Database and Information Systems

Course Roadmap

Chapter 1	Introduction to Databases
Chapter 2	Integrity Constraints and ER Model
Chapter 3	Relational Databases and Schema Refinement
Chapter 4	Query Language
Chapter 5	Transaction and Concurrency Control
Chapter 6	Indexing

Features of Good Relational Designs

- Suppose we have *in_dep table with instructor and department information* where ID is the Primary Key
  - A database designer created a database where the records are stored in the following way

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: in_dep

Features of Good Relational Designs

- Suppose we have *in_dep table with instructor and department information* where ID is the Primary Key

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: in_dep

- Class Activity
  - What are the problems in this table?

Features of Good Relational Designs

- Suppose we have *in_dep table with instructor and department information* where ID is the Primary Key

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: in_dep

- Class Activity
  - What are the problems in this table?
  - Two rows can be same or not?

Features of Good Relational Designs

- Suppose we have *in_dep table with instructor and department information*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: in_dep

- There is repetition of information
  - Redundancy

Features of Good Relational Designs

- Suppose we have *in_dep table with instructor and department information*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: in_dep

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

Anomaly

- Insertion Anomaly
  - If we want to add new department information
    - We cannot insert because we need primary key ID
- Deletion Anomaly
  - Deletion of some data would force to delete other data
  - If we remove only faculty in department
    - We will lose department information
    - For Example, if we remove faculty 98345, Kim in the previous table
- Updation Anomaly
  - All redundant copies has to be updated

Decomposition

- The only way to avoid the repetition-of-information problem in the i*n_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary)

  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

  The problem arises when we have two employees with the same name
- Decomposition should be **lossless** and **dependency preserving**

Decomposition

- Which Decomposition(s) is/are Good? And Why? (+1 **Reward Question**)
  - **D1**
    - R1(ID, name, salary) and R2(dept_name, building, budget)
  - **D2**
    - R1(ID, name, salary) and R2(dept_name, building, budget) and R3(ID, dept_name)
  - **D3**
    - R1(ID, name, salary, dept_name) and R2(dept_name, building, budget)
  - **D4**
    - R1(ID, name, salary,) and R2(dept_name, building, budget, ID)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Table: R

Normalization

- Normalization is a method to reduce or remove redundancy from a table
  - Redundancy occurs in a table or relation if two or more independent relations stored in a single table
- Normalization reduces or removes insertion, deletion, and updation anomaly
  - Decomposition helps to achieve (maintain) normalization

Normal Forms

- **Redundancy** in a relation may cause insertion, deletion and updation anomalies
- **Normal forms** are used to eliminate or reduce redundancy in **database** tables
- **Normalization** is the process of minimizing **redundancy** from a relation or set of relations

First Normal Form

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
  - **Relation should not have any multi valued attribute**
- Domain is **atomic** if its elements are considered to be indivisible units
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data

Roll No	Name	Course
1	Archit	C/C++
2	Harsh	DBMS
3	Ankush	OS/C

Conversion into First Normal Form

- First method

Roll No	Name	Course
1	Archit	C
1	Archit	C++
2	Harsh	DBMS
3	Ankush	OS
3	Ankush	C

- What can be a Primary Key in the above Table?

Conversion into First Normal Form

- Second Method

Roll No	Name	Course 1	Course 2
1	Archit	C	C++
2	Harsh	DBMS	NULL
3	Ankush	OS	C

- What can be a Primary Key in the above Table?
- Problem: NULL values

Conversion into First Normal Form

- Third Method

- Referenced Table

Roll No	**Name**
1	Archit
2	Harsh
3	Ankush

- Referencing Table

Roll No	**Course**
1	C
1	C++
2	DBMS
3	OS
3	C

Functional Dependency

- Require that the value for a certain set of **attributes determines uniquely the value** for another set of attributes
  - **Describe dependency or relationship between attributes**
  - For Example, **x** $\rightarrow$ ***y***
    - Here, x determines y or y is determined by x
      - x is a **determinant** attribute and y is **dependent** attribute
- Example: Sid $\rightarrow$ Sname

1	Ram
2	Ram

Different students but name same, determined by Sid

1	Ram
1	Ram

Same student determined by Sid, but record is replicated

- A functional dependency is a generalization of the notion of a *key*

Functional Dependencies Definition

- Let *R* be a relation schema

$$\alpha \subseteq R \quad and \quad \beta \subseteq R$$

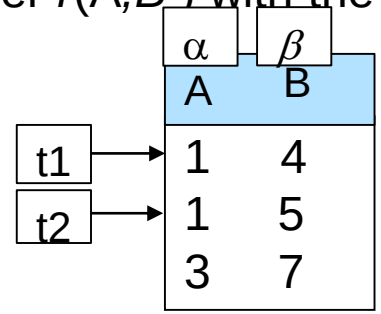
- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on *R* if and only if for any legal relations *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

- Example: Consider *r*(A,*B*) with the following instance of *r.*



- On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold

Functional Dependencies Example

- Example: $s_{id} \rightarrow s_{name}$

S_{id}	S_{name}
1	Ram
2	Shyam

valid

S_{id}	S_{name}
1	Ram
1	Ram

valid

S_{id}	S_{name}
1	Ram
2	Ram

valid

S_{id}	S_{name}
1	Ram
1	Shyam

invalid

- Check if two students same or not

Types and Properties of Functional Dependency

- Types of Functional Dependency
  - **Trivial Functional Dependency**
    - If x $\subseteq$ y then y $\rightarrow$ x is a trivial functional dependency (FD), where x and y are some attributes over relation R
      - **Always valid,** (RHS of FD is a subset of LHS of FD)
      - Example, Sid $\rightarrow$ Sid, SidSname $\rightarrow$ Sid
  - **Non-trivial Functional Dependency**
    - If x $\cap$ $y = \phi$ then y $\rightarrow$ x is a non trivial functional dependency
      - **Need to check validity**
      - Example, Sid $\rightarrow$ Sname

Properties of Functional Dependency

- **Reflexive rule:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (trivial and valid)
- **Augmentation rule:** if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ (valid)
- **Transitivity rule:** if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (valid)

Properties of Functional Dependency

- **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (valid)
- **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (valid)
- **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (valid)
- **Composition rule:** If $\alpha \rightarrow \beta$ holds and $\delta \rightarrow \gamma$ holds, then $\alpha\delta \rightarrow \beta\gamma$ holds (valid)
- Question: If $\beta\gamma \rightarrow \alpha$ holds, then $\beta \rightarrow \alpha$ holds and $\gamma \rightarrow \alpha$ holds (valid or invalid?)

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - etc.
- The **set of all functional dependencies logically implied by *F*** is the **closure** of *F*.
- We denote the *closure* of *F* by F^+.
- It **helps to find all candidate keys in a relation**

Closure of Functional Dependency

- Example
  - R(ABCD), FD{A -> B, B->C, C->D}
    - A^+ = ABCD , **A^+ indicates what A can determine**
    - B^+ = BCD
    - C^+ = CD
    - D^+ = D
    - Candidate key (CK) = {A}
      - As attribute A is determining all the attributes of the relation
    - Prime Attribute (PA) = {A}, Non Prime Attribute (NPA) = {BCD}
      - Prime attributes are used in making the Candidate keys (CK)
      - **PA= All the attributes that are present in the set of CK**
      - **NPA= {All the attributes of Relation} - PA**
- Prime attribute are used to form CK
- What are other possible candidate keys in the above relation? Is it AB?

Closure of Functional Dependency

- R(ABCD), FD{A -> B, B->C, C->D, D->A}
  - CK = ?, Prime Attribute = ?, Non Prime Attribute = ?

Closure of Functional Dependency

- R(ABCDE), FD{A->B, BC->D, E->C, D->A} **(Reward Question)**
  - CK = ?, Prime Attribute = ?, Non Prime Attribute = ?

