Module 4: Database Design Theory and Normalization

Introduction to Information Systems

The University of Queensland Australia
CREATE CHANGE

In This Module

- How can we measure the quality of database design?
- What is a functional dependency (FD) constraint?
- How do we reason with FDs?
- What is a normal form (NF)?
- How do you achieve a (higher) NF?

Learning Outcomes

- After successfully completing this module you should be able to reason with the logical foundation of the relational data model and understand the fundamental principles of correct relational database design.

  - Provide examples of modification, insertion, and deletion anomalies.
  - Given a set of functional dependencies that hold over a table, determine associated keys and superkeys.
  - Given a set F of functional dependencies and set X of attributes of a relation, compute the closure of X, which is X^+.
  - Justify why lossless-join decompositions are preferred decompositions.
  - Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 1NF, 2NF, 3NF, BCNF.
  - Given a universal relation schema R and a set of functional dependencies decompose into a set of lossless 3NF or BCNF relationships.

Design Guidelines
Functional Dependencies
Normalization
Relational Database Schema Design

Informal Design Guidelines

Informal measures of relational database schema quality and design guidelines

1. Making sure that the semantics of the attributes is clear in the schema.

2. Reducing the redundant values in tuples.

3. Reducing the null values in tuples.

4. Disallowing the possibility of generating spurious tuples.

Guideline 1

Design each relation so that it is easy to explain its meaning

Using meaningful names

Do not combine attributes from multiple entity types and relationship types into a single relation

- This can make the meaning of an entity type confusing
- This can also lead to redundancy

EMPLOYEE F.K.

Ename	Ssn	Bdate	Address	Dnumber

P.K.

DEPARTMENT F.K.

Ename	Dnumber	Dmgr_ssn

P.K.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

Redundant Values in Tuples

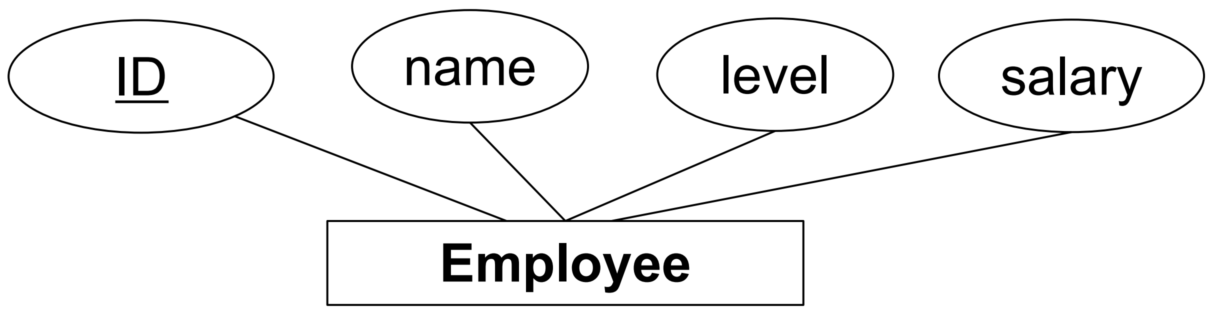
- One design goal is to minimise the storage space that base relations occupy.

- The way in which attributes are grouped into relational schema has a significant impact on storage space.

- In addition, an incorrect grouping may cause update anomalies which may result in inconsistent data or even loss of data.

A Motivating Example

Let’s consider an example of a company where an employee’s salary directly corresponds to the level or position, they hold. For example, a manager has a fix salary of $70,00 and a developer has a fix salary of $60,000.



ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

Update Anomalies

Employee

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

Modification Anomalies

- Updating the Salary of one developer, makes the “Developer” salary inconsistent.

Deletion Anomalies

- By deleting ”Charlie” we no longer store the salary of “Administration” Staff.

Insertion Anomalies

- We cannot store the salary of a “Cook” if no employee has that position.
- Inserting a new row with a different Salary for a developer, makes the “Developer” salary inconsistent.

Guideline 2

Design the base relation schema so that no insertion, deletion, or modification anomalies occur in the relations

- If any do occur, ensure that all applications that access the database update the relations in such a way as to not compromise the integrity of the database

Guideline 3

As far as possible, avoid placing attributes in a base relation whose values may be null

- Null values waste storage space, introduce ambiguity, and cannot be used for comparison

- If nulls are unavoidable, make sure that they apply in exceptional cases only in the relation

Decomposing a Relation

- A decomposition of R replaces R by two or more relations such that:
  - Each new relation contains a subset of the attributes of R (and no attributes not appearing in R)
  - Every attribute of R appears in at least one new relation.

R

A	B	C
1	2	3
4	5	6
7	8	9

decomposition →

R_1

A	B
1	2
4	5
7	8

R_2

B	C
2	3
5	6
8	9

Decomposing a Relation Example

Employee

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

ID	Name	Salary
1	Paris	60,000
2	Anna	70,000
3	Ben	70,000
4	Rose	50,000
5	Jack	60,000
6	Charlie	50,000

Salary	Level
60,000	Developer
70,000	Manager
50,000	Driver
50,000	Administration

ID	Name	Level
1	Paris	Developer
2	Anna	Manager
3	Ben	Manager
4	Rose	Driver
5	Jack	Developer
6	Charlie	Administration

Level	Salary
Developer	60,000
Manager	70,000
Driver	50,000
Administration	50,000

How should we decompose relations to remove anomalies?

The Join Operation

Definition: R1⋈ R2 is the natural join of the two relations

- Each tuple of R1 is concatenated with every tuple in R2 having the same values on the common attributes

R_1

A	B
1	2
4	5
7	8

R_2

B	C
2	3
5	6
8	9

$R_1 \bowtie R_2$ →

A	B	C
1	2	3
4	5	6
7	8	9

Lossless Join Decomposition

Decomposition of R into R_1 and R_2 is a lossless join *if for every instance r that satisfies F*:

We will discuss F and r satisfying F later

$$R = R_1 \bowtie R_2$$

Informally: If we break a relation R into bits, when we put the bits back together, we should get exactly R back again.

Example Lossy Join Decomposition

R

A	B	C
1	2	3
4	5	6
7	2	9

Decompose

R_1

A	B
1	2
4	5
7	2

R_2

B	C
2	3
5	6
2	9

Join

R'

A	B	C
1	2	3
1	2	9
4	5	6
7	2	3
7	2	9

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

- Maybe a better term here is "addition of spurious information"

...there are two extra rows. Why? What if B determined C?

Guideline 4

Design the relation schemas so that they can be (relationally) joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated

Design Guidelines

Functional Dependencies

Normalization

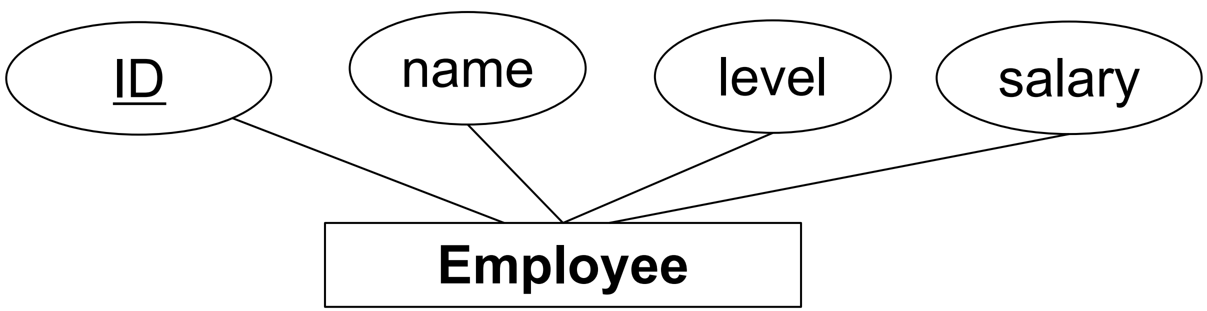
Relational Database Schema Design

Functional Dependency

How can we know for sure if all employee members appointed at the same level have the same salary?

Databases allow you to say that one attribute determines another through a functional dependency

Assume level determines salary but not id. We say that there is a functional dependency from level to salary (i.e., if we know an employee's level, we can find their salary)



Functional Dependency, Formally

A functional dependency (FD) X→Y holds on relation R if for every legal instance of R such as *r*, for all tuples t1, t2:

if t1[X] = t2[X] → t1[Y] = t2[Y]

- Which means given two tuples in *r*, if their X values agree, then their Y values must also agree
- Example: level → salary (i.e., if two employees have the same level, then they must have the same salary)

An FD X → Y is a constraint between two sets of attributes X and Y in a relational schema R

- It specifies a restriction on the possible tuples that can form a relation instance of R

Identifying Functional Dependencies

A FD is a statement about all allowable instances

- Must be identified by application semantics
- Given some instance of R, we can check if it violates some FD *f* but we cannot tell if *f* holds over R!

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

Based on this instance alone, we cannot conclude that level → salary

How can we find them then? Using our knowledge of the system or the UoD.

Question: Functional Dependencies

Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

Which FDs cannot be true given the instance above?

