Module 4 — [redacted] part 1

* FUNCTIONAL DEPENDENCIES AND NORMALIZATION FOR RELATIONAL Databases

vvimp *** State informal guidelines for relational schema design. illustrate how violation of these guidelines may be harmful.

There are four informal measures of quality for relation schema design

Relation

1. Semantics of the attributes.
2. Reducing the redundant values in tuples.
3. Reducing the null values in tuples.
4. Disallowing the possibility of generating spurious tuples.

Semantics of the relation Attributes

Semantics specifies how to interpret the attribute values stored in a tuple of the relation, in other words how the attribute values in a tuple relate to one another.

If the conceptual design is done carefully, followed by a mapping into relations.

Eg EMP(ENO, ENAME, SAL, FK DNO) DEPT(DNO, DNAME)

The meaning of EMP is clear, each tuple represents an employee with the above mentioned attributes. The DNO is a foreign key that represents an implicit relationship between EMPLOYEE & DEPT.

DEPT contains each entity of type DNO & DNAME.

contents of the table will be

EMP

ENO	NAME	SAL	DNO
123	Adarsh	15000	1
124	Chetan	20000	2
125	Bharath	25000	1

DEPT

DNO	DNAME
1	CS
2	IS

Hence the schema may be considered as good design.

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

Eg EMP_DEPT (ENO , NAME , SAL , DNO , DNAME)

It represents employee but includes additional information namely DNAME.

EMP_DEPT

ENO	NAME	SAL	DNO	DNAME
123	Adarsh	15000	1	CS
124	Chetan	20000	2	IS
125	Bharath	25000	1	CS

→ Logically it is not wrong, they are considered poor designs because it violates guidline 1.

Imp Discuss insertion, deletion & modification anamolies. Why are they considered bad? Illustrate with an Example for each.

one goal of schema design is to minimise the storage space that the base relation occupy.

Grouping attributes into relation schemas has a significant effect on storage space

Space used by EMP & DEPT as compared with EMP-DEPT is less.

Another serious problem with using relation EMP-DEPT as base relation is the problem of update anomalies. These can be classified into insertion anomalies, deletion anomalies & modification anomalies.

Insertion Anomalies There can be differentiated into two types.

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

Eg: to insert a new tuple for an employee who works in dept no 2, we must enter attribute values of department 2 correctly so that they are consistent with values for dept 2 in other tuples.

→ It is difficult to insert a new department that has no employees. The only way to do this is to place null values in the attributes for employee.

This causes a problem because ENO is primary key.

EMP_DEPT

OR

127 Chaitra 19000 NULL NULL

insert a new department

NULL NULL NULL 3 IT

↑
PK.

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 db.

This problem will not occur if it is saved saperately.

Delete 128 Ranjan. 28000 3 IT

information regarding DNO 3 will also be lost.

Modification Anomalies

In EMP_DEPT, if we change the value of one of the attribute of a particular department say from 'CS' to 'comp sci' then we must update the tuples of all employees who work in that dep Otherwise the db will become inconsistent.

GUIDELINE 2

Design the base relation schema so that no insertion, deletion or modification anomalies are present in the relation. If any anomalies are present, note them clearly & make sure that the programs that update the database will operate correctly.

** 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.

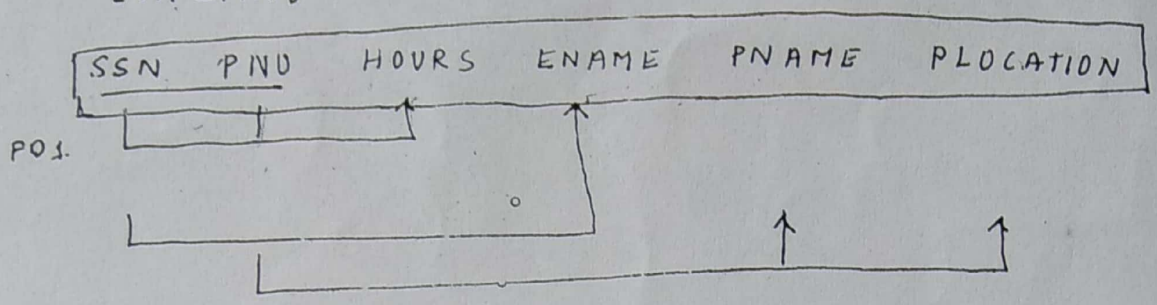
→ This can waste space at the storage level
→ problem with meaning of the attributes
→ specifying JOIN operation at the logical level.
→ Another problem with nulls is how to account for them when aggregate operation such 'SUM' are applied or COUNT.

Nulls can have multiple interpretations such as → The attribute does not apply to this tuple.
→ The value is unknown

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

Generation of Spurious tuples.

EMP_PROJ



123	1	25	Adarsh	ISRO	BNG
124	2	15	Amith	IISC	BNG

EMP_LOCS

ENAME	PLOCATION
Adarsh	BNG
Amith	BNG

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATI
123	1	25	ISRO	BNG
124	2	15	IISC	BNG

Consider the two relation schemas EMP_LOCS & EMP_PROJ1 which can be used instead of EMP_PROJ

A tuple in EMP_LOCS means that the employee whose name is ENAME works on some project whose location is PLOCATION.

A tuple in EMP_PROJ1 means that the employee whose social security NO is SSN works Hours per week on the project whose name, number, & location are PNAME, PNUMBER & PLOCATION.

Suppose that we used EMP_PROJ1 & EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design, bec

we cannot recover the information that was orginally in EMP_PROJ from EMP_PROJ1 & EMP_LOCS.

If we attempt a NATURAL JOIN operation on EMP_PROJ1 & EMP_LOCS, the result produces many more tuples than the orginal population of tuples in EMP_PROJ.

EMP_LOCS * EMP_PROJ1

	ENAME	PLOCATION	SSN	PNUMBER	HOURS	PNAME	PLOCATION
✓	Adarsh	BNG	123	1	25	ISRO	BNG.
*	Adarsh	BNG	124	2	15	IISC	
*	Amith	BNG	123	1	25	ISRO	
✓	Amith	BNG	124	2	15	IISC	

Additional tuples that were not in EMP_PROJ are called spurious tuples because they represent spurious or wrong information that is not valid. The spurious tuples are marked by astcrisks (*).

Decomposing EMP_PROJ into EMP_LOCS & EMP_PROJ1 is undesirable. This is because in this case PLOCATON is the attribute that relates EMP_LOCS & EMP_PROJ1, & PLOCATION is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.