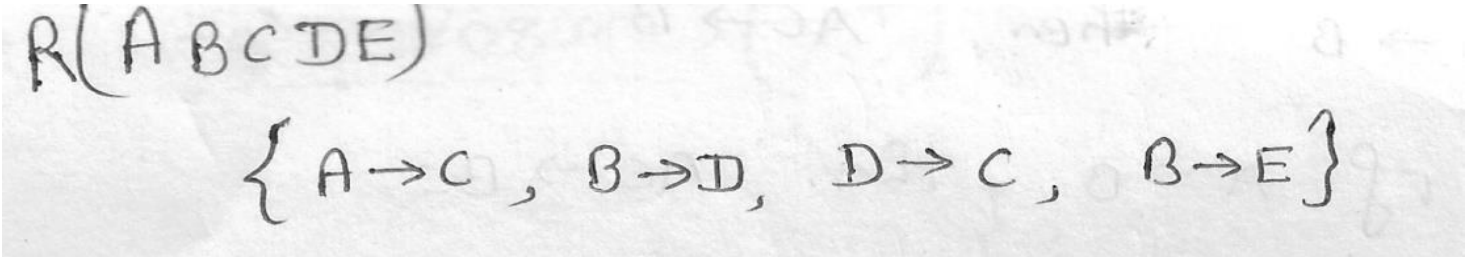
Closure of Functional Dependency

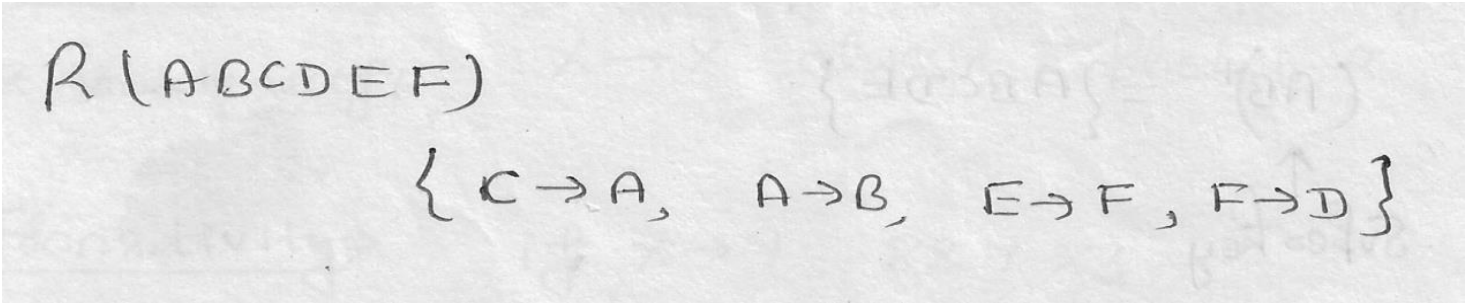
- R(ABCD), FD{A -> B, B->C, C->D, D->A}
  - A^+ = ABCD
  - B^+ = BCDA
  - C^+ = CDAB
  - D^+ = DABC
    - CK = {A,B,C,D}, Prime Attribute = {A,B,C,D}, Non Prime Attribute = {ϕ}

- R(ABCDE), FD{A->B, BC->D, E->C, D->A}
  - CK = {AE,DE,BE}, Prime Attribute = {A,B,D,E}, Non Prime Attribute = {C}
    - Note: E is not present in RHS of any of FD so it will always be a part of CK