A. B→ C
B. B→ D
C. D→ B
D. All of the above can be true
E. None of the above can be true

Fixing Anomalies

Employee

level → salary

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

ID	Name	Level
1	Paris	Developer
2	Anna	Manager
3	Ben	Manager
4	Rose	Driver
5	Jack	Developer
6	Charlie	Administration

Level	Salary
Developer	60,000
Manager	70,000
Driver	50,000
Administration	50,000

Anomalies Fixed?

Level → Salary

ID	Name	Level
1	Paris	Developer
2	Anna	Manager
3	Ben	Manager
4	Rose	Driver
5	Jack	Developer
6	Charlie	Administration

Level	Salary
Developer	60,000
Manager	70,000
Driver	50,000
Administration	50,000
Cook	50,000

Modification Anomalies Fixed

- Updating the Salary of one developer no longer makes the “Developer” salary inconsistent.

Deletion Anomalies Fixed

- Deleting ”Charlie” no longer removes the salary of “Administration” Staff.

Insertion Anomalies Fixed

- We can now store the salary of a “Cook” without any employees holding that position.
- Inserting a new row with a different Salary for a developer is no longer possible

General Anomaly Fixing

We were able to fix the anomalies by splitting the Staff table into two tables

In the rest of this module, we will discuss how anomalies can be addressed formally

Question: Anomalies

Consider the following relation R with the given functional dependencies.

R[A, B, C, D]:
D → AC

A	B	C	D
1	4	2	5
2	3	4	3
1	1	2	5

Which of the following is not an example of an update anomaly?

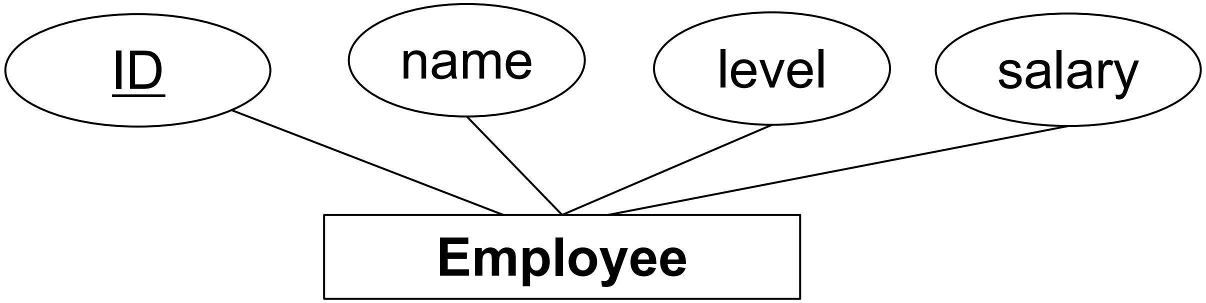
A. Deleting <2, 3, 4, 3> from R
B. Inserting values <3, 5, 3, 3> into R
C. Modifying <1, 1, 2, 5> to <1, 2, 2, 5> in R
D. Inserting values <1, null, 2, 4> into R
E. Modifying <1, 1, 2, 5> to <1, 2, 3, 5> in R

Keys

A key is a minimal set of attributes that uniquely identify a relation

- i.e., a key is a minimal set of attributes that functionally determines all the attributes in the relation

A superkey for a relation uniquely identifies the relation



- Example key: {ID}
- Example superkey: {ID, level}

Question: Possible Keys

Assume that the following FDs hold for a relation R(A,B,C,D):

B→C

C→B

D→ABC

Which of the following is a key for the above relation?

A. B
B. C
C. BD
D. All of the above
E. None of the above

Question: Possible Superkeys

Assume the following FDs hold for relation R(A,B,C,D):

B→C

C→B

D→ABC

Which of the following is a superkey for the above relation?

A. D
B. BD
C. BCD
D. All are superkeys
E. None are superkeys

Explicit and Implicit FDs

Given a set of (explicit) functional dependencies, we can determine implicit ones

- Explicit FDs: ID→ level, level→ salary
- Implicit FD: ID→ salary

Implicit FDs are also called inferred FDs

The notation $F \models X \rightarrow Y$ denotes that FD $X \rightarrow Y$ can be inferred from the set of functional dependencies F

- X is called the left-hand side (LHS)
- Y is called the right-hand side (RHS)
- Y can be derived from X under F
- X implies Y under F

Closure of F

Given a set F of FDs, many implicit FDs can be derived

- Everything implies itself, such as A → A, ABC → AB
- These are called trivial FDs (i.e., an FD is trivial if the LHS contains the RHS)
- The inference of trivial FDs does not depend on any F

Non-trivial FDs, such as A → B, A → AB

- The inference of non-trivial FDs depends on a given F

Closure of F (denoted as F^+): the set of all FDs that can be implied by F

- F^+ includes both trivial and non-trivial FDs

Inference Rules

It is practically impossible to specify all possible FDs that may hold

To infer additional FDs from a set of valid FDs we need a system of inference rules

- There are 6 inference rules
- The first three rules are referred to as Armstrong's Axioms

Armstrong's Axioms

Armstrong's Axioms (X, Y, Z are sets of attributes):

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
  - e.g., {ID, level} → level
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - e.g., if ID → level then {ID, salary} → {level, salary}
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
  - e.g., if ID → level and level → salary then ID → salary

These three rules are *sound* and *complete*

- **Sound**: Given a set of FDs F, holding on a relation schema R, any FD that we can infer from F by using these three rules holds in every legal relation instance
- **Complete**: Repeatedly applying these three rules generates all possible FDs that can be inferred from F

Additional Rules

The following rules are frequently used for convenience, but can be derived using Armstrong's Axioms

- Decomposition: if X → YZ then X → Y
  - e.g., if ID → {level, salary} then ID → level and ID → salary
- Union: if X → Y and X → Z then X → YZ
  - If ID → level and ID → salary then ID → {level, salary}
- Pseudotransitivity: if X → Y and WY → Z then WX → Z
  - If level → salary and {awards, salary} → raise then {awards, level} → raise

Proof of Decomposition Rule

To prove IR4: $X \rightarrow YZ \vDash X \rightarrow Y$

IR1 Reflexivity: $Y \subseteq X \vDash X \rightarrow Y$
IR2 Augmentation: $X \rightarrow Y, \vDash XZ \rightarrow YZ$
IR3 Transitivity: $\{X \rightarrow Y, Y \rightarrow Z\} \vDash X \rightarrow Z$

Proof:

1: $X \rightarrow YZ$: Given

2: $YZ \rightarrow Y$: Using IR1 (Y is a subset of YZ)

3: $X \rightarrow Y$: Using IR3 on results of 1 & 2

Proof of Union Rule

To prove IR5: $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

IR1 Reflexivity: $Y \subseteq X \models X \rightarrow Y$
IR2 Augmentation: $X \rightarrow Y, \models XZ \rightarrow YZ$
IR3 Transitivity: $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

Proof:

1: $X \rightarrow Y$: Given

2: $X \rightarrow Z$: Given

3: $XX \rightarrow XY$: Using IR2 on 1 (adding X)

4: $X \rightarrow XY$: Since XX=X (not using IR1-R3)

5: $XY \rightarrow YZ$: Using IR2 on 2 (adding Y)

6: $X \rightarrow YZ$: Using IR3 on 4 & 5

Proof of Pseudotransitive Rule

To prove IR6: $\{X \rightarrow Y, WY \rightarrow Z\} \vDash WX \rightarrow Z$

IR1 Reflexivity: $Y \subseteq X \vDash X \rightarrow Y$
IR2 Augmentation: $X \rightarrow Y, \vDash XZ \rightarrow YZ$
IR3 Transitivity: $\{X \rightarrow Y, Y \rightarrow Z\} \vDash X \rightarrow Z$

Proof

1: $X \rightarrow Y$: Given

2: $WY \rightarrow Z$: Given

3: $WX \rightarrow WY$: Using IR2 on 1 (adding W)

4: $WX \rightarrow Z$: Using IR3 on 3 & 2

The Six Inference Rules

IR1

Reflexivity:
$Y \subseteq X \vDash X \rightarrow Y$

IR2

Augmentation:
$X \rightarrow Y \vDash XZ \rightarrow YZ$

IR3

Transitivity:
$\{X \rightarrow Y, Y \rightarrow Z\} \vDash X \rightarrow Z$

IR4

Decomposition:
$X \rightarrow YZ \vDash X \rightarrow Y$

IR5

Union:
$\{X \rightarrow Y, X \rightarrow Z\} \vDash X \rightarrow YZ$

IR6

Pseudotransitivity:
$\{X \rightarrow Y, WY \rightarrow Z\} \vDash WX \rightarrow Z$

F^+ and X^+

F^+ is all FDs which can be derived from F

- The six inference rules can be used to compute F^+
- Too many and too time-consuming to compute
- And not necessary

X^+ is the set of attributes determined by X under F which is called the closure of X

- e.g., ID^+ = {name, level, salary}
- Computing X^+ is easy

Computing X^+ (Informally)