GUIDELINE4: DESIGN relation schemas so that they can be JOINed with equality condition on attributes that are either primary key or foreign key

in a way that guarantees that no spurious tuples are generated. Do not have relations that contain matching attributes other than foreign key - primary key combinations. If such relations are unavoidable, do not join them on such attributes, because the join may produce spurious tuples.

Imp Discuss the problem of spurious tuples and how we may prevent it?

Imp Functional Dependencies

A functional dependency is a constraint between two sets of attributes from the database.

Suppose that our relational database schema has n attributes $A_1, A_2, \ldots, A_n$. Let us think of the whole db as

Schema $R = \{A_1, A_2, \ldots, A_n\}$.

A functional dependency denoted by $X \rightarrow Y$, between two sets of attributes X & Y that are subset 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 t_1 & t_2 in r that have $t_1[X] = t_2[X]$, we must also have $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on, or are determined by, the value of the X component

or

The value of x component of a tuple uniquely determine the value of the Y component.

$$X \longrightarrow Y$$

Thus X functionally determines Y in a relation schema R if & 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 cannot be more than one tuple with a given x-value in any relation instance r(R) - ie, X is a candidate key of R this implies that $X \longrightarrow Y$ for any subset of attributes Y of R.

If $X \longrightarrow Y$ in R this does not say whether or not $Y \longrightarrow X$ in R.

Ex in EMP_PROJ relation schema the following functional dependencies should hold

a. SSN $\longrightarrow$ ENAME

b. PNUMBER $\longrightarrow$ {PNAME, PLOCATION}

c. {SSN, PNUMBER} $\longrightarrow$ HOURS.

Inference Rules for Functional Dependencies.

F the set of FD that are specified on relation schema R. schema designer will indicate it but several outher FD hold in all legal relation instances that satisfy the dependencies in F. Those other FD can be infered or deduced.

*** The set of all such dependencies ~~are~~ is called closure of F & is denoted by F^+.

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

$DNUMBER \rightarrow \{DNAME, DMGRSSN\} \}.$

We can infer or deduce additional FD from F

$SSN \rightarrow \{DNAME, DMGRSSN\}$

$SSN \rightarrow SSN,$

$DNUMBER \rightarrow DNAME$

$F \models X \rightarrow Y$ denote that the FD $X \rightarrow Y$ is infered from the set of functional dependencies F.

*** The following six rules are well-known inference rules for functional dependencies.

IR1 (reflexive rule): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (augmentation rule): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

IR4 (decomposition, or projective rule): $\{X \rightarrow YZ\} \models X \rightarrow Z$

IR5 (union, or additive, rule): ~~$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$~~

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ.$

IR6 (pseudo transitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z.$

* It has been shown by Armstrong (1974) that inference rules IR1 through IR3 are sound & complete.

By using IR1 - IR3 holds in every relation state r of R that satisfies the dependencies in F.

COMPLETE - given set of all possible dependencies that can be inferred from F.

F+ can be determined by using inference rules IR1 through IR3 are known as ARMSTRONG inference rules.

** Data Base designers first specify the set of FD F that can be easily determined from the semantics of the attributes of R.

Then IR1, IR2 & IR3 are used to infer additional functional dependencies that will hold on R.

A systematic way to determine these additional functional dependencies is first to determine each set of attributes X that appear as a left-hand side of some FD in F & then to determine the set of all attributes that are dependent on X.

Thus for each such set of attributes X, we determine the set X+ of attributes that are FD by X based on F; X+ is called the "closure of X under F

Imp
***

Algorithm

ALGORITHM: Determining X^+, the closure of X under F

$X^+ = X$;
repeat
    $oldX^+ := X^+$;
    for each functional dependency Y → Z in F do
        if $X^+ \supseteq Y$ then $X^+ = X^+ \cup Z$
until $(X^+ = oldX^+)$;

Starts by setting X^+ to all the attributes in X. By IR1, we know that all these attributes ~~in X by IR1~~ are functionally dependent on X. Using inference rules IR3 & IR4, we add attributes to X^+ using each FD in F. We keep going through all the dependencies in F until no more attributes are added to X^+.

F = { SSN → ENAME, PNO → { PNO, PNAME, PLOCATION },
{ SSN, PNO } → HOURS }.

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

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

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

Let F be a set of FD on a relation schema R.
Define the following
i) F^+ the closure of F ii) X^+ the closure of X under F.

Equivalence of Sets of Functional Dependencies

A set of functional dependencies E is covered by a set of functional dependencies F. If every FD in E is also in F^+

ie

"if every dependency in E can be infered from F"

Two sets of functional dependencies E & F are equivalent if $E^+ = F^+$

Hence equivalence means that every FD in E can be infered from F, & every FD in F can be infered from E. E is equivalent to F, if both the condition E covers F & F covers E hold.

what is a set of FD F said to be minimal? give an algorithm for finding a minimal cover G for F

Minimal set of Functional Dependencies.

A set of functional dependencies F is minimal if it satisfies the following condition.

1. Every dependency in F has a single attribute for its right-hand side.
2. we cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, & still have a set of dependencies that is equivalent to F.

3. we cannot remove any dependency from F & still have a set of dependencies that is equivalent to F.

A minimal cover of a set of functional dependencies F is a minimal set of dependencies Fmin that is equivalent to F.

Algorithm: Finding a 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
   for each attribute B that is an element of X
   if $((G - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\})$ is equivalent to G,
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in G.
4. For each remaining functional dependency $X \rightarrow A$ in G.
   if $(G - \{X \rightarrow A\})$ is equivalent to G,
   then remove $X \rightarrow A$ from G.

Eg 1

R = { sno, sname, cno, cname, instructor, address, office }

F = { sno → sname, cno → cname, sno → address, cno → instructor, instructor → office }

{ sno, cno }⁺ = sno, cno

sno, cno, sname (sno → sname)

sno, cno, sname, cname (cno → cname)

sno, cno, sname, cname, address (sno → address)

sno, cno, sname, cname, address, instructor (cno → instructor)

{ sno, cno }⁺ = { sno, cno, sname, cname, address, instructor, office }

(instructor → office)

Eg 2

Student (sno, sname, cno, cname)

F = { sno → sname, cno → cname }

{ sno, cno }⁺ = sno, cno

sno, cno, sname (sno → sname)

sno, cno, sname, cname (cno → cname)

{ sno, cno }⁺ = { sno, cno, sname, cname }

sno & cno is a candidate key because if sno & cno is indicated, the entire tuple will be retrieved.