Candidate Keys

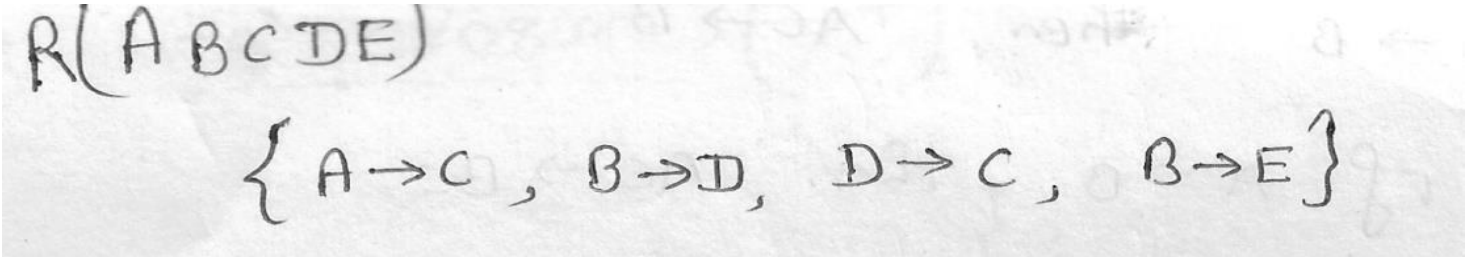
- Find Candidate Keys



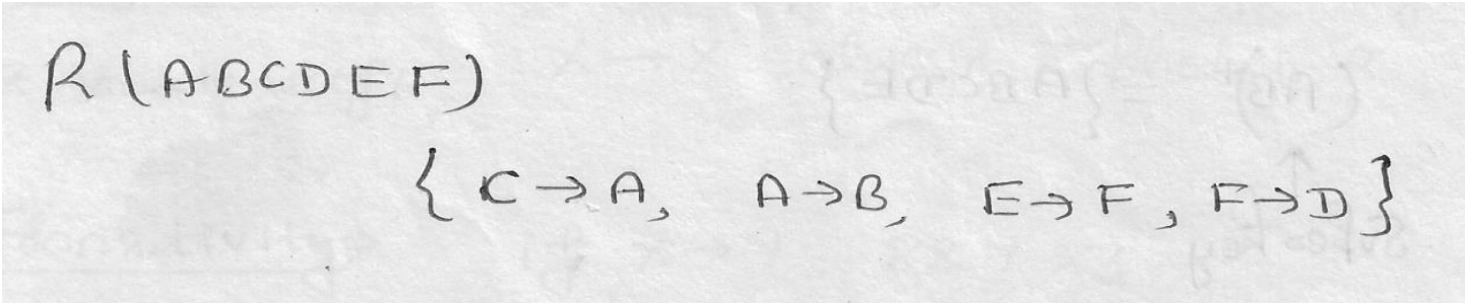


Candidate Keys

- Find Candidate Keys



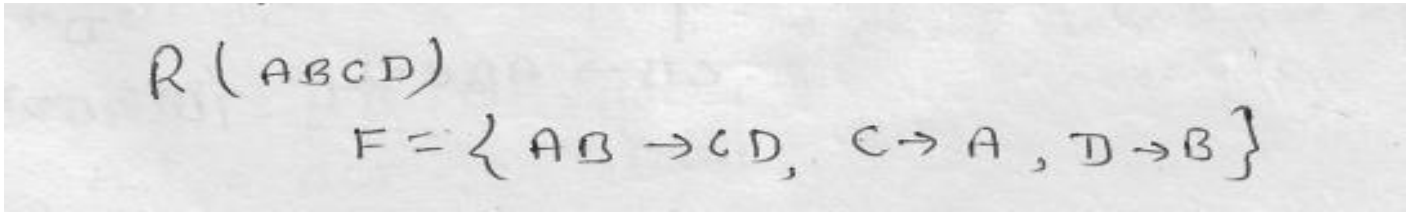
- CK = {AB}

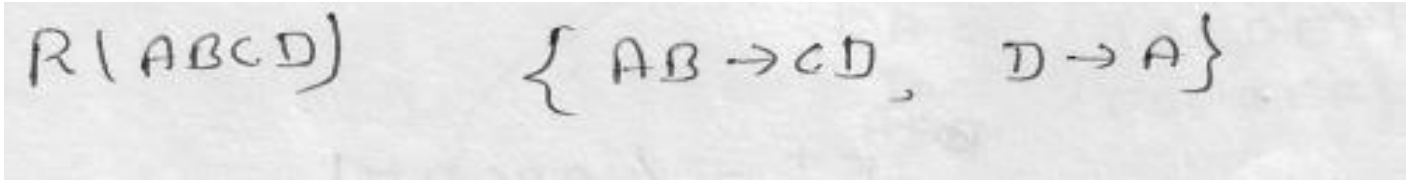


- CK = {CE}

Candidate Keys

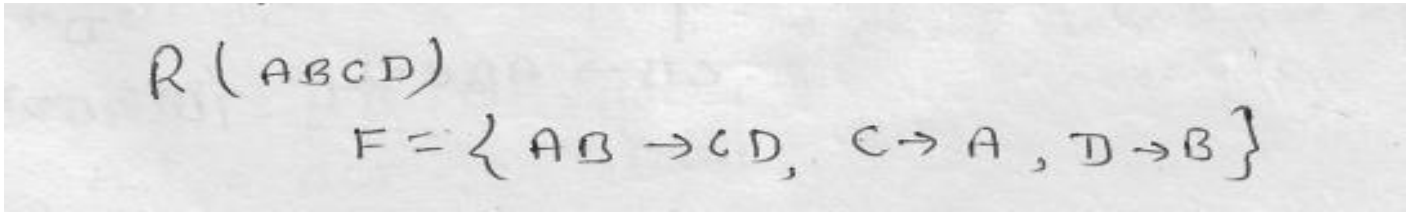
- Find Candidate Keys



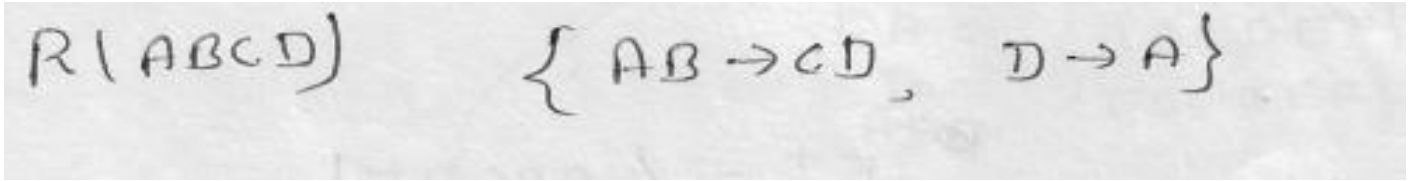


Candidate Keys

- Find Candidate Keys



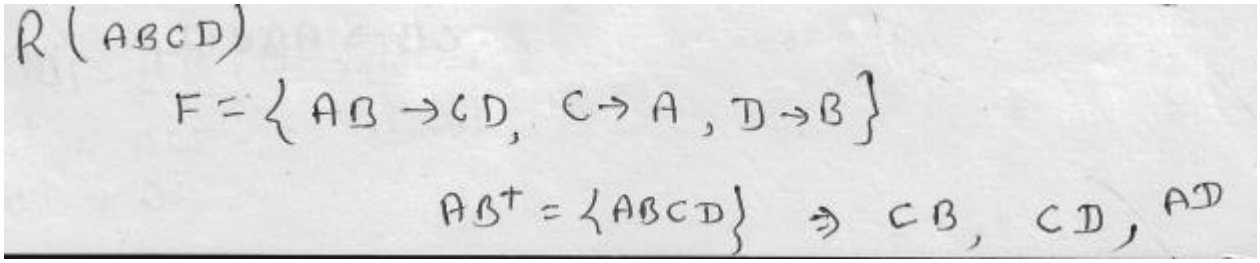
- CK = {AB, CD, CB, AD}

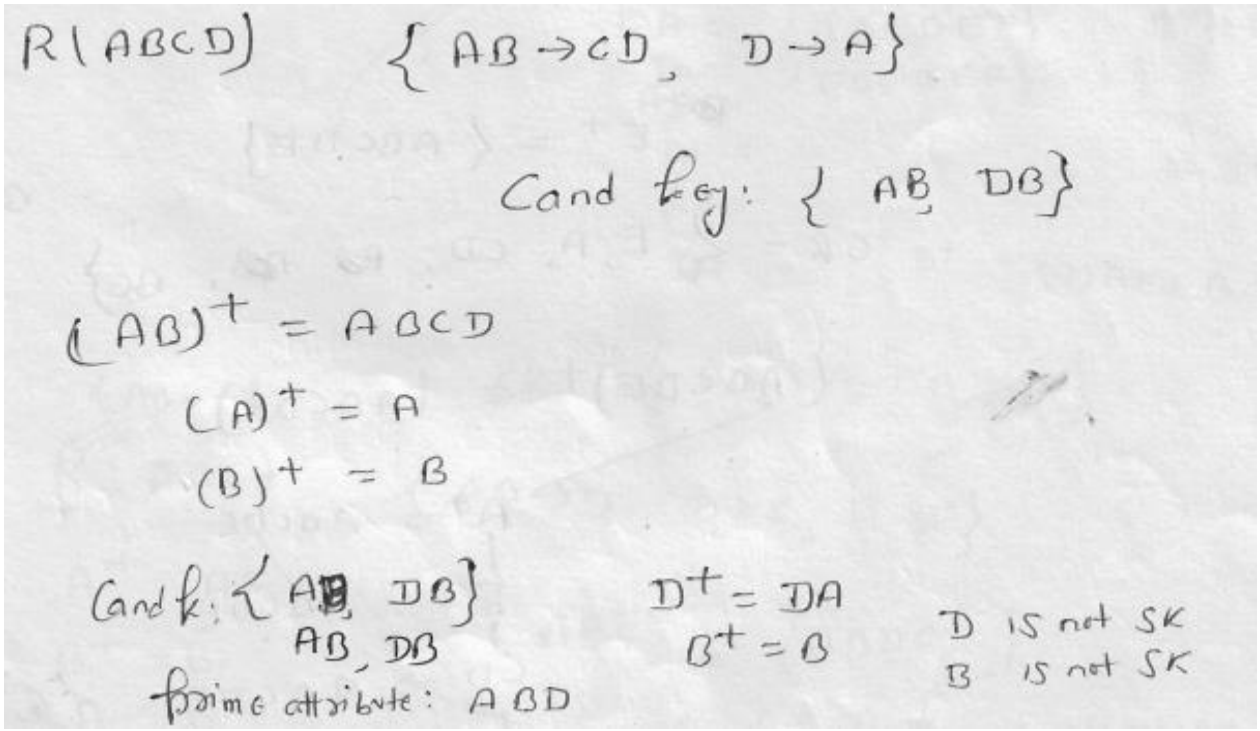


- CK = {AB, DB}

Candidate Keys

- Candidate keys Solutions





Candidate Keys

- Find the Candidate Keys
  - R(ABCDE), FD{A -> B, BC -> E, ED -> A}

Candidate Keys

- Find the Candidate Keys
  - R(ABCDE), FD{A -> B, BC -> E, ED -> A}
  - CK = {ACD, BCD, ECD}

Canonical Cover

- Suppose that we have a set of functional dependencies *F* on a relation schema. Whenever a user performs an update on the relation, the database system must ensure that the update **does not violate any functional dependencies**; that is, all the functional dependencies in *F* are satisfied in the new database state.

- If an update violates any functional dependencies in the set *F,* the system must roll back the update.