1. X^+ initially contains all attributes in X
2. For each FD in the set F:

   If the LHS of the FD is a subset of X^+ then add the RHS to X^+
3. If step 2 resulted in changes in X^+ then repeat 2, otherwise finish

Example

Employee (ID, level, salary), F{ID→ level, level→ salary, ID→ name}

1. ID^+ = {ID}
2. ID^+ = {ID, level} using ID→ level
3. ID^+ = {ID, level, salary} using level→ salary
4. ID^+ = {ID, name, level, salary} using ID→ name

Computing X^+

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

Example

Employee (ID, level, salary), F{ID→ level, level→ salary, ID→ name}

1. $ID^+ = \{ID\}$
2. $ID^+ = \{ID, level\}$ using ID→ level
3. $ID^+ = \{ID, level, salary\}$ using level→ salary
4. $ID^+ = \{ID, name, level, salary\}$ using ID→ name

Example X^+ Computation

R (pNumber, pName, pLocation, dNum, dName, mgrSSN, mgrStartDate)

F = {

pNumber → {pName, pLocation, dNum},

dNum → {dName, mgrSSN, mgrStartDate}

}

X = {pNumber}

1. X^+ := {pNumber}
2. X^+ := {pNumber, pName, pLocation, dNum}
3. X^+ := {pNumber, pName, pLocation, dNum, dName, mgrSSN, mgrStartDate}

Example: Supplier-Part DB

Suppliers supply parts to projects.

- SupplierPart (sName, city, status, pNum, pName, qty)
  - supplier attributes: sName, city, status
  - part attributes: pNum, pName
  - supplier-part attributes: qty

Functional dependencies:

- fd1: sName → city
- fd2: city → status
- fd3: pNum → pName
- fd4: sName, pNum → qty

Example Finding Superkey

Exercise: Show that (sName, pNum) is a **superkey** for SupplierPart

SupplierPart (sName, city, status, pNum, pPame, qty)

F = {
    sName → city,
    city → status,
    pNum → pName,
    {sName, pNum} → qty,
}

- $\{sName, pNum\}^+ = \{sName, pNum\}$
- $\{sName, pNum\}^+ = \{sName, pNum, city\}$ — Using sName → city
- $\{sName, pNum\}^+ = \{sName, pNum, city, status\}$ — Using city → status
- $\{sName, pNum\}^+ = \{sName, pNum, city, status, pName\}$ — Using pNum → pName
- $\{sName, pNum\}^+ = \{sName, pNum, city, status, pName, qty\}$ — Using {sName, pNum} → qty

A Note About Finding Keys

Given: A complete set of FDs F on relation R

Approach: for any subset S of attributes in R, S is a key iff (1) $S^+ = R$, and (2) there is no $S' \subset S$ such that $S'^+ = R$. If R has n attributes, there exist 2^n subsets to consider

Tips for finding keys:

- If an attribute does not appear on the RHS of any FDs in F, a key must contain that attribute
- If a subset S is a key, there is no need to test any superset of S (they must be a superkey and cannot be a key)
- One relation can have multiple keys of different length
  - For example, if ABC is a key, ABCD cannot be a key, but ABDE can also be a key

Example Finding Key

Exercise: Show that (sName, pNum) is a **key** for SupplierPart

SupplierPart (sName, city, status, pNum, pPame, qty)

F = {
sName → city,
city → status,
pNum → pName,
{sName, pNum} → qty,
}

- Show $\{sName, pNum\}^+$ = {sName, pNum, city, status, pName, qty}
- Show sName is not key $\{sName\}^+$ = {sName, city, status}
- Show pNum is not key $\{pNum\}^+$ = {pNum, pName}

Question: Finding Keys

Assume that the following FDs hold for a relation R(ABCDEF):

B→CF

C→E

EF→D

Which of the following is a key for the above relation?

A. B
B. BE
C. EF
D. AB
E. None of the above

Question: Finding Key

Which of the following is a key of relation R(ABCDE) with $F = \{D \rightarrow C, CE \rightarrow A, D \rightarrow A, AE \rightarrow D\}$

A. ABDE
B. BCE
C. CDE
D. All of these are keys
E. None of these are keys

Question: Finding Keys

Assume that the following FDs hold for a relation R(ABCDEF):

AB→E

C→BE

ED→F

Which of the following is a key for the above relation?

A. AB
B. ABC
C. ACD
D. AD
E. None of the above

Approaching Normality

Role of FDs in detecting redundancy:

- Consider a relation R with 3 attributes, A, B, C
  - No FDs hold: There is no redundancy here
  - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Normalization: the process of removing redundancy from data

We were able to fix the anomalies by splitting (decomposing) the Employee table discussed before, but how should we do this more formally? and how is it related to functional dependencies?

Design Guidelines

Functional Dependencies

Normalization

Relational Database Schema Design

Normalization

Normalization is a process that aims at achieving better designed relational database schemas using

- Functional Dependencies
- Primary Keys

The normalization process takes a relational schema through a series of tests to certify whether it satisfies certain conditions

- The schemas that satisfy certain conditions are said to be in a given Normal Form
- Unsatisfactory schemas are decomposed by breaking up their attributes into smaller relations that possess desirable properties (e.g., no anomalies)

“Key” Concepts

Superkey: a set of attributes such that no two tuples have the same values for these attributes

- That is, its closure contains all attributes in the relation

(Candidate) key: a minimal superkey

- Minimal ≠ shortest
- There can be many candidate keys for one relation
- Any superkey must contain at least one candidate key

Primary key: one of the selected candidate keys

- There can be only one primary key for a relation

Prime attribute: an attribute in any candidate key

- Not necessarily a member of the primary key!

Non-prime attribute: An attribute that is not a member of any candidate key

The University of Queensland Australia

Normalization – Overview

1NF

- **Outcome**: Removal of non-atomic values from relations.
- **Test**: Relation should have no multivalued attributes or nested relations.

2NF

- **Outcome**: Removal of partial dependencies, which remove some anomalies.
- **Test**: LHS of any non-trivial FD in F^+ is not a proper subset of a candidate key or RHS is a prime attribute

3NF

- **Outcome**: Removal of partial and transitive dependencies, which remove most anomalies
- **Test**: LHS of any non-trivial FD in F^+ is a superkey or RHS is a prime attribute.

BCNF

- **Outcome**: Removal of all anomalies at the cost of not preserving all FDs
- **Test**: LHS of any non-trivial FD in F^+ is a superkey.

Normalization – Overview

Employee

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

Level → Salary

1NF

2NF

3NF

BCNF

Normalization – Overview

1NF

- **Outcome**: Removal of non-atomic values from relations.
- **Test**: Relation should have no multivalued attributes or nested relations.

2NF

- **Outcome**: Removal of partial dependencies, which remove some anomalies.
- **Test**: LHS of any non-trivial FD in F^+ is not a proper subset of a candidate key or RHS is a prime attribute

3NF

- **Outcome**: Removal of partial and transitive dependencies, which remove most anomalies
- **Test**: LHS of any non-trivial FD in F^+ is a superkey or RHS is a prime attribute.

BCNF

- **Outcome**: Removal of all anomalies at the cost of not preserving all FDs
- **Test**: LHS of any non-trivial FD in F^+ is a superkey.

First Normal Form (1NF)

A relation schema is in 1NF if the domains of the attributes include only atomic (simple, indivisible) values

- The value of an attribute is a single value from the domain of that attribute
- 1NF disallows having a set of values, a tuple of values (nested attributes), or a combination of both

Examples of Non-1NF Relations

customerName	orderNum	items
Tom Jones	123	Hat
Sri Gupta	876	Glass, Pencil

name		orderNum	item
firstName	familyName		
Tom	Jones	123	Hat
Sri	Gupta	876	Glass
Sri	Gupta	876	Pencil

Normalized to 1NF

customerName	orderNum	item
Tom Jones	123	Hat
Sri Gupta	876	Glass
Sri Gupta	876	Pencil

customerName	orderNum	item1	item2
Tom Jones	123	Hat	null
Sri Gupta	876	Glass	Pencil

Problems:

- Above: redundancy
- Below: lack of flexibility

Normalization – Overview

1NF

- **Outcome**: Removal of non-atomic values from relations.
- **Test**: Relation should have no multivalued attributes or nested relations.

2NF

- **Outcome**: Removal of partial dependencies, which remove some anomalies.
- **Test**: LHS of any non-trivial FD in F^+ is not a proper subset of a candidate key or RHS is a prime attribute

3NF

- **Outcome**: Removal of partial and transitive dependencies, which remove most anomalies
- **Test**: LHS of any non-trivial FD in F^+ is a superkey or RHS is a prime attribute.

BCNF

- **Outcome**: Removal of all anomalies at the cost of not preserving all FDs
- **Test**: LHS of any non-trivial FD in F^+ is a superkey.

Full and Partial Functional Dependencies

A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold anymore.

- $A \in X$, (X - {A}) does not functionally determine Y

A functional dependency X → Y is a **partial dependency** if some attribute A can be removed from X and the dependency still holds.