Eg3 $R = (A, B, C, D, E)$

$F = \{ A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A \}$

Compute A^+ & B^+

A^+ A

ABC

ABCD

ABCDE

B^+ B

BD

4) Consider the universal relation $R = \{A, B, C, D, E, F, G, H, I, J\}$ & a set of functional dependencies

$F = \{ AB \rightarrow C,\ A \rightarrow DE,\ B \rightarrow F,\ F \rightarrow GH,\ D \rightarrow IJ \}$

Find the following

i) Key for R.

AB^+ AB

ABC

ABCDE

ABCDEF

ABCDEFGH

$AB^+ = \{ ABCDEFGHIJ \}$

B^+ B

BF

$B^+ = \{ BFGH \}$

$F^+ = \{FGH\}$

D^+ D

$D^+ = \{DIJ\}$

A^+ A

ADE

$A^+ = \{ ADEIJ \}$

only AB will identify all the attributes

AB is the Key.

5) Consider the following two sets of functional dependencies.

$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ &

$G = \{A \rightarrow CD, E \rightarrow AH\}$ check whether they are equivalent

They are equivalent if $F^+ = G^+$.

with F

A^+ A
AC
ACD

AC^+ AC
ACD
ACDE
ACDEH

E^+ E
EAD
ACDE
ACDEH

E^+ E
EH
ADEH
ACDEH

ACD, ACDEH

with G.

A^+ A
ACD

E E
AHE
ACDEH

ACD ACDEH

They are equivalent.

Introduction to Normalization

The normalization process as first proposed by codd

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.

What is the need for Normalization?

Normalization of data can hence be looked upon as a process of analyzing the given relation schemas based on their FD's and primary keys to achieve the desirable properties of 1) Minimizing redundancy
2) Minimizing the insertion, deletion & update anomalies.

Normal forms when considered in isolation from other factors, do not guarantee a good database design. The other factors include two properties

- The lossless join or nonadditive join property. which guarantees that the spurious tuple generation problem
- The dependency preservation property which ensures that each FD is represented in some individual relations resulting after decomposition

The process of storing the join of higher normal form relations as a base relation which is in a lower normal form is known as denormalization

An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.

An attribute of relation is called nonprime if it is not a prime attribute ie if it is not a meriber of any candidate key.

WORKS-ON (SSN, PNO, Hours)

SSN & PNO are prime attribute

First Normal Form

it states that the domain of an attribute must include only atomic values and that the value of any attribute in a ruple must be a single value from the domain of that attribute.

so it disallows multivalued attributes, composite attributes & their combinations.

Eg DEPT

DNAME	DNO	SSSN	DLOCATION
CS	1	1234	{Bng, Mys, mangalore}
IS	2	1257	{Bng}

we assume that each department can have a no of locations. This is not in INF since DLOCATION is not an atomic attribute.

There are three main techniques to achieve First normal form

1. Remove the attribute DLOCATION that violates INF & place it in a separate relation

DEPT_LOCATION

DNO	DLOCATION
1	Bng
1	Mys
1	Mangalore
2	Bng.

DEPT

DNAME	DNO	SSSN
CS	1	1234
IS	2	1257

2. Expand the Key so that there will be a separate tuple in the orginal DEPT

DEPT

DNAME	DNO	SSSN	DLOCATION
CS	1	1234	Bng
CS	1	1234	Mys
CS	1	1234	Mangalore
IS	2	1257	Bng

primary Key {DNO, DLOCATION} This solution has the disadvantage of introducing redundancy in the relation.

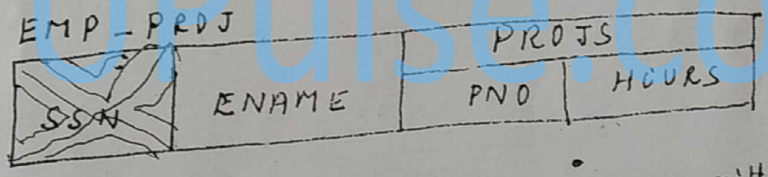
3. If a maximum no of value is known for the attribute ex it is know that DLOCATION can have atmost three locations. Replace Dlocation with DL1, DL2 & DL3.

DEPT	DNAME	DNO	SSSN	DL1	DL2	DL3
	CS	1	1234	Bng	Mys	Mangalore
	IS	2	1257	Bng	NULL	NULL

Disadvantage :- introduces null values

Of the three solutions first is superior because it does not suffer from redundancy.

The first normal form also disallows multivalued attributes that are themselves composite. They are called nested relations because each tuple can have a relation within it.



The above relation can be written as

EMP_PROJ1
(SSN , ENAME)

EMP_PROJ2
(SSN, PNO , HOURS).

Secound Normal Form

is based on the concept of full functional dependency.

A functional dependency $X \rightarrow Y$ is a full functions dependency. if removal of any attribute A from X means

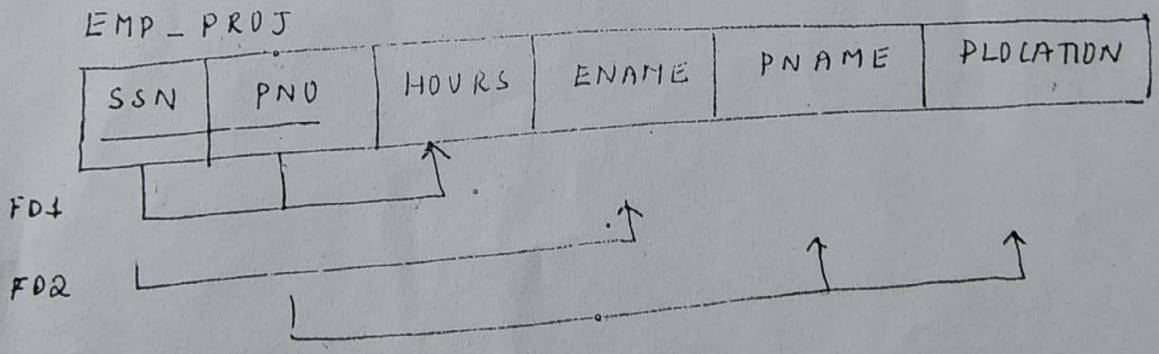
A functional dependency $X \rightarrow Y$ is partial dependency if some attribute $A \in X$ can be removed from X & dependency still holds.

$\{SSN, PNO\} \rightarrow HOURS$ is fully dependent

$\{SSN, PNO\} \rightarrow ENAME$ is partial because

$SSN \rightarrow ENAME$ holds.