- We can **reduce the effort spent in checking for violations by testing a simplified set of functional dependencies that has the same closure as the given set.**

- This simplified set of functional dependency is termed the **canonical cover. It is irreducible set of functional dependency.**

Canonical Cover

A **canonical cover** for F is a set of dependencies F_c such that

- F logically implies all dependencies in F_c , and
- F_c logically implies all dependencies in F, and
- No functional dependency in F_c contains an extraneous attribute
  - An attribute of a functional dependency is said to be extraneous if we **can remove it without changing the closure of the set of functional dependencies**
- Each **left side of functional dependency in F_c is unique**. That is, there are no two dependencies in F_c
  - $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that
  - $\alpha_1 = \alpha_2$

Canonical Cover

- To compute a canonical cover for *F*:

  repeat

  Use the union rule to replace any dependencies in *F* of the form

  $$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

  Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β
  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until (F_c not change)

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Canonical Cover vs. Minimal Cover

- A canonical cover can have more than one attribute on the right hand side.
- A minimal cover cannot allow more than one attribute on the right hand side.
- Example: A -> BC can be canonical cover
  - Whereas the minimal cover for the same functional dependency would be A -> B, A -> C

Minimal Cover

- Method to find Minimal Cover
  - First, make RHS of each FD as a single attribute or decompose RHS
    - Use **decomposition rule**
  - Second, pick each functional dependency (FD) and check if other functional decencies (excluding current FD) can generate the same results/dependencies
    - If other FDs can generate, remove the current FD
      - Find the closure of LHS of each FD from other FDs (excluding current FD) and check if other dependencies can generate RHS of current FD. If so, remove current FD
    - This step is called **removal of redundant FD**

Minimal Cover

- Method to find Minimal Cover
  - Third, pick those FDs that have at least 2 attributes at LHS and try to reduce it (preferably make it one attribute on the LHS). E.g. xy->z
    - Check if closure of one attribute (x) at LHS can derive other attribute (y) of LHS. If so, remove other attribute (y)
      - Similarly we can check if x can be removed
    - This step is called **removal of extraneous attribute**

Minimal Cover

- Example: FD: {A -> B, C -> B, D -> ABC, AC -> D}
  - Step 1. {A -> B, C -> B, D -> A, D -> C, D -> B, AC -> D}
  - Step 2. {A -> B, C -> B, D -> A, D -> C, ~~D -> B~~, AC -> D}
  - Step 3. {A -> B, C -> B, D -> A, D -> C, AC -> D}
    - A^+ is not able to derive C and C^+ is not able to derive A. We therefore cannot remove any attribute at LHS of AC -> D
  - Final Minimal Cover: {A -> B, C -> B, D -> A, D -> C, AC -> D}
  - Final Canonical Cover: {A -> B, C -> B, D -> AC, AC -> D}

Second Normal Form

- Conditions for second normal form (2nd NF):
  - Relation must be in 1st normal form and
  - All non-prime attributes should be fully functional dependent on candidate key
    - Or Relation **should not contain any partial dependency**
      - Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key

Second Normal Form

Partial dependancy ⇒

X: Any cand key

Y: proper subset of ck

A: Non prime attribute

$Y \rightarrow A$: partial dependancy.

Second Normal Form

- In the following table, Store ID -> Location

Customer ID	**Store ID**	**Location**
1	1	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai

Table: CustomerStore

- In the above Table, PA ={Customer_ID, Store_ID}, NPA ={Location}
- Not in 2nd NF

Second Normal Form

- Conversion into second normal form (2nd NF)

Customer ID	**Store ID**
1	1
1	3
2	1
3	2
4	3

Table: Customer

Store ID	**Location**
1	Delhi
2	Bangalore
3	Mumbai

Table: Store

- Partial Dependency: **LHS is a proper subset of CK and RHS is a non prime attribute**
  - For 2nd NF, there should be no partial dependency

Second Normal Form

- Steps to check if a relation is in 2nd normal form
  - Step 1. Find all candidate keys from functional dependencies of a relation
  - Step 2. Find Prime and Non-Prime attributes
  - Step 3. Check partial dependency for all the dependencies in a relation
    - If any partial dependency exists then that relation is not in 2nd normal form

Second Normal Form

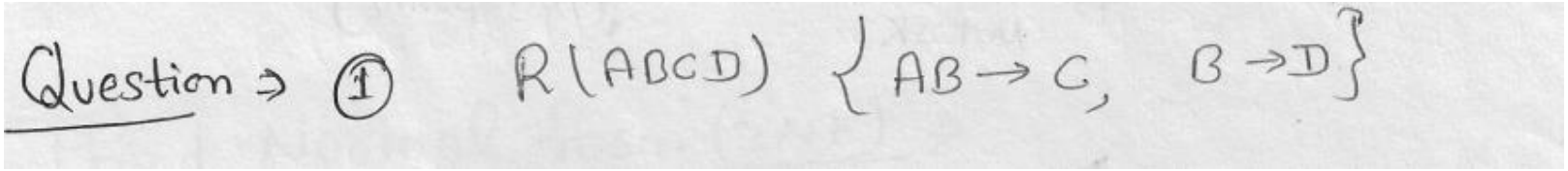
- Example: R(ABCDEF), FD{C->F, E->A, EC->D, A->B}
  - Check if the given relation is in 2nd NF?

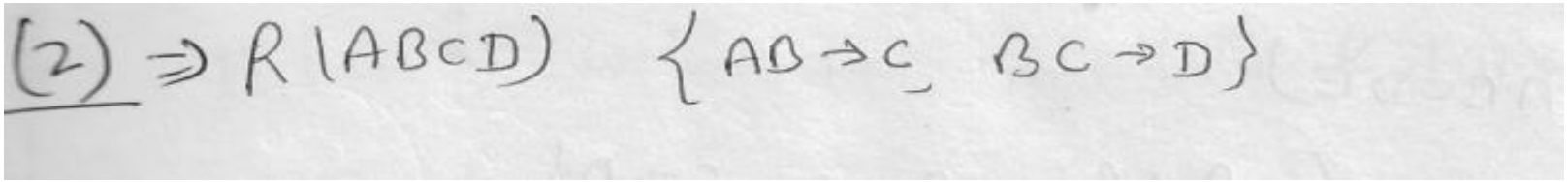
Second Normal Form

- Example: R(ABCDEF), FD{C->F, E->A, EC->D, A->B}
  - EC^{+}= ECDABF, CK={EC}
  - Prime Attribute = {E,C}, Non Prime Attribute= {A,B,D,F}
  - Partial Dependencies PD{C->F, E->A}
  - Two partial dependencies exist, so relation is not in 2nd NF

Second Normal Form

- Check if these relations are in 2nd NF?





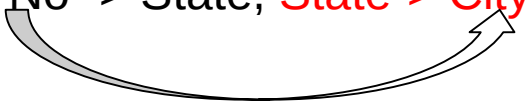
Third Normal Form

- Conditions for third normal form (3rd NF):
  - Relation should be in 2nd NF and
  - And there should be no transitive dependency
    - Non prime attribute should not be determined by non prime attribute

Roll No	**State**	**City**
1	Punjab	Mohali
2	TS	Hyderabad
3	MP	Indore
4	Punjab	Mohali
5	TS	Hyderabad
6	MP	Indore

- CK= {Roll No}, PA= {Roll No}, NPA = {State, City},

  FD: {Roll No -> State, State-> City}



Third Normal Form

- For 3rd NF, NPA -> NPA should not be there and it should be in 2nd NF
- Examples:
  - R(ABCD), FD: {AB -> C, C-> D}
    - Is it in 2nd NF?
    - Is it in 3rd NF?
  - R(ABCD), FD: {AB -> CD, D->A}
    - Is it in 2nd NF?
    - Is it in 3rd NF?

Third Normal Form

- For 3rd NF, NPA -> NPA should not be there and it should be in 2nd NF
- Examples:
  - R(ABCD), FD: {AB -> C, C-> D}, CK: {AB}, PA: {A,B}, NPA: {C, D}
    - Is it in 2nd NF?
    - Is it in 3rd NF?
  - R(ABCD), FD: {AB -> CD, D->A}, CK: {AB, DB}, PA:{A,B,D}, NPA: {C}
    - Is it in 2nd NF?
    - Is it in 3rd NF?

- Note: Simple condition for 3rd NF - LHS must be CK(or SK) or RHS must be PA
  - Using this condition, you can directly check if relation is in 3rd NF or not.