- $A \in X$, (X - {A}) → Y

Example

Address [houseNum, street, postcode, state, value]

F = {{houseNum, street, postcode} → {state, value}, postcode → state}

- {houseNum, street, postcode} → value **full functional dependency**
  - Since no subset of {houseNum, street, postcode} determines value
- {houseNum, street, postcode} → state **partial dependency**
  - Since postcode → state

Second Normal Form (2NF)

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

Example

Address [houseNum, street, postcode, state, value]

F = {{houseNum, street, postcode} → {state, value}, postcode → state}

- Key = {houseNum, street, postcode}
- State is not a prime attribute
- State is partially dependent on the primary key (postcode → state)

FD postcode → state violates 2NF; therefore, Address is not in 2NF

…2NF can be said informally to have no partial dependency

Second Normal Form – More Formally

A relation schema R with the set F of FDs is in 2NF iff

- For all subset of attributes $X \subset R$
- and, for all attributes $A \in R$
- such that for any non-trivial FD: $X \rightarrow A$ in F^+

Then

1. X is NOT a proper subset of a candidate key for R, or
2. A is a prime attribute

Example

- Address [houseNum, street, postcode, state, value]
- F = {{houseNum, street, postcode} → {state, value}, postcode → state}

Consider X → A where X = postcode and A = state

- 1) state is not a prime attribute, and
- 2) postcode → state is a non-trivial FD, and
- 3) postcode is a proper subset of {houseNum, street, postcode}, which is a candidate key

Example Normalized to 2NF

Example

- Address [houseNum, street, postcode, state, value]
- F = {{houseNum, street, postcode} → {state, value},

  postcode → state}

Normalizing Address

- Address [<u>houseNum, street, postcode</u>, value]
- Postcodes [<u>postcode</u>, state]

  Address.postcode → Postcodes.postcode

2NF eliminates anomalies due to **partial dependencies**. However, 2NF does not eliminate anomalies which are due to **transitive dependencies**

Transitive Dependency

A functional dependency X → Y in a relation R is a **transitive dependency** if there exists a set of attributes Z in R such that:

- Z is neither a candidate key nor a subset of any key of R.
- Both X → Z and Z → Y

Example

- Employee [<u>ID</u>, name, level, salary]
- level → salary

Consider the FD ID → salary

- There exists an attribute “level” which is neither a candidate key nor a subset of any key
- ID → level and level → salary hold
- ID → salary is a transitive dependency

Problems with 2NF Relations

Employee [ID, name, level, salary]

- level → salary

level → salary does not violate 2NF

- Staff is in 2NF

ID	Name	Level	Salary
1	Paris	Developer	60,000
2	Anna	Manager	70,000
3	Ben	Manager	70,000
4	Rose	Driver	50,000
5	Jack	Developer	60,000
6	Charlie	Administration	50,000

Modification Anomalies

- Updating the Salary of one developer, makes the “Developer” salary inconsistent.

Deletion Anomalies

- By deleting ”Charlie” we no longer store the salary of “Administration” Staff.

Insertion Anomalies

- We cannot store the salary of a “Cook” if no employee has that position.
- Inserting a new row with a different Salary for a developer, makes the “Developer” salary inconsistent.

Normalization – Overview

1NF

- **Outcome**: Removal of non-atomic values from relations.
- **Test**: Relation should have no multivalued attributes or nested relations.

2NF

- **Outcome**: Removal of partial dependencies, which remove some anomalies.
- **Test**: LHS of any non-trivial FD in F^+ is not a proper subset of a candidate key or RHS is a prime attribute

3NF

- **Outcome**: Removal of partial and transitive dependencies, which remove most anomalies
- **Test**: LHS of any non-trivial FD in F^+ is a superkey or RHS is a prime attribute.

BCNF

- **Outcome**: Removal of all anomalies at the cost of not preserving all FDs
- **Test**: LHS of any non-trivial FD in F^+ is a superkey.

Third Normal Form (3NF)

A relation schema R with the set F of FDs is in 3NF iff

- For all subset of attributes $X \subset R$
- and, for all attributes $A \in R$
- such that for any non-trivial FD: $X \rightarrow A$ in F^+

Then

1. X is a superkey for R, or
2. A is a prime attribute

In other words, a relation is in 3NF iff for any non-trivial FD $X \rightarrow A$ where A is a non-prime attribute, X must be a superkey

Example

- Employee [ID, name, level, salary]
  - level → salary (transitive dependency)
- level is not a superkey and salary is not a prime attribute
- level → salary violates 3NF; therefore Employee is not in 3NF

Example Normalized to 3NF

Example

- Employee [<u>ID</u>, name, level, salary]
  - level → salary (transitive dependency)

Normalizing Employee

- StaffAppointment [<u>ID</u>, name, level]
- StaffIncome [<u>level</u>, salary]

  StaffAppointment.level → StaffIncome.level

Most 3NF tables are free of anomalies; however, some 3NF tables, rarely met with in practice, are still affected by anomalies.

Question: partial dependency vs. transitive dependency

What is the difference between partial dependency and transitive dependency?

A. Partial dependency is where an attribute only depends on a subpart of the primary key to be identified. Normalizing to third normal form solves this. Transitive dependency is where a non-prime attribute depends on other non-prime attributes to be identified. Normalizing to second normal form solves this.

B. Partial dependency is where an attribute only depends on a subpart of the primary key to be identified. Normalizing to second normal form solves this. Transitive dependency is where a non-prime attribute depends on other non-prime attributes to be identified. Normalizing to third normal form solves this.

C. Partial dependency is where an attribute only depends on a subpart of the primary key to be identified. Normalizing to second normal form solves this. Transitive dependency is where a non-prime attribute depends on other non-prime attributes to be identified. Normalizing to third normal form solves this.

D. Partial dependency is where a non-prime attribute depends on other non-prime attributes to be identified. Normalizing to third normal form solves this. Transitive dependency is where an attribute only depends on a subpart of the primary key to be identified. Normalizing to second normal form solves this.

Problems with 3NF Relations

Consider the following example:

- Teach [studentID, courseID, lecturer]
- {studentID, courseID} → lecturer
- lecturer → courseID
  - Keys: {studentID, courseID}, {studentID, lecturer}
  - Teach is in 3NF

Teach

studentID	courseID	lecturer
1234	INFS1200	Jane

Existing anomalies

- **Deletion anomaly**
  - E.g., Deleting studentID 1234 would lead to the unwanted deletion of the lecturer of INFS1200
- **Insertion anomaly**
  - E.g., cannot store the lecturer of a course that doesn't have students.

Normalization – Overview

1NF

- **Outcome**: Removal of non-atomic values from relations.
- **Test**: Relation should have no multivalued attributes or nested relations.

2NF

- **Outcome**: Removal of partial dependencies, which remove some anomalies.
- **Test**: LHS of any non-trivial FD in F^+ is not a proper subset of a candidate key or RHS is a prime attribute

3NF

- **Outcome**: Removal of partial and transitive dependencies, which remove most anomalies
- **Test**: LHS of any non-trivial FD in F^+ is a superkey or RHS is a prime attribute.

BCNF

- **Outcome**: Removal of all anomalies at the cost of not preserving all FDs
- **Test**: LHS of any non-trivial FD in F^+ is a superkey.

Boyce-Codd Normal Form (BCNF)

A relation schema R with the set of F of FDs is in BCNF iff

- For all subset of attributes $X \subset R$
- and, for all attributes $A \in R$
- such that for any non-trivial FD: $X \rightarrow A$ in F^+

Then

1. X is a superkey for R

Informally: Whenever a set of attributes of R determine another attribute, it should determine all the attributes of R.

Example

- Teach [studentID, courseID, lecturer]
- {studentID, courseID} → lecturer
- lecturer → courseID
  - Keys: {studentID, courseID}, {studentID, lecturer}
  - Lecturer is not a superkey
  - lecturer → courseID violates BCNF; therefore, Teach is not in BCNF

BCNF Example

For the given relation schema R with the set of FDs F, determine whether or not R is in BCNF

$$R\ (A, B, C, D), \quad F = \{AD \rightarrow BC,\ B \rightarrow A\}$$

Find candidate keys: AD and BD

$B \rightarrow A$ is a non-trivial FD, B is not a superkey

$B \rightarrow A$ violates BCNF, therefore, R is not in BCNF

BCNF is Great, But…

- Guaranteed that there will be no redundancy of data
- Easy to understand (just look for superkeys)
- Easy to do

So what is the main problem with BCNF?

- It may not preserve all functional dependencies

Dependency Preservation

Given a relation R and a set F of FD, when R is decomposed into R_1, R_2, … R_n, F is decomposed into F_1, F_2, … F_n

- F_i contains all FDs in F with the attributes completely in R_i

R is dependency preserving If $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$

Example: R(ABCD) and F = {A→B, A→D}

- For R_1=(ABC) and R_2=(BCD), F_1 = {A→B}, F_2=∅
  - A→D is not preserved

- For R_1=(ABC) and R_2=(ABD), F_1 = {A→B}, F_2={A→D}
  - All FDs in F are preserved

Problems with BCNF Relations

studentID	courseID	lecturer

{studentID, courseID} → lecturer
lecturer → courseID

- This relation in not in BCNF

Is lecturer a key?

studentID	lecturer

<u>lecturer</u>	courseID

lecturer → courseID

- The new relations no longer violate BCNF
- {studentID, courseID} → lecturer is no longer preserved

So What's the Problem?