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



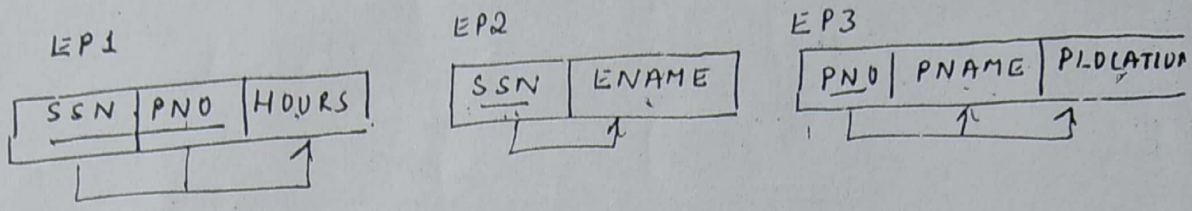
$\{SSN, PNO\} \rightarrow HOURS$

$SSN \rightarrow ENAME$ so ENAME is not full dependent

$PNO \rightarrow \{PNAME, PLOCATION\}$

PNO is not full dependent on the key.

All the partial dependency of EMP_PROJ are indicated as saperate tables

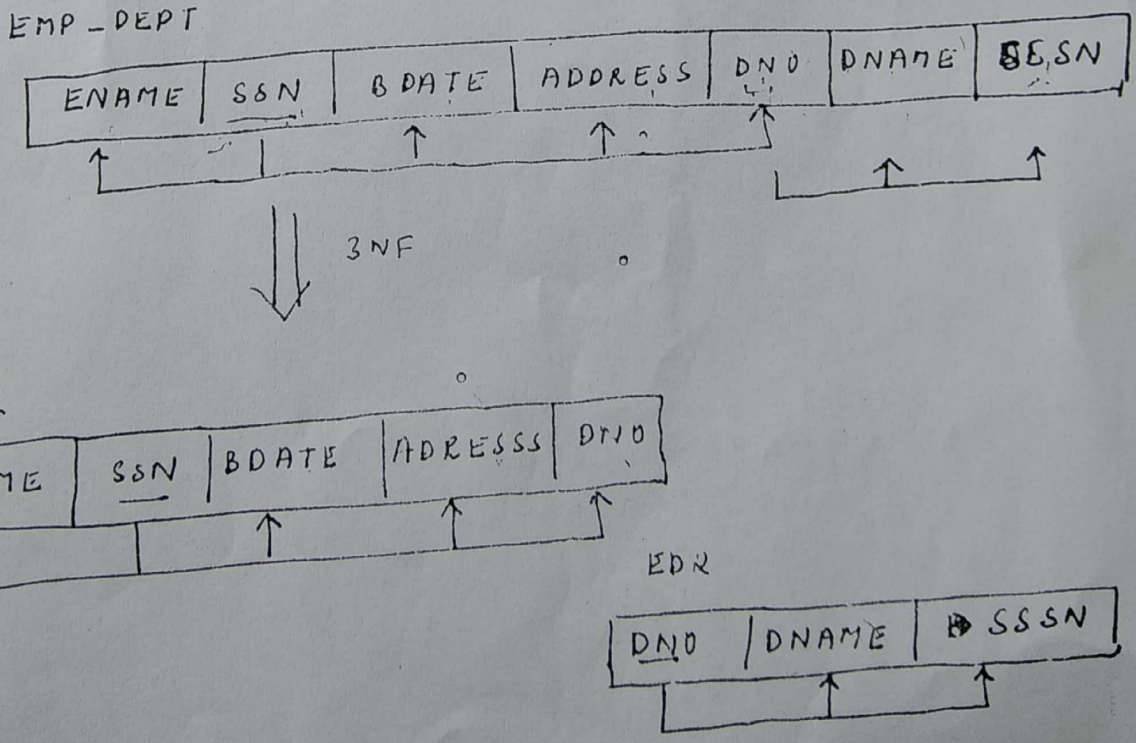


Third Normal Form

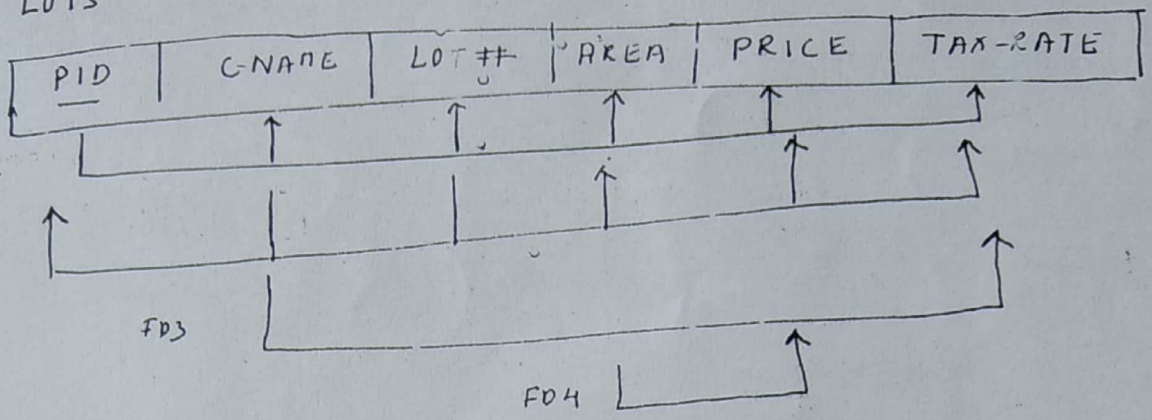
is based on the concept of transitive dependency

A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, & both $X \rightarrow Z$ & $Z \rightarrow Y$ hold.

A relation schema R is in 3NF if it satisfies 2NF & no nonprime attribute R is ~~in 3NF~~ transitively dependent on the primary key.