Boyce Codd Normal Form

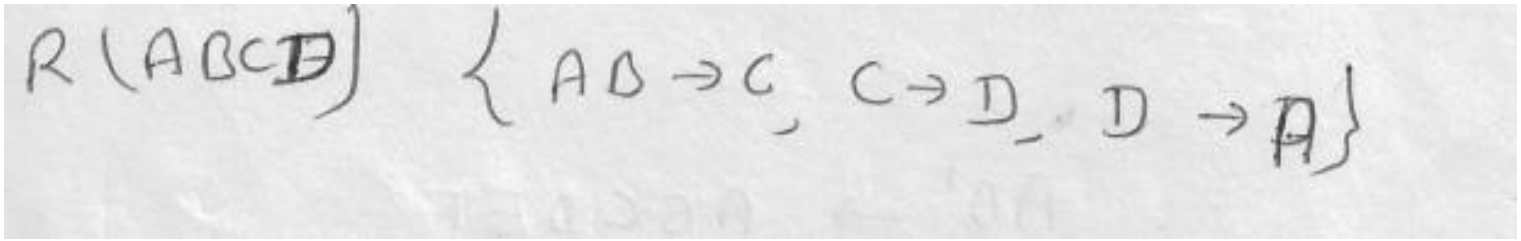
- Condition for Boyce Codd Normal Form (BCNF):
  - Relation should be in 3rd NF and
  - And all attributes should be functional dependent on CK
- Example: FD: {rollno -> name, rollno -> voterid, voterid -> age, voterid -> rollno}; CK: {rollno, voterid}

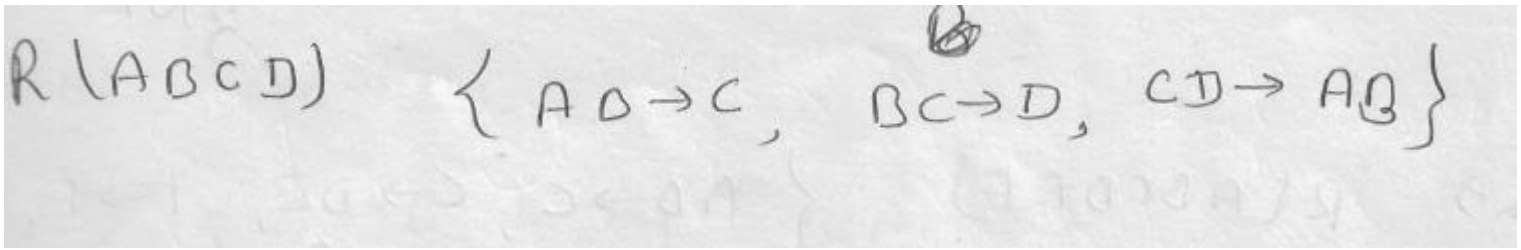
rollno	name	voterid	age
1	Ram	r0123	20
2	Shyam	v0546	21
3	Ram	r0678	22
4	Mohan	r0765	23

- Simple Condition: LHS of each FD should be CK or SK

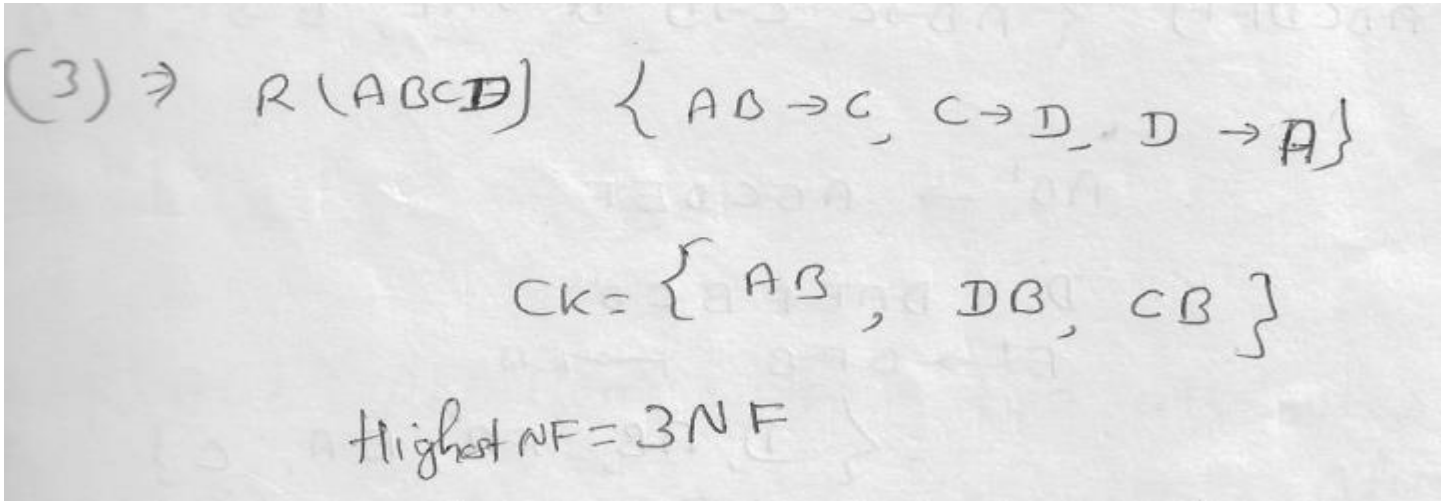
Boyce Codd Normal Form

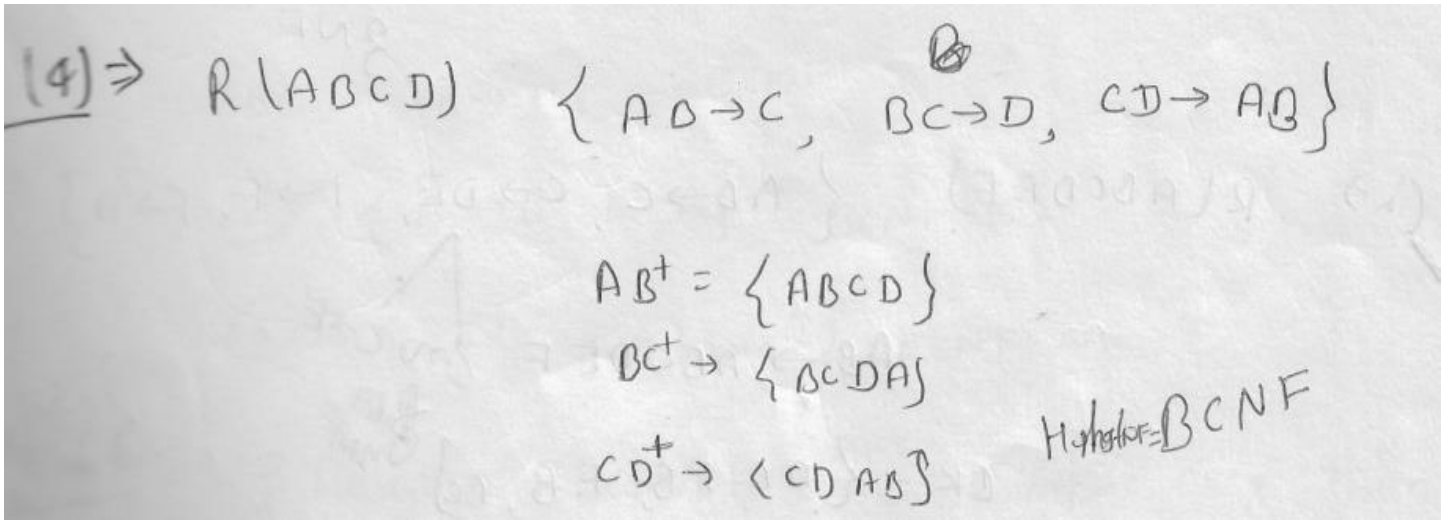
- Check if these relations are in BCNF?