<u>lecturer</u>	courseID
John	INFS1200
Jane	INFS1200

lecturer → courseID

studentID	lecturer
1234	John
1234	Jane

No problem so far. All *local* FD's are satisfied.
Let's join the relations back into a single table again:

<u>studentID</u>	<u>courseID</u>	lecturer
1234	INFS1200	John
1234	INFS1200	Jane

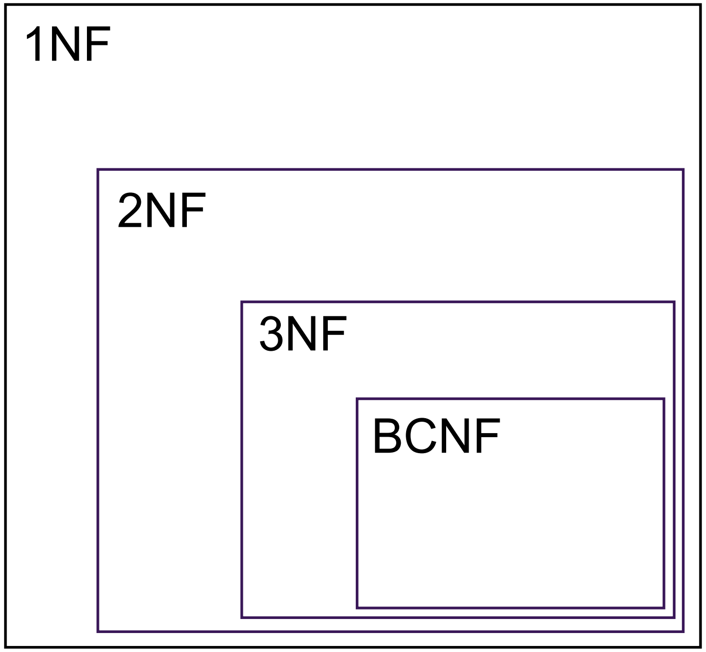
Violates the FD: studentID, courseID → lecturer

Decomposition into BCNF may lead to dependencies not being preserved.

BCNF $\subset$ 3NF $\subset$ 2NF $\subset$ 1NF

For a relation R with FD set F, for any non-trivial $X \rightarrow A$ in F^+

- **1NF:** Removal of non-atomic values from relations
  - Relation should have no multivalued attributes or nested relations
- **2NF:** Removal of partial dependencies
  - X is not a proper subset of a candidate key for R, or A is a prime attribute
- **3NF:** Removal of partial and transitive dependencies
  - X is a superkey for R, or A is a prime attribute
- **BCNF:** Removal of all anomalies at the cost of not preserving all FDs
  - X is a superkey for R



Question: Validating Normal Form

Assume that the following FDs hold for a relation R(ABCDEF):

$AB \rightarrow CDE$

$C \rightarrow F$

$E \rightarrow AB$

$X \rightarrow A$

2NF:

X is not a proper subset of a candidate key for R, or

A is a prime attribute

3NF:

X is a superkey for R, or

A is a prime attribute

BCNF:

X is a superkey for R

What is the highest normal form for the above relation?

A. 1NF
B. 2NF
C. 3NF
D. BCNF

Question: Validating Normal Form

Assume that the following FDs hold for a relation R(ABCD):

AB→CD

CD→A

D→B

> 2NF: X → A
>
> X is not a proper subset of a candidate key for R, or
>
> A is a prime attribute
>
> 3NF:
>
> X is a superkey for R, or
>
> A is a prime attribute
>
> BCNF:
>
> X is a superkey for R

What is the highest normal form for the above relation?

A. 1NF
B. 2NF
C. 3NF
D. BCNF

Question: Validating Normal Form

Assume that the following FDs hold for a relation R(ABCDE):

B→CD

A→E

2NF: X → A

X is not a proper subset of a candidate key for R, or

A is a prime attribute

3NF:

X is a superkey for R, or

A is a prime attribute

BCNF:

X is a superkey for R

What is the highest normal form for the above relation?

A. 1NF
B. 2NF
C. 3NF
D. BCNF

Question: Validating Normal Form

Assume that the following FDs hold for a relation R(ABCDE):

A→BCDE

E→A

X → A

2NF:
- X is not a proper subset of a candidate key for R, or
- A is a prime attribute

3NF:
- X is a superkey for R, or
- A is a prime attribute

BCNF:
- X is a superkey for R

What is the highest normal form for the above relation?

A. 1NF

B. 2NF

C. 3NF

D. BCNF

Question: BCNF and 3NF

Consider relation R(ABCD) and the following FDs:

ACD → B ; AC → D ; D → C ; AC → B

Which of the following is true:

A. R is in neither BCNF nor 3NF
B. R is in BCNF but not 3NF
C. R is in 3NF but not in BCNF
D. R is in both BCNF and 3NF

Normalization

The normalization process takes a relational schema through a series of tests to certify whether it satisfies certain conditions

- The schemas that satisfy certain conditions are said to be in a given Normal Form Unsatisfactory schemas are decomposed by breaking up their attributes into smaller relations that possess desirable properties

- Most organizations aim for designing relational databases that are in

BCNF	3NF
• Removes all anomalies ✓	• Does not removes all anomalies ✗
• Does not preserve all FDs ✗	• Preserve all FDs ✓

How can we design relational databases that are in a given Normal Form (3NF or BCNF)?

The University of Queensland Australia | CREATE CHANGE

Design Guidelines

Functional Dependencies

Normalization

Relational Database Schema Design

Algorithms for Relational Database Schema Design

Two algorithms for creating a relational decomposition from a universal relation:

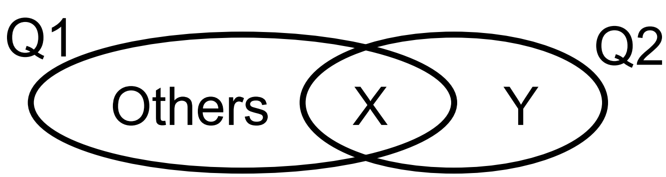
Lossless join and anomaly-free decomposition into **BCNF** schemas

Lossless join and dependency-preserving synthesis into **3NF** schemas

BCNF Decomposition

Input: a universal relation R and a set F of FDs

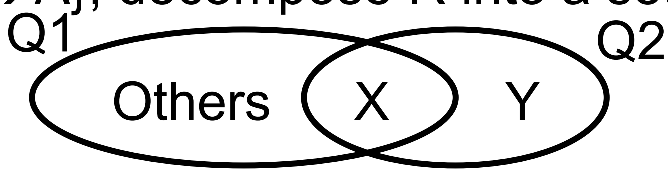
Let D := {R};
While (a relation Q in D is not in BCNF) {
  Find one FD X → Y in Q that violates BCNF;
  Replace Q in D by Q1(Q - Y) and Q2(X ∪ Y);
};



Note: answer may vary depending on order you choose. That's okay

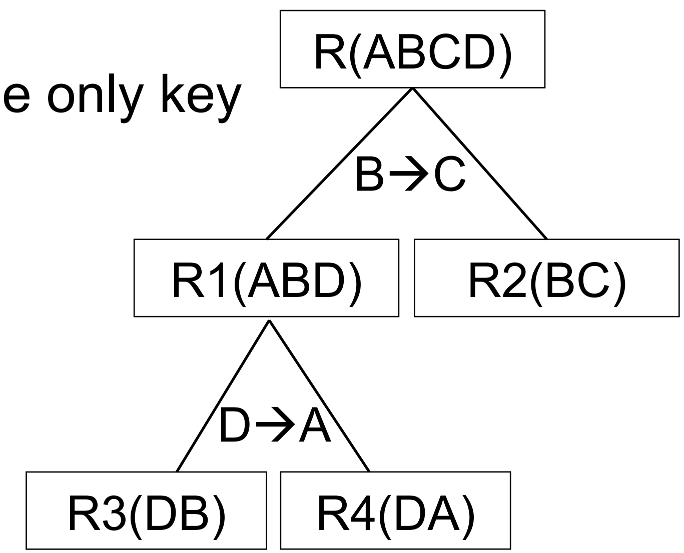
BCNF Decomposition Example

Given relation R(ABCD) and F = {B→C, D→A}, decompose R into a set of relation schemas which are in BCNF.



Find closures and keys

- $\{B\}^+$ = BC, $\{D\}^+$ = DA, $\{BD\}^+$ = BDCA is the only key



Considering B → C, is B a superkey in R?

- No. Decompose R to R1(ABD), R2(BC)

Considering D → A, it does not exist in R2. Is D a superkey for R1?