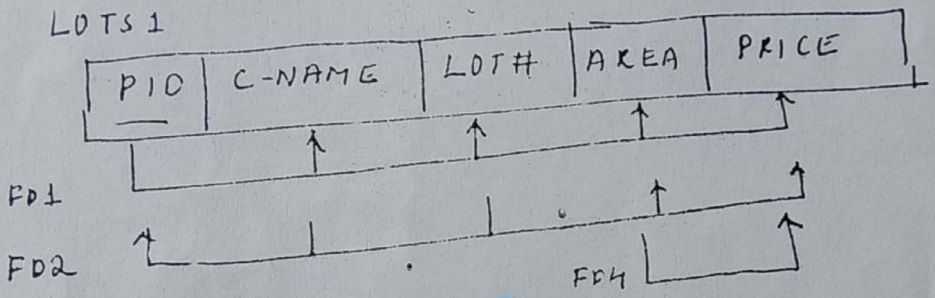
LOTS



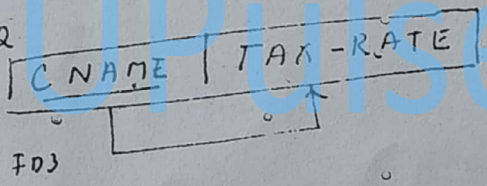
Keys are {PID} {CNAME, LOT#}

2NF

LOTS 1

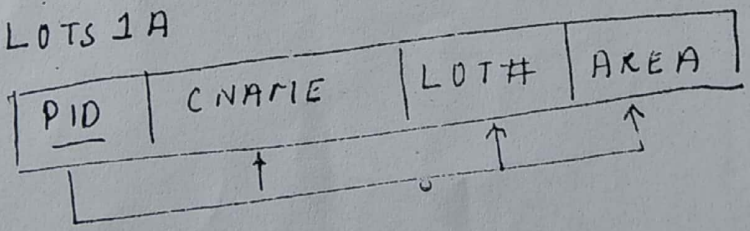


LOTS 2

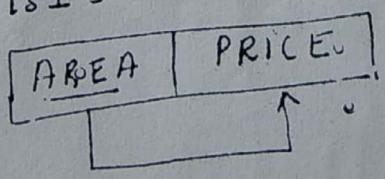


3NF

LOTS 1A



LOTS 1B



General Definition of Second Normal Form

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

General Definition of Third Normal Form (3NF)

A relation schema R is in 3NF if, whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, either

a) X is a super key of R or

b) A is a prime attribute.

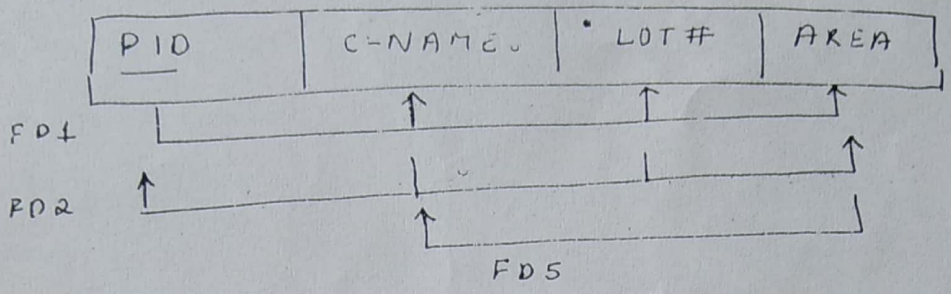
BOYCE - CODD Normal form

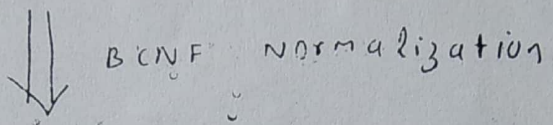
was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF, because every relation in BCNF is also in 3NF; however a relation in 3NF is not necessarily in BCNF.

"A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then X is a super key of R."

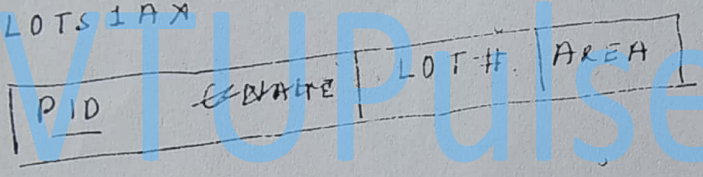
*** The only difference between the defenition of BCNF & 3NF is that condition b) of 3NF which allows A to be prime is absent from BCNF..

LOTS1A

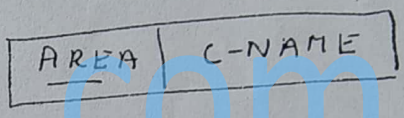




LOTS1AX



LOTS1AY



Module #4 - part 2

Algorithms for Relational Database Schema Design

1) Relation Decomposition & Insufficiency of Normal Forms

The relational database design algorithms that we present here start from a single universal relation schema

$R = \{A_1, A_2, \ldots, A_n\}$ that includes all the attributes of the database.

The set F of functional dependencies that should hold on the attribute ~~name is~~ unique. of R is specified by the database designers & is made available to the design algorithm.

Using FD & decomposition algorith R can be a set of relations schemas $D = \{R_1, R_2, \ldots, R_m\}$ that will become the relational database schema D is called a decomposition of R.

→ Each attribute of R must appear in atleast one relation.

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

This is called the attribute preservation condition of a decomposition.

- Each Relation D be in BCNF (or 3NF)

2) Decomposition & Dependency Preservation

It would be useful if each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relation schema R_i in the decomposition D. or could be inferred from the dependencies that appear in some R_i. this is

Dependency preservation condition.

Given a set of dependencies F on R, the projection of F on R_i, denoted by $\pi_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that $X \cup Y$ are all contained in R_i.

Hence, the projection of F on R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left & right-hand-side attributes are in R_i

we say that a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is

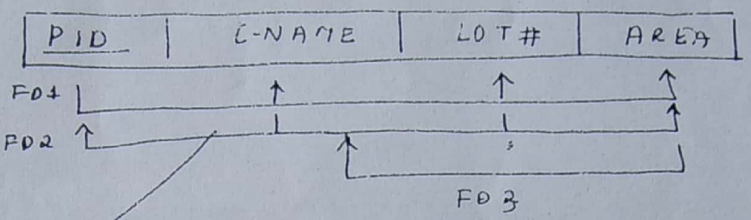
Dependency preserving w.r.t F if the union of the projection of F on each R_i in D is equivalent to F ie

$$\left((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)) \right)^+ = F^+$$

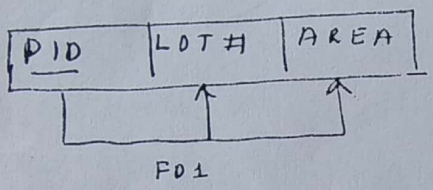
If a decomposition is not dependency - preserving, some dependency is lost

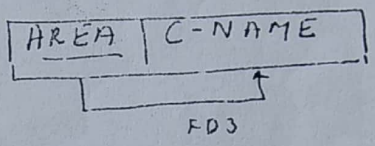
e.g. Decomposition that does not preserve dependencies