Boyce Codd Normal Form

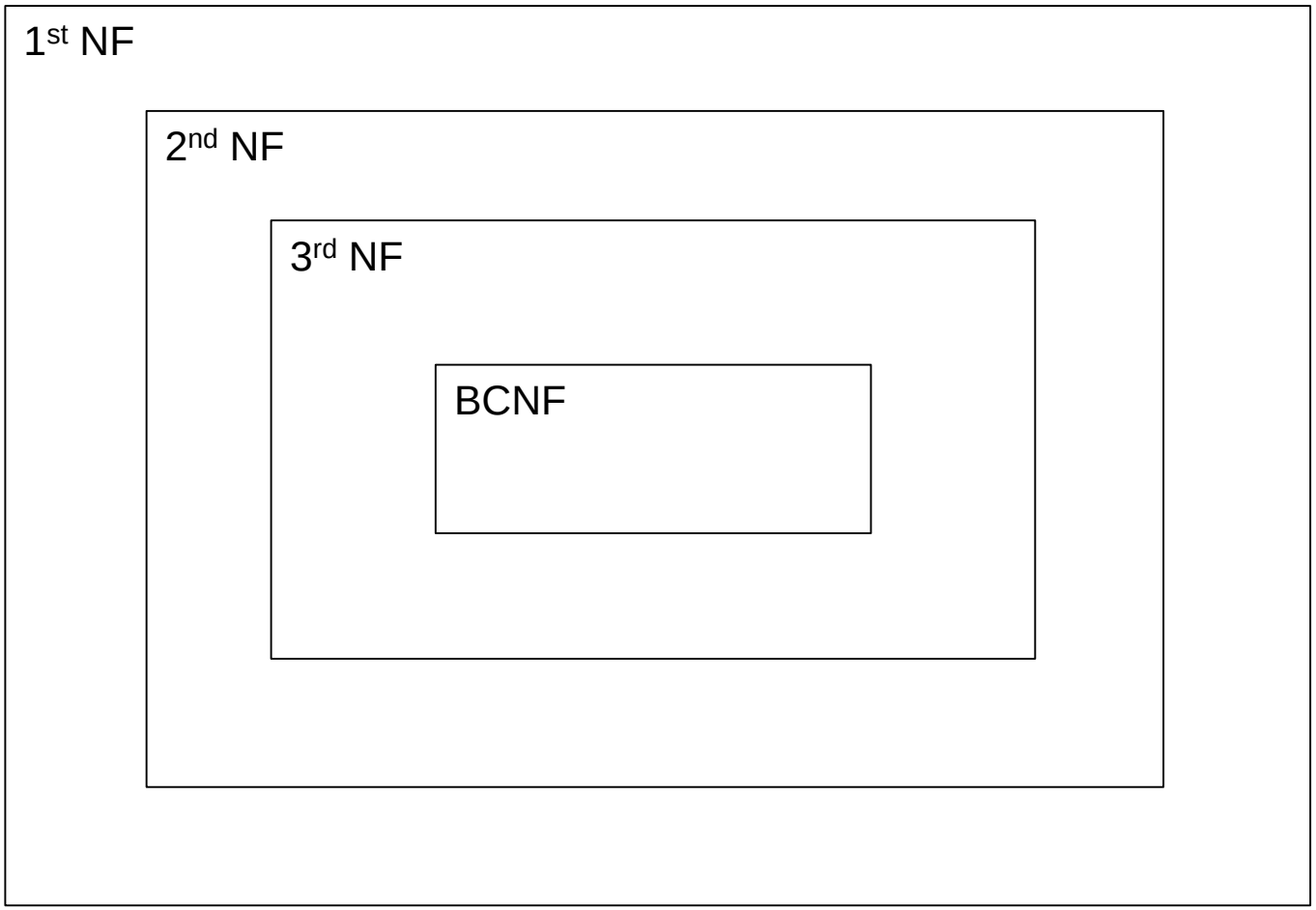




Highly Normalized Database

- Advantage of highly normalized database
  - Less redundancy
- Disadvantage of highly normalized database
  - It has more number of tables, which takes more query access time

Expressive Power



- 1st NF has highest expressive power and BCNF has highest restriction in the above diagram

Questions

- If a relation is in 3rd NF, it would be in 2nd NF?
  - Yes or No?
- If a relation is in 3rd NF, it would be in BCNF?
  - Yes or No?
- If a relation is in BCNF, it would be in 1st NF?
  - Yes or No?

Decomposition

- Decomposition removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables
  - When we convert a relation to higher normal form, we decompose the relation
- Properties of Decomposition
  - Lossless Decomposition
  - Dependency Preserving Decomposition

Lossless Decomposition

- Lossless Decomposition
  - A decomposition is a **lossless decomposition** if there is no loss of information by decomposing a relation (or replacing a relation with two relations)

A	B	C
1	2	1
2	2	2
3	3	2

R(ABC)

A	B
1	2
2	2
3	3

R1(AB)

B	C
2	1
2	2
3	2

R2(BC)

R is decomposed into R1 and R2

  - If this decomposition is lossless, we should get R after joining R1 and R2 (R1 Natural Join R2)
    - Check and tell if it is lossless?
  - Example Query: find the value C where the value of A = ‘1’

Decomposition

- Example Query
  - Select R2.C from R2 Natural Join R1 where R1.A = ‘1’ ;
  - Result of R1 Natural Join R2:

A	B	C
1	2	1
1	2	2
2	2	1
2	2	2
3	3	2

Spurious Tuples

  - Decomposition is lossy in terms of inconsistency

Decomposition

- Example Query
  - Select R2.C from R2 Natural Join R1 where R1.A = ‘1’ ;
  - Result of R1 Natural Join R2:

A	B	C
1	2	1
1	2	2
2	2	1
2	2	2
3	3	2

Spurious Tuples

  - Decomposition is lossy in terms of inconsistency
- Why this problem?
- And how we can remove this problem?
  - Reward

Why Lossy Decomposition

- Why lossy decomposition problem?
  - Because of common attribute, attribute B is not CK or SK of decomposed tables (either R1 or R2)

A	B	C
1	2	1
2	2	2
3	3	2

R(ABC)

A	B
1	2
2	2
3	3

R1(AB)

B	C
2	1
2	2
3	2

R2(BC)

Decomposition

- Question
  - In the previous example if we decompose R(ABC) into R1(AC) and R2(BC), it will be lossy or lossless?

Decomposition

- Real-life Example

ID	name	salary	dept_name	building	budget
1	Ram	120000	CS	POD1A	7000000
2	Shyam	130000	EE	POD1B	5000000
3	Mohan	140000	CS	POD1A	7000000
4	Gopal	150000	EE	POD1B	5000000
5	Keshav	160000	CS	POD1A	7000000
6	Anuj	170000	ME	POD1C	2000000
7	Bharat	180000	IT	POD1D	6000000

Table: R

Table R is decomposed into R1 and R2

ID	name	Salary	dept_name
1	Ram	120000	CS
2	Shyam	130000	EE
3	Mohan	140000	CS
4	Gopal	150000	EE
5	Keshav	160000	CS
6	Anuj	170000	ME
7	Bharat	180000	IT

Table: R1

dept_name	building	budget
CS	POD1A	7000000
EE	POD1B	5000000
ME	POD1C	2000000
IT	POD1D	6000000