- No. Decompose R1 to R3(DB), R4(DA)

Final answer: R2(BC), R3(DB), R4(DA)

Correctness of the Algorithm

For an offending FD $X \rightarrow Y$, Q is replaced by $Q1(Q - Y)$ and $Q2(\underline{X} \cup Y)$

- $X \rightarrow Y$ no longer violates BCNF in Q1 or Q2
  - For Q1, Y does not exist anymore
  - For Q2: X is a key
- So it fixes the non-BCNF problem caused by $X \rightarrow Y$ in Q

It fixes the problems caused by all offending FDs in the end.

The algorithm terminates since all two attribute relations are in BCNF.

- R(a, b)
- No FD so no redundancy
- $a \rightarrow b$ so a is key, so in BCNF
- $b \rightarrow a$ so b is key, so in BCNF
- $a \rightarrow b$ and $b \rightarrow a$, both a and b are keys, so in BCNF

BCNF Decomposition

Input: a universal relation R and a set F of FDs

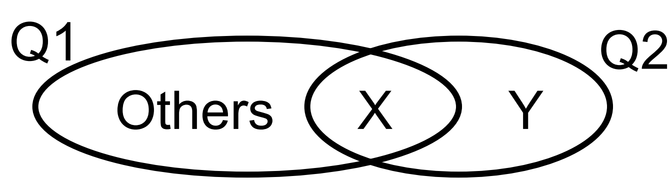
Let D := {R};

While (a relation Q in D is not in BCNF) {

    Find one FD X → Y in Q that violates BCNF;

    Replace Q in D by Q1(Q - Y) and Q2($\underline{X} \cup$ Y);

};



How do we know if FD X → Y holds in Q?

Determining Which FDs Apply

For an FD $X \rightarrow Y$, if the decomposed relation S contains $\{X \cup Y\}$, and $Y \subset X^+$ then the FD holds for S.

For example

- Consider R(ABCDE) and
- F = {AB → C, BC → D, CD → E, DE → A, and AE → B},
- project these FDs onto S(ABCD)

Does AB→D hold?

- First check if A, B and D are all in S? If not, it does not.
- Find $\{AB\}^+$ under F: ABCDE
- So yes, AB → D does hold in S

Does CD→E hold?

- First check if C, D and E are all in S? If not, it does not.
- So, no CD → E does not hold

Question: Determining Which FDs Apply

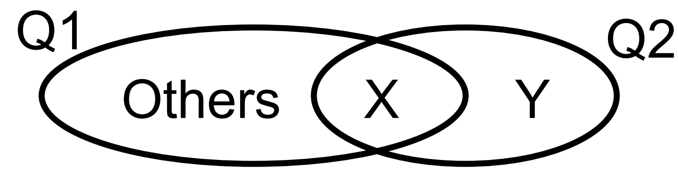
Consider relation R(ABCDE) with functional dependencies
AB → C, BC → D, CD → E, DE → A, and AE → B.
Project these FD's onto the relation S(ABCD).

Which of the following hold in S?

A. A→B
B. AB→E
C. AE→B
D. BCD→A
E. None of the above

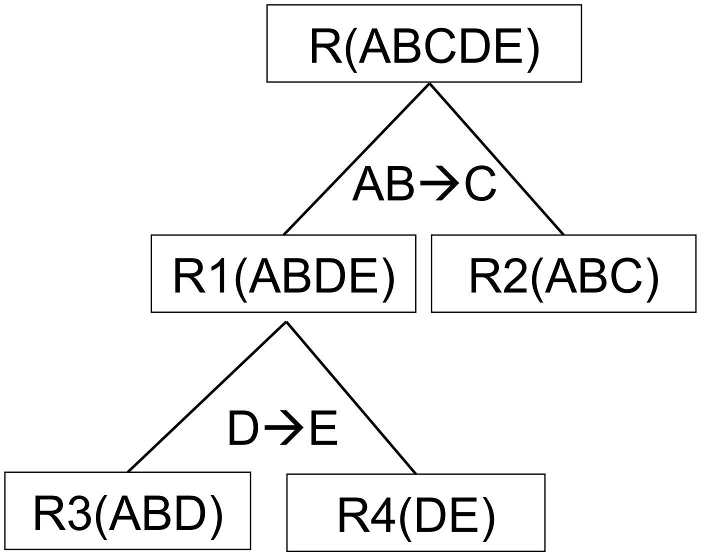
BCNF Decomposition Again

R(ABCDE) and F = {AB → C, D → E}



Find closures and keys

- $\{AB\}^{+}$ = ABC, $\{D\}^{+}$ = DE, the only key is ABD



AB → C violates BCNF in R

- R1(ABDE), R2(ABC)

D → E violates BCNF in R1

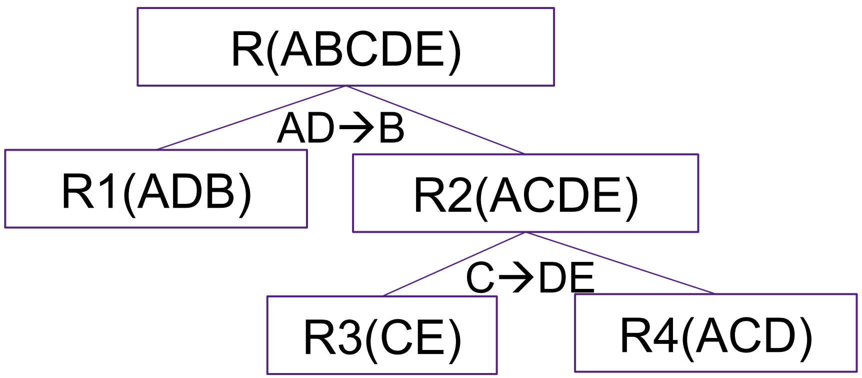
- R3(ABD), R4(DE)

Final answer: R2(ABC), R3(ABD), R4(DE)

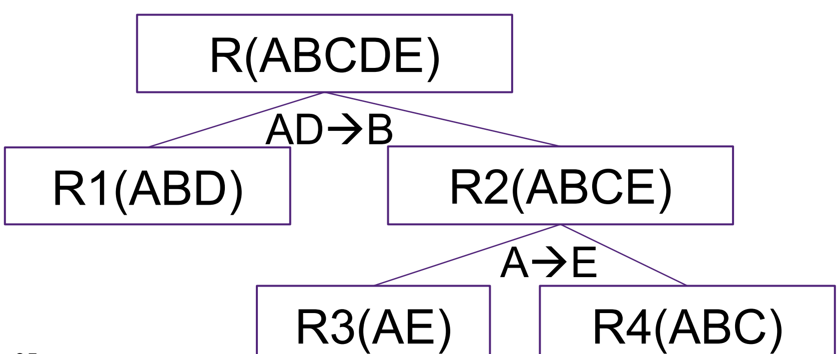
Question: BCNF Decomposition

Which of the following is a correct BCNF decomposition for R(ABCDE):
with FDs AD → B, C → DE and A → E

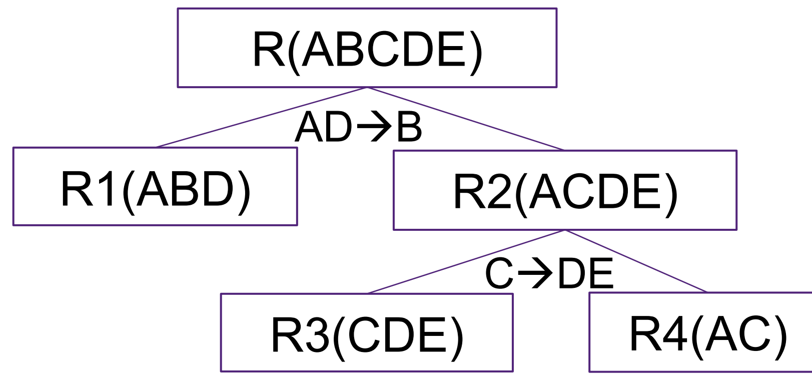
A.



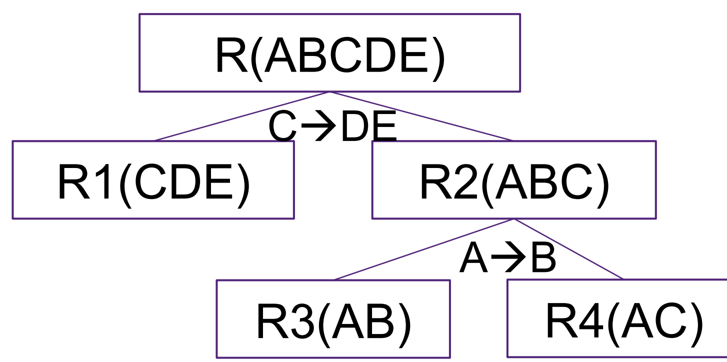
B.



C.



D.



