

3. place any remaining (unplaced) attributes in a single relation schema to ensure the attribute preservation property.

Discussion

- The dependency preservation property is hold in this algorithm.
- It is obvious that all the dependencies in G are preserved by the algorithm because each dependency in G appears in one of the relations R_i in the decomposition.
- Since G is a minimal cover of F, it is equivalent to F and all the dependencies in F are either preserved directly in the decomposition or are derivable from those in the resulting relations.
- This algorithm is called relational synthesis because each relation schema Ri in the decomposition is synthesized from a set of dependencies in G with the left-hand side.

Minimal Sets of Functional Dependency

Minimal Set of FDs:

- A set of functional dependencies F is minimal if it satisfies the following three conditions:

1. Every dependency in F has a single attribute for its right-hand side.
2. We can not remove any dependency from F and still have a set of dependencies that is equivalent to F.
3. We can not replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.

- A minimal set of dependencies as a set of dependencies in a standard or canonical form with no redundancies.
- Condition 1 ensures that every dependency is in a canonical form with a single attribute on the right hand side.
- Condition 2 makes sure there are no redundancies in the dependencies.
- Condition 3 makes sure there are no left hand side redundant attributes.
- A minimal cover of a set of functional dependencies F is a minimal set of dependencies F_{min} that is equivalent to F. There can be several minimal covers for a set of functional dependencies.

Formal Definition of Minimal Set of Functional Dependencies

1. Every right hand side of a functional dependency in F is a single attribute.
2. For no $X \rightarrow A$ in F is the set $F - \{ X \rightarrow A \}$ equivalent to F.
3. For no $X \rightarrow A$ in F and proper subset Z of X is $F - \{ X \rightarrow A \} \cup \{ Z \rightarrow A \}$ equivalent to F.

Example:

Consider the following F set

$A \rightarrow B$, $B \rightarrow A$, $B \rightarrow C$, $A \rightarrow C$, $C \rightarrow A$

$B \rightarrow A$ and $A \rightarrow C$ can be removed, or $B \rightarrow C$ can be removed.

Example:

Consider the following F set

$AB \rightarrow C$, $A \rightarrow B$, $B \rightarrow A$

Attribute either A or B can be removed in $AB \rightarrow C$.

Algorithm to Compute Minimal Set of Functional Dependencies

1. For each functional dependency in the form of $X \rightarrow A_1A_2 \dots A_n$, decompose it into the functional dependencies with single right hand attribute form such that $X \rightarrow A_1$, $X \rightarrow A_2$, ... , $X \rightarrow A_n$.

2. Eliminate any redundant functional dependencies by membership algorithm.

3. Eliminate any redundant attributes in the left hand side of left functional dependencies.

Find a Minimal Cover G for F

Algorithm: Minimal Cover
Input: set of FDs F
Output: Minimal Cover G for F

1. set G := F;
2. replace each functional dependency $X \rightarrow A_1, A_2, \ldots, A_n$ in G by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
3. for each functional dependency $X \rightarrow A$ in G
   {compute X^+ with respect to the set of dependencies (G - ($X \rightarrow A$));
   if X^+ contains A, then remove $X \rightarrow A$ from G};
4. for each remaining functional dependency $X \rightarrow A$ in G
   for each attribute B that is an element of X
   { compute $(X - B)^+$ with respect to the set of functional dependencies
   ((G - ($X \rightarrow A$)) $\cup$ ($(X - B) \rightarrow A$));
   compute X^+ wither respect to FDs in G;
   if $(X - B)^+ \equiv X^+$ then replace $X \rightarrow A$ with $(X - B) \rightarrow A$ in G};

Membership Algorithm

- Given a FD:$A \rightarrow B$ in FD set F and to detect whether or not $A \rightarrow B$ is redundant.

1. Initialize T = A (here T is a variable that contains a set of attributes;
   A is a determinant of a FD in the FD set F.
2. Look at FDs other than $A \rightarrow B$ to see if an FD $X \rightarrow Y$ can be found with its determinant in T (i.e. $X \subseteq T$) . If any such FD $X \rightarrow Y$ is found, then add the attributes in Y to the set of attributes in T (union and transitivity rule).
3. Repeat step 2 every time T is changed until no more attributes can be added to T.
4. If at the conclusion of steps 2 and 3, B is in T, then $A \rightarrow B$ can be derived from the other FDs in F set, so $A \rightarrow B$ is redundant and can be removed from the current FD set F.