LOTS1A



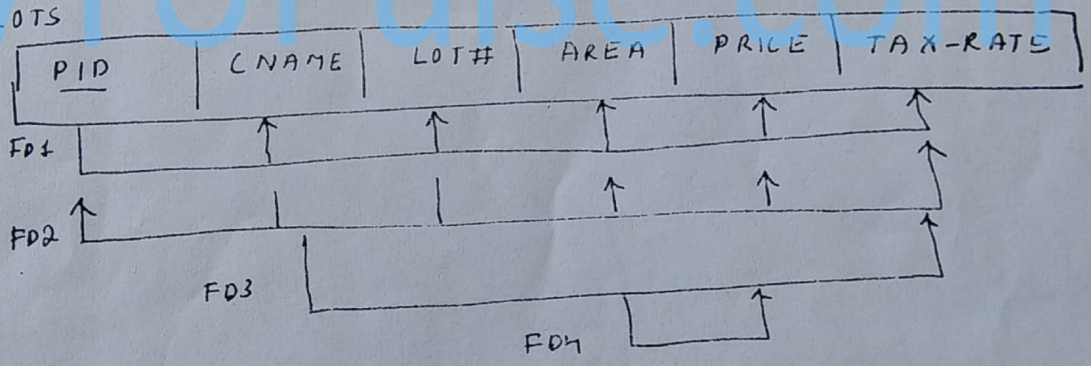
FD2 is lost in the below decomposition



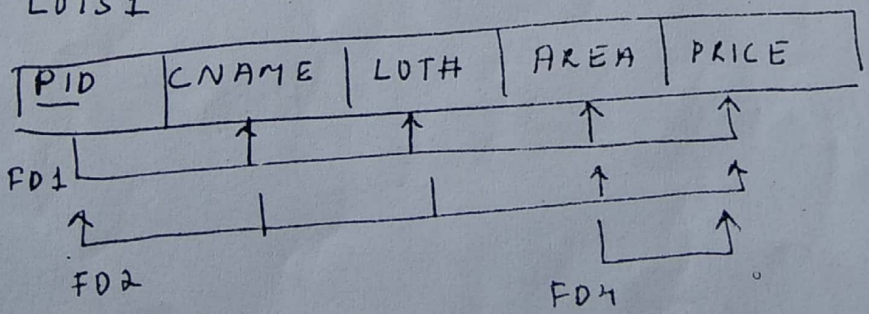


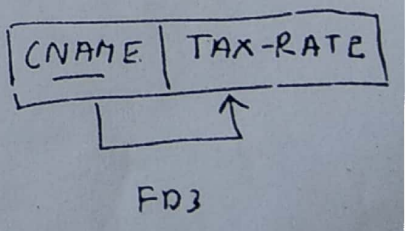
Decomposition that preserve dependencies

LOTS



LOTS1





3) Decomposition & lossless (Nonadditive) joins

Another property a decomposition D should possess is the lossless join or nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations in the decomposition.

Lossless join property is always defined with respect to a specific set F of dependencies

Formally a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R has the lossless (nonadditive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all relation in D:

$$*(\pi_{R_1}(r), \ldots, \pi_{R_m}(r)) = r$$

The word loss in lossless refers to loss of information not to loss of tuples.

If a decomposition does not have the lossless join property, we may get spurious tuples after PROJECT (π) & NATURAL JOIN ($*$) operations are applied.

Algorithm to check whether a given decomposition D has the lossless join property with respect to a set of FD F.

Algorithm Testing for the lossless (nonadditive) join property.

Input: A universal relation R, a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R, & a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation R_i in D, & one column j for each attribute A_j in R.

2. Set $S(i,j) = b_{ij}$ for all matrix entries.
(* each b_{ij} 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 (relation R_i includes attribute A_j) then
set (i, j) = a_j }; };
(* each a_j 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 $X \rightarrow Y$ 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 corresponds 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 row to the 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 same "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.

Consider the universal relation

$R = \{A, B, C, D, E, F, G, H, I, J\}$ & a set of functional dependencies $F = \{ AB \rightarrow C,\ A \rightarrow DE,\ B \rightarrow F,\ F \rightarrow G,H,\ D \rightarrow IJ \}$

Determine the decomposition

$D = \{ R_1, R_2, R_3, R_4, R_5 \}$;

$R_1 = \{A, B, C\}$ $R_2 = \{A, D, E\}$ $R_3 = \{B, F\}$ $R_4 = \{F, G, H\}$

$R_5 = \{D, I, J\}$ is lossless join decomposition.

	A	B	C	D	E	F	G	H	I	J
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}	b_{17}	b_{18}	b_{19}	$b_{1,10}$
R_2	a_1	b_{22}	b_{23}	a_4	a_5	b_{26}	b_{27}	b_{28}	b_{29}	$b_{2,10}$
R_3	b_{31}	a_2	b_{33}	b_{34}	b_{35}	a_6	b_{37}	b_{38}	b_{39}	$b_{3,10}$
R_4	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	a_6	a_7	a_8	b_{49}	$b_{4,10}$
R_5	b_{51}	b_{52}	b_{53}	a_4	b_{55}	b_{56}	b_{57}	b_{58}	a_9	a_{10}

$AB \rightarrow C$

No change

$A \rightarrow DE$.

R_1 $a_1 \rightarrow a_4$ a_5

R_2 $a_1 \rightarrow a_4$ a_5

	A	B	C	D	E	F	G	H	I	J
1	a_1	a_2	a_3	a_4	a_5	b_{16}	b_{17}	b_{18}	b_{19}	$b_{1,10}$
2	a_1	b_{22}	b_{23}	a_4	a_5	b_{26}	b_{27}	b_{28}	b_{29}	$b_{2,10}$
3	b_{31}	a_2	b_{33}	b_{34}	b_{35}	a_6	b_{37}	b_{38}	b_{39}	$b_{3,10}$
4	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	a_6	a_7	a_8	b_{49}	$b_{4,10}$
5	b_{51}	b_{52}	b_{53}	a_4	b_{55}	b_{56}	b_{57}	b_{58}	a_9	a_{10}