Table: R2

Decomposition

- Criteria for common attribute while decomposing a table
  - Common attribute must be CK or SK of either R1 or R2 or both
    - In previous example, B and C are not CK because their values are repeating, duplicity
    - For lossless decomposition, we can make A as a common attribute: R1(AB) and R2(AC)
- Let *R* be a relation schema and let R_1 and R_2 form a decomposition of R
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas R_1 and R_2
- Formally,

$$\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

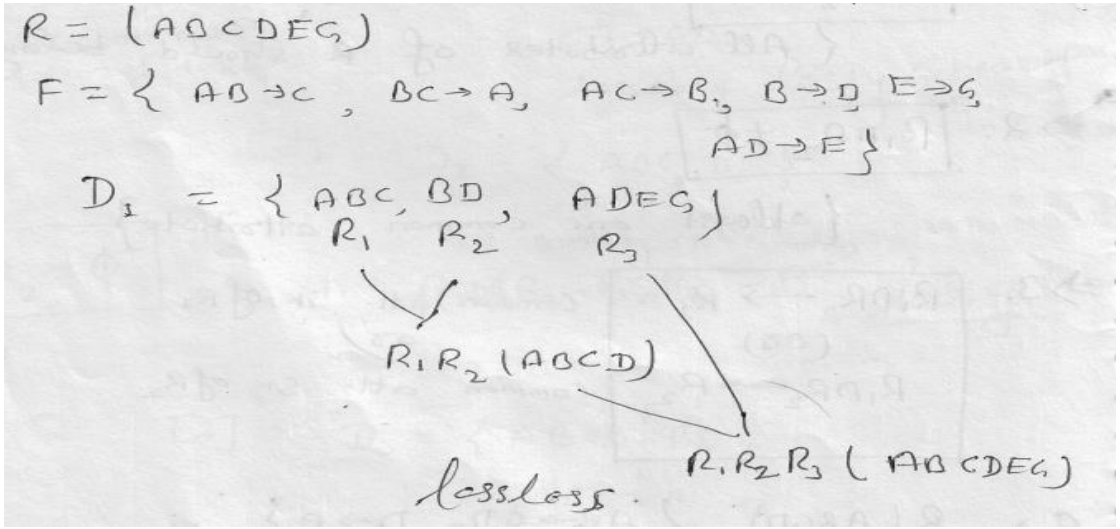
$$r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$$

Decomposition

- Simple conditions to check if decomposition is lossless
  - *Attribute*(R_1) U *Attribute*(R_2) = R
  - *Attribute*(R_1) ∩ *Attribute*(R_2) != ∅
  - Common attribute is super key (or candidate key) of either R1 or R2 or both

Decomposition

- Example
  - R(ABCDEG), F{AB->C, BC->A, AC->B, B->D, E->G, AD->E}
    - D_1 = {ABC, BD, ADEG} lossless



- Can we merge R1 and R3?
- Can we merge R2 and R3?

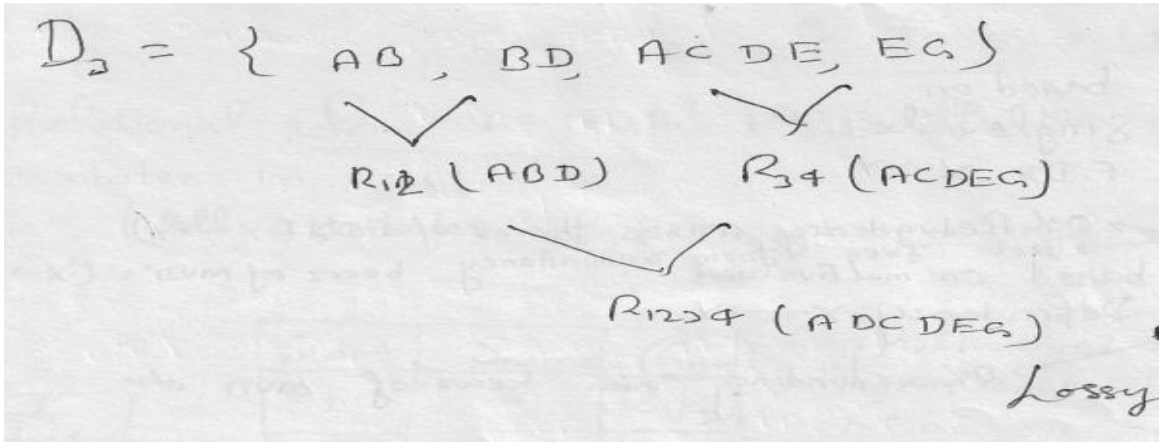
  - If you are able to merge all the decomposed relations without any problem and can get original relation, it is a lossless decomposition
    - While merging, please remember that common attribute should be a key of any of two relations

Decomposition

- Example of Decomposition: R(ABCDEG), F{AB->C, BC->A, AC->B, B->D, E->G, AD->E}
  - D2 = {AB, BD, ABD, EG}
    - Check lossy or lossless?
  - D3 = {AB, BD, ACDE, EG}
    - Check lossy or lossless?

Decomposition

- D3 = {AB, BD, ACDE, EG} lossy



Equivalence of Functional Dependency

- Equivalence of Functional Dependency
  - F & D two functional dependencies are logical equivalent (F≡G) if
    - F covers G, or G $\subseteq$ F : every FD's of G implied in F

      and
    - G covers F, or F $\subseteq$ G : every FD's of F implied in G
- Simple Method
  - To check G $\subseteq$ F
    - For each FD in G, pick it's LHS and take closure from F
    - If all closure from F are able to determine FD in G then G $\subseteq$ F
  - To check F $\subseteq$ G
    - For each FD in F, pick it's LHS and take closure from G
    - If all closure from G are able to determine FD in F then F $\subseteq$ G

Equivalence of Functional Dependency

- Example: G: {A -> B, B -> C, A -> C}, F: {A -> B, B -> C}
  - First step to check G $\subseteq$ F: pick A from A->B FD in G, find closure of A by looking FDs in F. A^+ = ABC; Here A is able to determine B. First dependency in G, A->B is satisfied.
    - Repeat the process for all FD in G and if all dependencies in G are satisfied by F then we can say G $\subseteq$ F
  - Similarly, we can check if F $\subseteq$ G
  - If both are satisfied, we can say F$\equiv$G
- **G $\subseteq$ F ensures that all the dependencies in G are present in F**
- **F $\subseteq$ G ensures that all the dependencies in F are present in G**

Equivalence of Functional Dependency

- We may find logical equivalence by looking FD's of F and G
- Another Example: F: {AB -> CD, B ->C, C -> D}, G: {AB ->C, AB -> D, C -> D}
- logical equivalent (F≡G) ? G $\subseteq$ F Yes or No; F $\subseteq$ G Yes or No

Dependency Preserving Decomposition

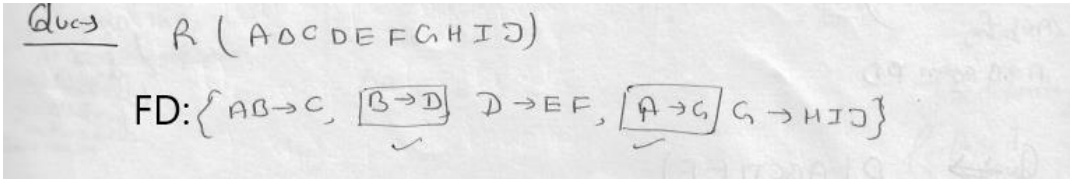
- If a relational schema R with functional dependency set F is decomposed into R_1, R_2,…, R_n with functional dependencies F_1, F_2,…, F_n in such a way that $\{F_1 \cup F_2 \cup \ldots\ldots \cup F_n\} \equiv F$
  - Or DDR $\equiv$ F where, DDR is dependencies in decomposed relations $\{F_1 \cup F_2 \cup \ldots\ldots \cup F_n\}$
    - Then we say that the decomposition is a dependency preserving decomposition.