* Note step 2 and step 3 is the algorithm to compute the closure A^+ without $A \rightarrow B$.

- Repeat step 2 every time any new attributes are added to T in step 3, and examine al l remaining FDs at each such repetition.

Membership Algorithm

Algortihm FD- Membership
INPUT: Set of FDs F and A → B
OUTPUT: True if A → B is redundant, false if A → B not redundant

```
f = A → B
F = F - f
T = A (T initially contains the determinant→ B)
while (changes to T) do
    for each functional dependency X → Y in F do
      begin
        if X is in T then add Y to T
      end
if B is in T then return TRUE (A → B is redundant)
   else return FALSE (A → B is not redundant)
```


Suppose given a set of FDs as follow:

Z → A B → X AX → Y ZB → Y

First, consider Z → A:

Step 1: T = Z

Step 2: Nothing is added to T, as there is no other FD X → Y where X ⊆ T.

Hence Z → A is not redundant. The FDs B → X and AX → Y can be shown to be nonredundant in a similar way.

Now consider ZB → Y:

Step 1: T = ZB

Step 2: T = ZB ∪ A = ZBA

because Z → A is in the remaining set of FDs and Z ⊆ T.

T = ZBA ∪ X = ZBAX because B → X is in the remaining set of FDs and B ⊆ T.

T = ZBAX ∪ Y = ZBAXY because AX → Y is in the remaining set of FDs and AX ⊆ T. Now Y is in T. So ZB → Y is redundant.

Decomposition and Lossless Join

Lossless (Nonadditive) Joins:

- **A decomposition D = {R_1, R_2, ... , R_m} of R has the lossless (nonadditive) join property with respect to the set of dependencies F on R if for every relation instance r of R that satisfies F, the following is hold:**

  $((\pi_{<R1>} (r)) * (\pi_{<R2>} (r)) * .... * (\pi_{<Rm>} (r))) = r$

  where * is the natural join operation.

- **The word "lossy join" refers to the loss of information, not loss of tuples.**
- **If a decomposition does not have lossless join property, the spurious tuples may generated after natural join operation is applied.**
- **if the lossless join property holds on a decomposition, then no spurious tuples bearing wrong information are added to the result after natural join is applied.**

Testing for the Lossless Join Property

Algorithm: TLJP

Input: R, D = {R_1, R_2, ... , R_m}, F
Output: matrix S which give the result of testing

1. **create a matrix S with one row i for each relation Ri in the decomposition D, and one column j for each attribute A_j in R;**
2. **set S(i,j) := $b_{i,j}$ for all matrix entries;**
   (* each $b_{i,j}$ is a distinct symbol associated with indices (i,j) *)
3. **for each row i representing relation schema R_i**
   for each column j representing attribute A_j
   if Ri includes attributes A_j then set S(i,j) = a_j;
   (* each aj is a distinct symbol associated with index (j) *)

4. repeat the following until a loop execution results in no changes to S

   for each functional dependency X → Y in F

   for all rows in S which 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 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 appear in one of the rows for the attribute and set the other rows to that "b" symbol in the column;

5. if a row is made up entirely of "a" symbols, then the decomposition has the lossless join property;

   otherwise, it does not;

Discussion

- The algorithm creates a relation instance r in the matrix S that satisfies all the functional dependencies in F.
- At the end of the loop of applying functional dependencies, any two rows in S - which represent two tuples in r - that agree in their values for the left-hand side attributes of a functional dependency X → Y in F will also agree in their values for the right-hand side attributes of the dependency.

  Hence S satisfies all the functional dependencies.
- It has been proved that

  if any row in S ends up with all "a" symbols at the end of the algorithm, then the decomposition has the lossless join property with respect to F.

  if, on the other hand, no row ends up being all "a" symbols, then the relation instance r of R that satisfies the dependencies in F but does not have the lossless join property.