BCNF Decomposition

Input: a universal relation R and a set F of FDs

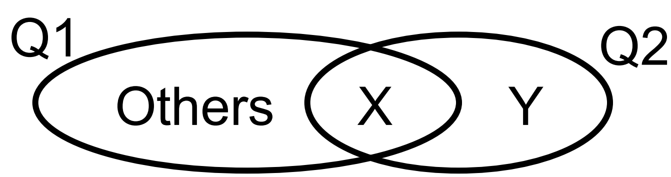
Let D := {R};

While (a relation Q in D is not in BCNF) {

Find one FD X → Y in Q that violates BCNF;

Replace Q in D by Q1(Q - Y) and Q2($\underline{X} \cup Y$);

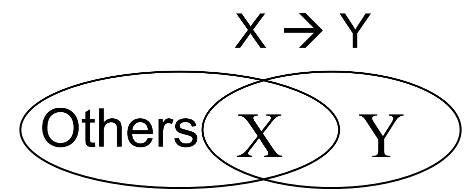
};



Note that implicit FDs should also be considered.

Example: Implicit FDs matter

R(ABCDEF), F = {A → B, DE → F, B → C}



Find closures and keys

- $\{A\}^+$ = ABC, $\{B\}^+$ = BC, $\{DE\}^+$ = DEF, the only key is ADE
- F ⊨ A → C, so we need add A → C to F

A → B violates BCNF in R

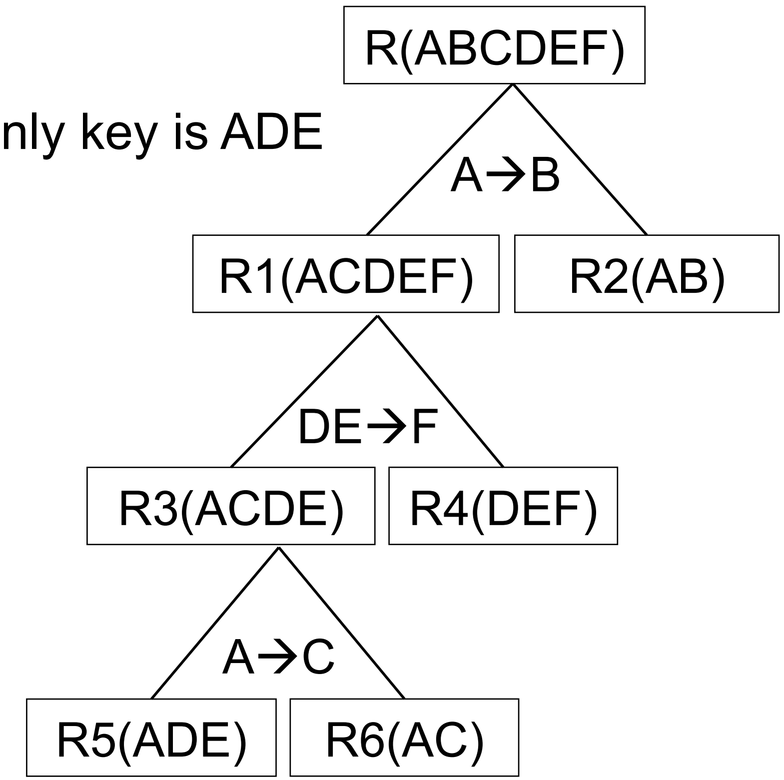
- R1(ACDEF), R2(AB)

DE → F violates BCNF in R1

- R3(ACDE), R4(DEF)

A → C violates BCNF in R3

- R5(ADE), R6(AC)



Final answer: R2(AB), R4(DEF), R5(ADE), R6(AC)

Question: BCNF Decomposition

For R(ABCD) with F = {A → B, C → D, AD → C, BC → A}, decompose it into BCNF.

Which of the following is a lossless-join decomposition of R into BCNF?

A. {AB, AC, BD}
B. {AB, AC, CD}
C. {AB, AC, BCD}
D. All of the above
E. None of the above

3NF Synthesis

```
Input: a universal relation R and a set F of FDs

    S = ∅;
    Compute the minimal cover G of F;
    Combine all FDs in G with the same LHS into one;

    For each X → Y in G {
        if (no relation in S contains X ∪ Y)
            Add a relation with schema X ∪ Y to S;
    }
    if (any candidate key is missing from the relations)
        add a relation with all prime attributes (i.e. all candidate keys);

    Eliminate redundant relations from the resulting set of relations
```

To decompose into 3NF we rely on the *minimal cover*

Minimal Cover for a Set of FDs

Goal: Transform FDs to be as small as possible

G is a minimal cover for F, iff

- $F^+ = G^+$ (i.e., they are equivalent in terms of the FDs implied)
- RHS of each FD in G is a single attribute
- If we delete any FD in G or delete any attribute from any FD in G to get G', then $G^+ \neq G'^+$

Informally: every FD in G is needed, and is "as small as possible" in order to get the same closure of F

Example:

- $F = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow BC\}$, $G = \{A \rightarrow B,\ B \rightarrow C\}$

Finding Minimal Covers of FDs

Step 1	Step 2	Step 3
RHS simplification Every FD has only one attribute on RHS	**LHS simplification** Remove any redundant attributes from the LHS of each FD	**FD set simplification** Delete any redundant FDs

Minimal Cover Example: Step 1

Step 1: RHS simplification	Step 2: LHS simplification	Step 3: FD set simplification
• Every FD has only one attribute on RHS	• Remove any redundant attributes from the LHS of each FD	• Delete any redundant FDs

Example

- R (ABCDEFGH)
- F = {A→B, ABCD→E, EF→G, EF→H, **ACDF→EG**}

Replace the last FD with

- ACDF → E
- ACDF → G

Minimal Cover Example: Step 2

Step 1: RHS simplification	**Step 2: LHS simplification**	Step 3: FD set simplification
• Every FD has only one attribute on RHS	• **Remove any redundant attributes from the LHS of each FD**	• Delete any redundant FDs

For each FD in F in the form of X → A, where X has multiple attributes including B, if $X^+ = (X - B)^+$ in F, then B can be dropped and X → A is replaced by (X - B) → A).

Example

- R (ABCDEFGH)
- F1 = {A→B, **ABCD→E**, EF→G, EF→H, ACDF→E, ACDF→G}

Can we take any attributes out from the LHS of ABCD → E?

- $\{ABCD\}^+$= ABCDE
- $\{ACD\}^+$= ABCDE, so remove B from the FD

Minimal Cover Example: Step 3

Step 1: RHS simplification	Step 2: LHS simplification	**Step 3: FD set simplification**
• Every FD has only one attribute on RHS	• Remove any redundant attributes from the LHS of each FD	• **Delete any redundant FDs**

Example

- R (ABCDEFGH)
- F2 = {A→B, ACD→E, EF→G, EF→H, **ACDF→E**, **ACDF→G**}

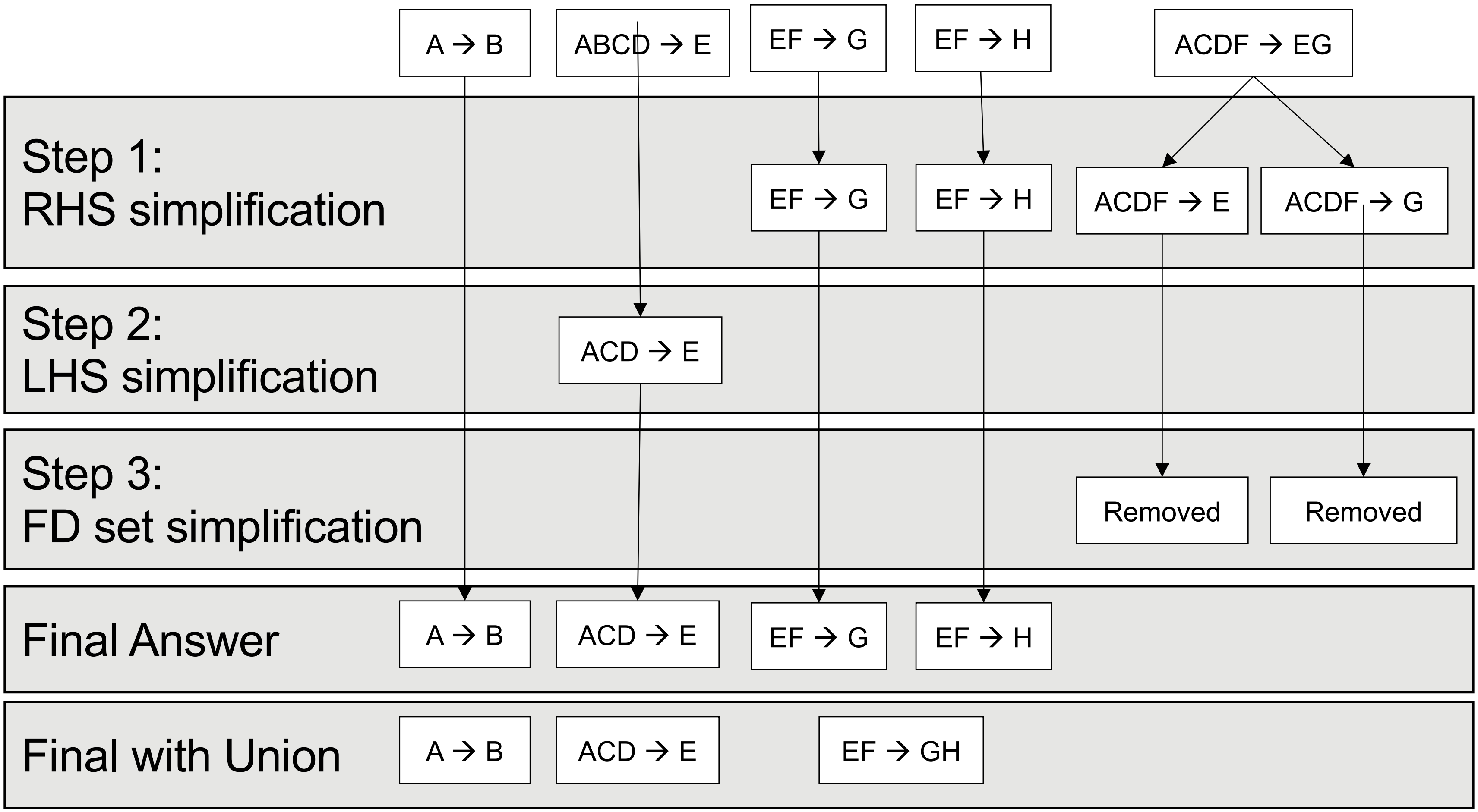
Can we drop any FD completely?