Dependency Preserving Decomposition

- Example 1
  - $R = (A, B, C)$
    $F = \{A \rightarrow B, B \rightarrow C\}$
    Key = {A}. R is not in BCNF.
  - How to do decomposition and what will be attributes in decomposed relations?
    - First, the dependency that is creating a problem, make a separate relation (r2) for that. And attributes of that separate relation will be generated by taking closure of attribute present on the left side of problem creating dependency
    - Now, we make a new separate relation (r1) and attribute of this new relation will be remaining attributes that are not present in first relation (r2) and candidate key of first relation (r2)
  - Decomposition $R_2 = (B, C)$ $FD2 = \{B \rightarrow C\}$; $R_1 = (A, B)$ $FD_1 = \{A \rightarrow B\}$
    - R_1 and R_2 are in BCNF
    - Dependency preserving and Lossless-join decomposition

Attributes of Decomposed Relations

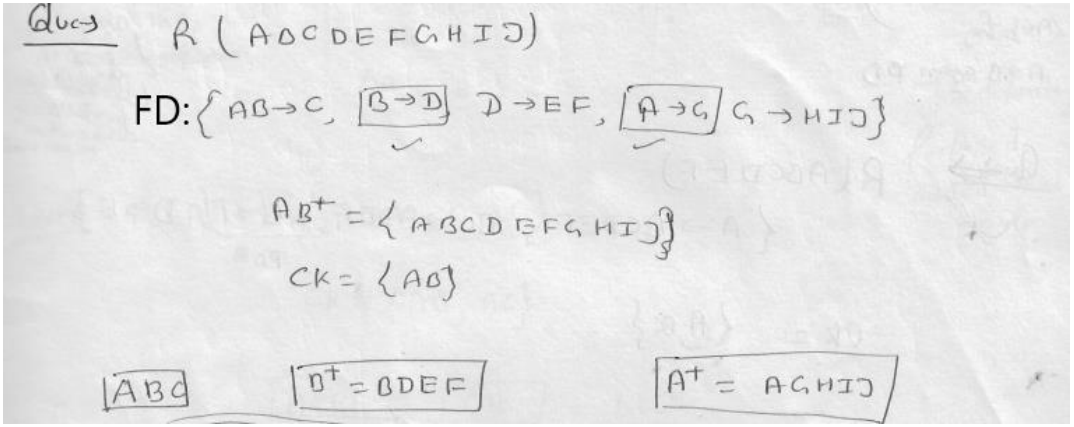
- We want to convert the following relation into 2nd NF:



- How many relations, we need to create?
- What will be attributes of those relations?
- Why are we doing this way?

Attributes of Decomposed Relations

- Solution of the last example:



- Total Relations: 3
- Attributes
  - R1 ={BDEF}, R2 = {AGHIJ}, R3 = {ABC}
    - Is it a lossy or lossless decomposition?

Dependency Preserving Decomposition

- Example 2
  - R = (*A, B, C, D*)

    F = {$AB \rightarrow CD$, $D \rightarrow A$}

    Key = {*AB*, *DB*}. R is not in BCNF
  - Decomposition R_1 =(*A, D*), FD_1 ={*D -> A*}; R_2 =(*B, C, D*) , *FD2 ={BD -> C}*
    - R_1 and R_2 are in BCNF
    - Not dependency preserving (DP) but lossless decomposition
  - Will follow simple method from next slide to find FDs of decomposed relations and to check if decomposition is a DP decomposition

Dependency Preserving Decomposition

- Method to find functional dependencies in decomposed relation (DDR) or DDR $\subseteq$ F
  - Place all the possible dependencies that are satisfying original relation dependency, F
    - First, place maximum number of possible dependencies in decomposed relation
    - For each FD in decomposed relation, pick it's LHS and take closure from original relation FD, F
      - If closure from original relation is able to determine FD in decomposed relation then we keep this dependency otherwise, discord it
- Method to check if decomposed FDs cover original relation or F $\subseteq$ DDR
  - For each FD in original relation, pick it's LHS and take closure from decomposed relation FD
    - If closure from decomposed relation is able to determine FD in original relation then we say that it is a dependency preserving decomposition

Dependency Preserving Decomposition

- See Example 1 from the earlier slide
  - *R* = (*A, B, C*); *F* = {*A* → *B, B* → *C*}; Key = {*A*} and *R* is not in BCNF
  - Decomposition R_1 = *(A, B) and* R_2 = *(B, C),*

R1(AB)	R2(BC)
A -> B possible by looking original FD F	B -> C possible
B -> A not possible	C -> B not possible

- Functional dependency of R1, FD_1 = {A -> B}; Functional dependency of R2, FD2 = {B -> C}
- R_1 and R_2 are in BCNF
- Total Dependencies in Decomposed Relation (DDR) = {FD1 U FD2} = {A -> B , B -> C }
- Now, we will check if DDR covers F, (F ⊆ DDR). Valid.
  - So it is a dependency preserving decomposition
  - Also it is a Lossless decomposition

Dependency Preserving Decomposition

- From earlier Slide, check if Example 2 is DP Decomposition
- Example 2
  - R = (*A, B, C, D*)

    F = {$AB \rightarrow CD$, $D \rightarrow A$}

    Key = {*AB*, *DB*}. R is not in BCNF
  - Decomposition R_1 =(A, D), FD_1 ={D -> A}; R_2 =(B, C, D) , FD2 ={BD -> C}
    - R_1 and R_2 are in BCNF
    - Not dependency preserving (DP) but lossless decomposition

Dependency Preserving Decomposition

- Check if Example 2 is DP Decomposition
  - R = (*A, B, C, D*)

    F = {$AB \rightarrow CD$, $D \rightarrow A$}

    Key = {*AB*, *DB*}. R not in BCNF. Decomposition R_1 =(*A, D*), R_2 =(*B, C, D*)

R1(AD)	R2(BCD)
A -> D not possible	B -> CD not possible
D -> A possible	D -> CB not possible
	C -> DB not possible
	BC -> D not possible
	CD -> B not possible
	BD -> C possible
	B->C,B->D,C->B,C->D,D->C,D->B not possible

  - Dependencies in Decomposed Relation(DDR) ={BD -> C, D -> A}
  - Now check, if F $\subseteq$ DDR; Not True
    - So it is not a dependency preserving decomposition

Decomposition

- Real-life Example: R(ID name salary dept_name building budget)
- FD : {ID -> name salary dept_name building budget, dept_name -> building budget}

Normal Form?

ID	name	salary	dept_name	building	budget
1	Ram	120000	CS	POD1A	7000000
2	Shyam	130000	EE	POD1B	5000000
3	Mohan	140000	CS	POD1A	7000000
4	Gopal	150000	EE	POD1B	5000000
5	Keshav	160000	CS	POD1A	7000000
6	Anuj	170000	ME	POD1C	2000000
7	Bharat	180000	IT	POD1D	6000000

Table: R

Table R is decomposed into R1 and R2

ID	name	salary	dept_name
1	Ram	120000	CS
2	Shyam	130000	EE
3	Mohan	140000	CS
4	Gopal	150000	EE
5	Keshav	160000	CS
6	Anuj	170000	ME
7	Bharat	180000	IT

Table: R1

R1 (ID name salary dept_name), FD1 : {ID -> name salary dept_name}

Normal Form?

dept_name	building	budget
CS	POD1A	7000000
EE	POD1B	5000000
ME	POD1C	2000000
IT	POD1D	6000000

Table: R2

R2 (dept_name building budget), FD2: {dept_name -> building budget}

Normal Form?

Decomposition into Higher Normal Forms

- R(ABCDE), FD(R) : {AB -> C, C -> D, B -> E}
  - Find CK(R)?
  - Find PA and NPA?
  - Find the dependency that is creating problem?
  - Decompose the relation into sub-relations
    - Decomposed relations should satisfy
      - Lossless decomposition
      - Dependency preserving decomposition
    - How many relations would be there if we decompose this relation into BCNF?

Decomposition into Higher Normal Forms

- R(ABCDE), FD(R) : {AB -> C, C -> D, B -> E}
  - Find CK(R) = {AB}
  - Due to partial dependency, its not in 2nd NF
  - Decompose R: R1(ABCD), R2(BE)
    - FD(R1): {AB -> C, C -> D}, FD(R2): {B -> E}
    - CK(R1) = {AB}, CK(R2) = {B}
    - Due to transitive dependency, R1 is not in 3rd NF
    - Decompose R1 : R3(ABC), R4(CD)
      - FD(R3): {AB -> C}, FD(R4): {C -> D}
      - CK(R3) = {AB}, CK(R4) = {C}
      - Now relation is in BCNF

References

- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. Vol. 6. New York: McGraw-Hill, 1997.
- Ramez Elmasri, Shamkant B. Navathe. *Fundamentals of Database Systems.* Edition 6. Pearson, 2010.