Example of Loss Join Decomposition

R = {SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS}

D = {R1, R2}

R1 = EMP_LOCS = {ENAME, PLOCATION}

R2 = EMP_PROJ = {SSN, PNUMBER, HOURS, PNAME, PLOCATION}

FDs: SSN $\rightarrow$ ENAME;
PNUMBER $\rightarrow$ PNAME, PLOCATION
SSN,PNUMBER $\rightarrow$ HOURS

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R2	a_1	b_{22}	a_3	a_4	a_5	a_6

(no changes to matrix after applying functional dependencies)

Example of Lossless Join Decomposition

EMP

SSN	ENAME

PROJECT

PNUMBER	PNAME	PLOCATION

WORKS_ON

SSN	PNUMBER	HOURS

R = {SSN,ENAME, PNUMBER, PNAME, PLOCATION, HOURS}

D = {R1, R2, R3}

R1 = EMP = {SSN, ENAME }

R2 = PROJECT = {PNUMBER, PNAME, PLOCATION }

R3 = WORKS_ON = {SSN, PNUMBER, HOURS }

FDs: SSN $\rightarrow$ ENAME;
PNUMBER $\rightarrow$ PNAME, PLOCATION
SSN,PNUMBER $\rightarrow$ HOURS

Application of The Testing Algorithm

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(original matrix S at start of algorithm)

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(matrix S after applying the first two functional dependencies - last row is all "a" symbols, so we stop).

Properties

Property LJ1:

A decomposition D = $\{R_1, R_2\}$ of R has lossless join property with respect to a set of functional dependencies F on R if and only if either:

- **The FD $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ is in F^+, or**
- **The FD $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ is in F^+ .**

Property LJ2:

If a decomposition D = $\{R_1, R_2, \ldots, R_m\}$ of R has the lossless join property with respect to a set of functional dependencies F on R, and if a decomposition $D_1 = \{ Q_1, Q_2, \ldots, Q_k \}$ of R_i has the lossless property with respect to the projection of F on Ri , then the decomposition

$D_1 = \{ R_1, R_2, \ldots, R_{i-1}, Q_1, Q_2, \ldots, Q_k, R_{i+1}, R_{i+2} \ldots, R_m\}$

of R has the lossless join property with respect to F.

- **Property LJ2 says that if a decomposition D already has the lossless join property - with respect to F - and we further decompose one of the relation schemas Ri in D into another decomposition D_1 that also has the lossless join property - with respect to $\pi_F(R_i)$ - then replacing R in D by D_1 will result in a decomposition that also has the lossless join property ω.ρ.τ. F.**

Lossless Join Decomposition into BCNF

Algorithm:

Input: R and F

Output: $D = \{R_1, R_2, \ldots, R_m\}$ and each R_i in BCNF

1. set $D \leftarrow \{R\}$;

2. while there is a relation schema Q in D that is not in BCNF do
   begin
   choose a relation schema Q in D that is not in BCNF;
   find a functional dependency $X \rightarrow Y$ in Q that violates BCNF;
   replace Q in D by two schemas (Q - Y) and $(X \cup Y)$;
   end;

   By the property LJ1 and LJ2, the decomposition D will always have the lossless join property.

   At the end of algorithm, all relation schemas in D will be in BCNF.

Lossless Join and Dependency-Preserving Decomposition into 3NF Relation Schemas

Algorithm:

Input: R and F

Output: D in 3NF with lossless join property

1. find a minimal cover G for F;
   (* F is the set of functional dependencies specified on R*)

2. for each left-hand side X that appear in G
   create a relation schema $\{ X \cup A_1 \cup A_2 \cup \ldots \cup A_m \}$ where
   $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_m$ are all the dependencies in G with X as left-hand side;

3. place all remaining (unplaced) attributes in a single relation schema;

4. if none of the relation schemas contains a key of R, create one more relation schema that contains attributes that form a key for R;

Find a Key K for Relation Schema R

1. set K := R;

2. for each attribute A in K

   {compute (K- {A})$^+$ with respect to the given set of functional dependencies;

   if (K - {A})$^+$ contains all the attributes in R, then set K:= K - { A } };

Example

Consider the relation schema CTHRSG, where C = course, T = teacher, H = hour, R = room, S= student, and G = grade.
The minimal set of functional dependencies F are assumed:

C $\rightarrow$ T each course has one teacher
HR $\rightarrow$ C only one course can meet in a room at one time.
HT $\rightarrow$ R a teacher can be in only one room at one time.
CS $\rightarrow$ G each student has one grade in each course.
HS $\rightarrow$ R a student can be in only one room at one time.