B → F

	B		F
R_1	a_2	→	a_6
R_3	a_2	→	a_6

} changes

	A	B	C	D	E	F	G	H	I	J
R_1	a_1	a_2	a_3	a_4	a_5	a_6	b_{17}	b_{18}	b_{19}	$b_{1,10}$
R_2	a_1	b_{22}	b_{23}	a_4	a_5	b_{26}	b_{27}	b_{28}	b_{29}	$b_{2,10}$
R_3	b_{31}	a_2	b_{33}	b_{34}	b_{35}	a_6	b_{37}	b_{38}	b_{39}	$b_{3,10}$
R_4	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	a_6	a_7	a_8	b_{49}	$b_{4,10}$
R_5	b_{51}	b_{52}	b_{53}	a_4	b_{55}	b_{56}	b_{57}	b_{58}	a_9	a_{10}

F → G H

	F	G	H
R_1	a_6	→ a_7	a_8
R_3	a_6	a_7	a_8
R_4	a_6	a_7	a_8

} changes

	A	B	C	D	E	F	G	H	I	J
R_1	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	b_{19}	$b_{1,10}$
R_2	a_1	b_{22}	b_{23}	a_4	a_5	b_{26}	b_{27}	b_{28}	b_{29}	$b_{2,10}$
R_3	b_{31}	a_2	b_{33}	b_{34}	b_{35}	a_6	a_7	a_8	b_{39}	$b_{3,10}$
R_4	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	a_6	a_7	a_8	b_{49}	$b_{4,10}$
R_5	b_{51}	b_{52}	b_{53}	a_4	b_{55}	b_{56}	b_{57}	b_{58}	a_9	a_{10}

D → I, J

	D	I	J
R_1	a_4	→ a_9	a_{10}
R_2	a_4	a_9	a_{10}
R_3	a_4	a_9	a_{10}

} changes.

	A	B	C	D	E	F	G	H	I	J
R_1	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}

←

Hence lossless join decomposition.

consider $R = (A\ B\ C\ D\ E)$ which is decomposed into

$R_1 = (A, B, C)$ $R_2 = (C, D, E)$ with

FD $A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$, $E \rightarrow A$

Show that above decomposition of the schema R is not a lossless

	A	B	C	D	E
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	b_{21}	b_{22}	a_3	a_4	a_5

$A \rightarrow BC$

No change.

$CD \rightarrow E$

No change.

$E \rightarrow A$

No change.

The above table does not contain any of the row completely with symbol a so it is not a lossless decomposition.

Consider $R = (A_1, A_2, A_3, A_4, A_5)$ which is decomposed into $R_1(A_1, A_2, A_3, A_5)$ $R_2(A_1, A_3, A_4)$ $R_3(A_4, A_5)$

FD $A1 \rightarrow A3\ A5$ $A5 \rightarrow A1A4$ $A3\ A4 \rightarrow A2$

	A_1	A_2	A_3	A_4	A_5
R_1	a_1	a_2	a_3	b_{14}	a_5
R_2	a_1	b_{22}	a_3	a_4	b_{25}
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5

$A_1 \rightarrow A_3 \quad A_5$

	A_1	A_3	A_5
R_1	a_1	a_3	a_5
R_2	a_1	a_3	a_5

	A_1	A_2	A_3	A_4	A_5
R_1	a_1	a_2	a_3	b_{14}	a_5
R_2	a_1	b_{22}	a_3	a_4	a_5
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5

$A_5 \rightarrow A_1 \quad A_4$

	A_5	A_1	A_4
R_1	a_5	a_1	a_4
R_2	a_5	a_1	a_4
R_3	a_5	a_1	a_4

	A_1	A_2	A_3	A_4	A_5
R_1	a_1	a_2	a_3	a_4	a_5
R_2	a_1	b_{22}	a_3	a_4	a_5
R_3	a_1	b_{32}	b_{33}	a_4	a_5

Decomposition is lossless join

problems with Null values and Dangling tuples

problem occurs when some tuples have null values for attributes that will be used to JOIN individual relations in the decomposition.

EMP	ENO	ENAME	FK DNO
	123	Adarsh	1
	124	Sharath	2
	125	Chetan	NULL

DEPT	DNO	DNAME
	1	CS
	2	IS

Chetan Employee is not been assigned to DNO.

To retrieve a list of (ENAME, DNAME) then we will apply NATURAL JOIN

EMP * DEPT

ENO	ENAME	DNO	DNAME
123	Adarsh	1	CS
124	Bharath	2	IS

SO '125 Chetan' is lost.

This problem can be over come by using OUTER JOIN

EMP ⟕ DEPT

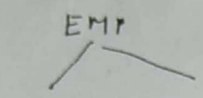
EMP.DNO = DEPT.DNO

ENO	ENAME	DNO	DNAME
123	Adarsh	1	CS
124	Bharath	2	IS
125	Chetan	NULL	NULL

SO NULL values can cause potential loss of information

A related problem is that of dangling Tuples, which may occur if we carry a decumposition too far.

Suppose that we decompose the EMP relation further.



EMP1	ENO	ENAME
	123	Adarsh
	124	Bharath
	125	Chetan

EMP2	ENO	DNO
	123	1
	124	2
	125	NULL

If we apply NATURAL JOIN on EMP1 * EMP2 we get orginal EMP relation.

Instead of EMP2 we use

EM3	ENO	DNO
	123	1
	124	2

Then EMP1 * EM3 will loose the information 123 Chetan NULL is called Dangling tuple

Multivalued Dependencies and Fourth Normal form

Multivalued dependency (MVD) are a consequence of first normal form (1NF), which disallowed an attribute in a tuple to have a set of values.

If we have two or more multivalued independent attribute in the same relation schema, we get into a problem of having to repeat

every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

for eg

EMP

ENAME	PNAME	DNAME
SMITH	X	John
SMITH	Y	Anna
SMITH	Y	John
SMITH	X	ANNA

Employee works on the project & has dependent. An EMP may work in several projects & may have several dependents & the EMP, project & dependent are independent of one another

informally, whenever two independent 1:N relationships A:B & A:C are mixed in same relation an MVD may arise.

Formal Defenition of Multivalued Dependency

Formally a multivalued dependency (MVD) $X \twoheadrightarrow Y$ specified on relation schema R, where X & Y are both subset of R; specifies the following constraint on any relation. state r of R:

If two tuples t_1 & t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 & t_4 should also exist in r with the following properties, where we use Z to denote $(R-(X \cup Y))$

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

$$t_3[Y] = t_1[Y] \quad \& \quad t_4[Y] = t_2[Y]$$

$$t_3[Z] = t_2[Z] \quad \& \quad t_4[Z] = t_1[Z]$$

whenever $X \twoheadrightarrow Y$ holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R, so does $X \twoheadrightarrow Z$. Hence $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$ & $X \twoheadrightarrow Z$. Therefore it is sometimes written as $X \twoheadrightarrow Y|Z$.

In the above example

ENAME $\twoheadrightarrow$ PNAME & ENAME $\twoheadrightarrow$ DNAME

hold in the EMP relation.

An MVD $X \twoheadrightarrow Y$ in R is called a trivial MVD if a) Y is a subset of X or b) $X \cup Y = R$

Ex EMP_PROJECTS

ENAME	PNAME
SMITH	X
SMITH	Y

ENAME $\cup$ PNAME $\longrightarrow$ Relation so it is trivial MVD.

An MVD that satisfies neither a) nor b) is called a nontrivial MVD.

If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples.

Inference Rules for Functional & Multivalued Dependencies

Universal relation schema $R = \{A_1, A_2, \ldots, A_n\}$ & that X, Y, Z & W are subsets of R

IR1 (reflexive rule of FDs): If $X \supseteq Y$, then $X \rightarrow Y$

IR2 (augmentation rule for FDs): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

IR3 (transitive rule for FDs): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

IR4 (complementation rule for MVDs): $\{X \twoheadrightarrow Y\} \models \{X \twoheadrightarrow (R - (X \cup Y))\}$

IR5 (augmentation rule for MVDs): If $X \twoheadrightarrow Y$ & $W \supseteq Z$ then $WX \twoheadrightarrow YZ$

IR6 (transitive rule for MVDs): $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$

IR7 (replication rule for FD TO MVD): $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$

IR8 (coalescence rule for FDs & MVDs): If $X \twoheadrightarrow Y$ & there exists W with the properties that (a) $W \cap Y$ is empty, (b) $W \rightarrow Z$ & (c) $Y \supseteq Z$, then $X \rightarrow Z$.

Fourth Normal Form

A relation schema R is in 4NF with respect to a set of dependencies F if, for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ in F^+, X is a superkey for R.

Ex

EMP

ENAME	PNAME	DNAME

is not in 4NF bec in the nontrivial MVDs

ENAME $\twoheadrightarrow$ PNAME & ENAME $\twoheadrightarrow$ DNAME

ENAME is not a superkey of EMP.

We decompose EMP into

EMP_PROJ & EMP_DEPENDENT

ENAME $\twoheadrightarrow$ PNAME

~~EMP $\twoheadrightarrow$ DNAME~~

ENAME $\twoheadrightarrow$ DNAME

ENAME	PNAME
SMITH	X
SMITH	Y

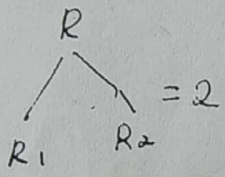
ENAME	DNAME
SMITH	JOHN
SMITH	ANNA

Decomposition saves storage, but also anomalies associated with updation are avoided.

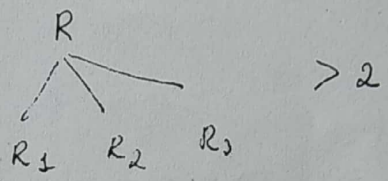
Join Dependencies & Fifth Normal Form

generally a Relation schema R when divided in to R_1 & R_2 has the loss less join property if we can get back the orginal relation R.

In some cases there may be no lossless join decompooition of R in to two relation schema but there may be a lossless join decomposition in to more than two relation schemas.



No lossless join



lossless join

This dependency is called join dependency & if it is present, carry out a multiway decomposition into fifth normal form (5NF).

NOTE such a dependency is very difficult to detect in practice & therefore Normalization upto 5NF is considered very rarely in practice.

A relation schema R is in fifth normal form (5NF) or project-join normal form (PJNF) with respect to a set F of functional, multivalued & join dependencies if, for every nontrivial join

dependency JD ($R_1, R_2, \ldots, R_n$) in F^+, every R_i is a superkey of R.

Inclution Dependencies

Inclution dependencies were defined in order to formalize certain interrelational constraints.

For example, the foreign key (or referential integrity) constraint cannot be specified as a functional or multivalued dependency because it relates attributes across relations. but it can be specified as inclution dependency

Inclution dependency can also be used to represent the constraint between two relation that represent a class / subclass

Formally an inclution dependency R.X < S.Y between two sets of attributes X of relation schema R, & Y of relation schema S, specifies the constraint that, at any specific time when r is a relation state of R & s a relation state of S, we must have

$$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

set of attributes on which the inclution dependency is specified X of R & Y of S

must have same attributes. In addition domain for each pair of corresponding attributes should be compatible.

Ex

EMP (ENO, NAME, DNO) DEP(DNO, NAME)

inclation dependency can be specified as

EMP. DNO < DEP. DNO

class / subclass relationship

PERSON

SSN	NAME	DOB	Address

EMP

SSN	sal	position	Rank	Project	courxe

EMP. SSN < PERSON. SSN

** Template Dependencies

Constraint that is based on the semantics of attributes with in a relations. The idea behind template dependencies is to specify a template - or example that defines each constraint or dependency There are two types of tuples that may appear in one or more relation.

Tuple generation template &

constraint - generating templates.

A template consists of a number of hypothetical tuples that are meant to show an example of the tuples that may appear in one or more relations.

(a) $R = \{A, B, C, D\}$

hypothesis
$a_1 \quad b_1 \quad c_1 \quad d_1$ $\quad X = \{A, B\}$
$a_1 \quad b_1 \quad c_2 \quad d_2$ $\quad Y = \{C, D\}$

conclusion $c_1 = c_2$ & $d_1 = d_2$

Template for functional dependency $X \rightarrow Y$

(b)
hypothesis
$a_1 \quad b_1 \quad c_1 \quad d_1$ $\quad X = \{A, B\}$
$a_1 \quad b_1 \quad c_2 \quad d_2$ $\quad Y = \{C\}$

conclusion
$a_1 \quad b_1 \quad c_1 \quad d_2$
$a_1 \quad b_1 \quad c_2 \quad d_1$

Template for Multivalued dependency.

EMPLOYEE = {NAME, SSN, ..., SAL, SUPERVISORSSN}

hypothesis
$a \quad b \quad c \quad d$
$e \quad d \quad f \quad g$

conclusion $c < f$

Template for the constraint that an employee salary must be less than the supervisor salary.

Domain Key Normal Form (DKNF)

The idea behind domain-key normal form is to specify the "ultimate normal form" that takes in to account all possible types of dependencies & constraints.

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

BEST OF LUCK