The decomposition of R based on the algorithm is R = {R1, R2, R3, R4, R5}

R1 = {CT} R2 = {HRC} R3 = {HTR} R4 = {CSG} R5 = {HSR}

HS is the key of R, and is a part of R5.

Problems with Null Values

<u>Null Values in Designing a Relational Database Schema:</u>

- **There is no fully satisfactorygiven if the NULL values appear in the join attribute column.**
- **Usually, the join operations is necessary because the decompositions of the relation schema in the normalization process, or the implementation of relationships inherited from ER schema of the database.**
- **Natural join will not combine these tuples with NULL values on the join attributes into the result relation.**
- **Outer-Join will combine these tuples with NULL values on the join path with padding the NULL values for the rest of the attributes in another relation.**
- **Potential watching should be given for the NULL values in the foreign key attributes which define the referential integrity constraints.**
- **If the NULL values appear in the numerical attribute column attribute, application of aggregate functions such as sum(), avg(), and count() should be given careful attention and evaluation.**

Dangling Tuples

<u>Dangling Tuples:</u>

- **If a tuple does not contribute to the join, then we say that it is *dangling*.**

<u>Complete Join:</u>

- **If neither of two relations contains dangling tuples, then the join is called a *complete join*.**

Example:

A	B
a_1	b_1

(a.) R_1

B	C
b_1	c_1
b_2	c_2

(b.) R_2

A	B	C
a_1	b_1	c_1

(c.) $R_1 * R_2$

- $\pi_{BC}(R_1 * R_2) \neq r_2(R_2)$
- $r_1(R_1) = \{a_1b_1\}$, $r_2(R_2) = \{b_1c_1, b_2c_2\}$, and $r_{12}(R_1 * R_2) = \{a_1b_1c_1\}$

EMPLOYEE

ENAME	SSN	BDATE	ADDRESS	DNUMBER
John Smith	123456789	09-JAN-55	731 Fondren, Houston, TX	5
Franklin Wong	333445555	08-DEC-45	638 Voss, Houston, TX	5
Alicia Zelaya	999887777	19-JUL-58	3321 Castle, Spring TX	4
Jennifer Wallace	987654321	19-JUN-31	291 Berry, Bellaire, TX	4
Ramesh Narayan	666884444	15-SEP-52	975 FireOak, Humble, TX	5
Joyce English	453453453	31-JUL-62	5631 Rice, Houston, TX	5
Ahmad Jabbar	987987987	29-MAR-59	980 Dallas, Houston, TX	4
James Borg	888665555	10-NOV-27	450 Stone, Houston, TX	1
Anders Berger	999775555	26-ARP-55	6530 Braes, Bellaire, TX	null
Carlos Benitez	888664444	09-JAN-53	7654 Beech, Houston, TX	null

EMPLOYEE_1

ENAME	SSN	BDATE	ADDRESS
John Smith	123456789	09-JAN-55	731 Fondren, Houston, TX
Franklin Wong	333445555	08-DEC-45	638 Voss, Houston, TX
Alicia Zelaya	999887777	19-JUL-58	3321 Castle, Spring TX
Jennifer Wallace	987654321	19-JUN-31	291 Berry, Bellaire, TX
Ramesh Narayan	666884444	15-SEP-52	975 FireOak, Humble, TX
Joyce English	453453453	31-JUL-62	5631 Rice, Houston, TX
Ahmad Jabbar	987987987	29-MAR-59	980 Dallas, Houston, TX
James Borg	888665555	10-NOV-27	450 Stone, Houston, TX

EMPLOYEE_2

ENAME	SSN
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

EMPLOYEE_3

ENAME	SSN
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	null
888664444	null