- Let's find $\{ACDF\}^+$ without considering the highlighted FDs
- $\{ACDF\}^+$= ACDFEBGH, so the highlighted rules can be removed

Final answer: A→B, ACD→E, EF→G, EF→H

Final answer after using union: A→B, ACD→E, EF→GH

Minimal Cover Example Visualised



Minimal Cover: Another Example

Consider the relation R(CSJDPQV) with FDs
F = {C→SJDPQV, JP→C, SD→P, J→S}, find a minimal cover of F

Step 1:
F = {C→S, C→J, C→D, C→P, C→Q, C→V, JP→C, SD→P, J→S}

Step 2: consider JP→C, SD→P

- Let's consider shortening JP→C
  - Not possible: JP^+ = CSJDPQV, J^+ = JS, P^+ = P

Let's consider shortening SD→P

- Not possible: SD^+ = SDP, S^+ = S and D^+ = D

Minimal Cover: Another Example

Consider the relation R(CSJDPQV) with FDs
F = {C→SJDPQV, JP→C, SD→P, J→S}, find a minimal cover of F

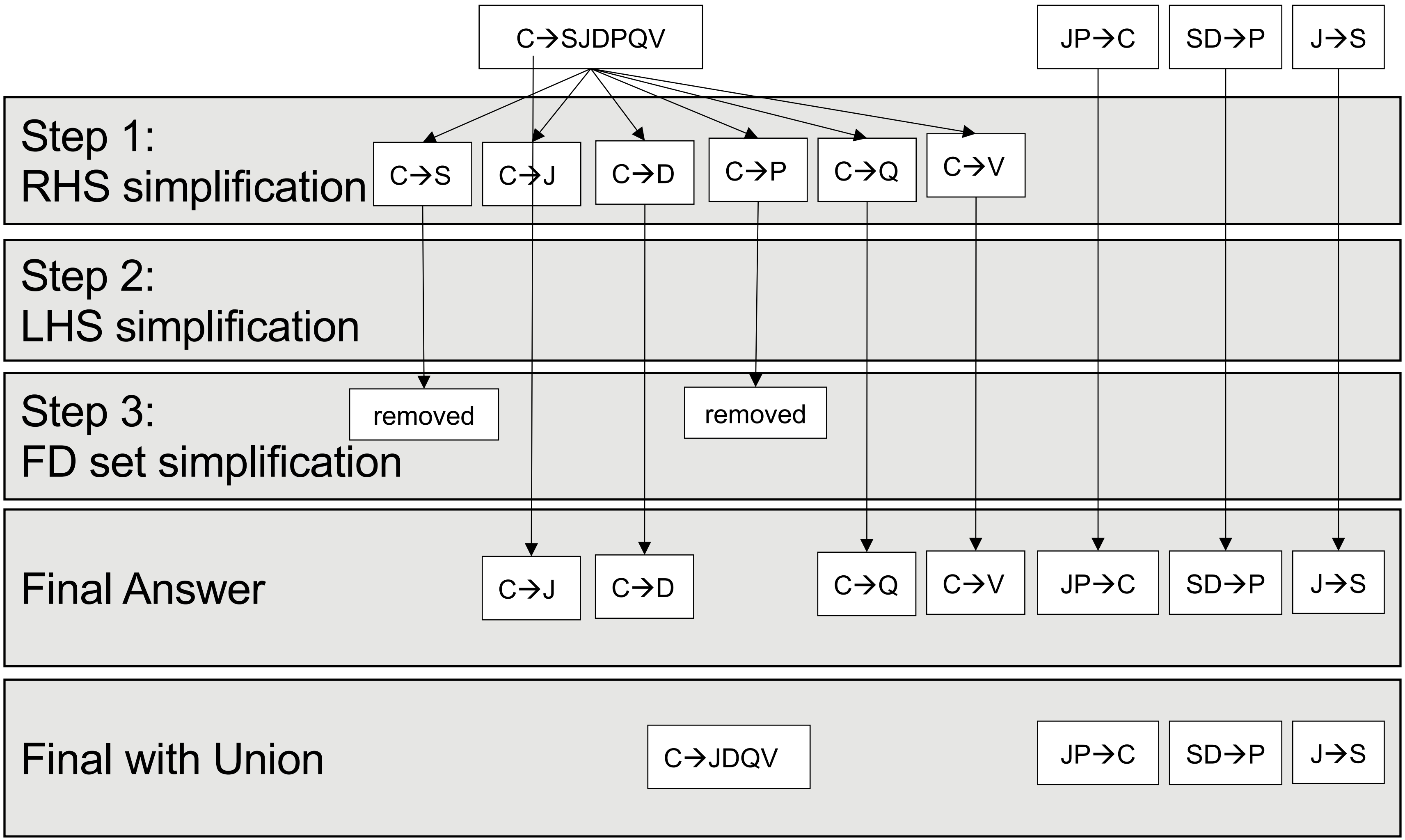
Step 3:
F1 = {C→S, C→J, C→D, C→P, C→Q, C→V, JP→C, SD→P, J→S},
can we remove any FDs?

- Let's consider C→S and find C^+ without considering this rule
  - C^+ = SJDPQV, so we can delete this FD
- Let's consider C→P and find C^+ without considering this rule
  - C^+ = SJDQVP, so we can delete this FD

So the final answer:
F2 = {C→J, C→D, C→Q, C→V, JP→C, SD→P, J→S}

F2 with union = {C→JDQV, JP→C, SD→P, J→S}

Minimal Cover Example Visualised



Question: Minimal Cover

Assume that the following FDs hold for a relation R(ABCDEF):

DEF→C

AB→DC

D→F

Which of the following is a minimal cover for the relation above?

A. {DEF → C, AB → DC, D → F}
B. {DEF → C, AB → D, D → F}
C. {DE → C, AB → D, AB → C, D → F}
D. {DE → C, AB → CD}

3NF Decomposition Revisited

Input: a universal relation R and a set F of FDs

S = ∅;
Compute the minimal cover G of F;
Combine all FDs in G with the same LHS into one;

For each $X \rightarrow Y$ in G {
    if (no relation in S contains $X \cup Y$)
        Add a relation with schema $X \cup Y$ to S;
}
if (any candidate key is missing from the relations)
    add a relation with all prime attributes;

Eliminate redundant relations from the resulting set of relations

A relation R is considered redundant if R is a projection of another relation S in the schema.

3NF Example

R(ABCDE), F = {AB → C, C → D}

- Cover already minimal
- Key: ABE

Create tables based on F = {AB → C, C → D}

- R1(ABC), R2(CD)

if (any candidate key is missing from the relations)
    add a relation with all prime attributes

- R3(ABE)

Remove redundant relations

- Nothing is redundant. Final answer is R1(ABC), R2(CD), R3(ABE)

3NF Example

R(CSJDPQV), F = {SD→P, JP→C, J→S}

- Key: JDQV
- F is already minimal

Create tables based on F = {SD→P, JP→C, J→S}

- R1(SDP), R2(JPC), R3(JS)

if (any candidate key is missing from the relations)

  add a relation with all prime attributes

- R4(JDQV)

Remove redundant relations

- Nothing is redundant.
- Final answer is R1(SDP), R2(JPC), R3(JS), R4(JDQV)

Question: 3NF Synthesis

The following is a minimal cover for a relation R(ABCDEF):

AC → E

BD → A

A → B

E → CF

Which of the following is a 3NF synthesis for R?

A. R1(ACE), R2(DBA), R3(AB), R4(ECF)
B. R1(ACE), R2(ABD), R3(AB), R4(ECF), R5(ACD)
C. R1(ACE), R2(BDA), R3(ECF), R4(ACD)
D. R1(ACE), R2(BDA), R3(ECF), R4(ACD), R5(ADE)

Top-Down vs. Bottom-Up Design

Top-Down Approach

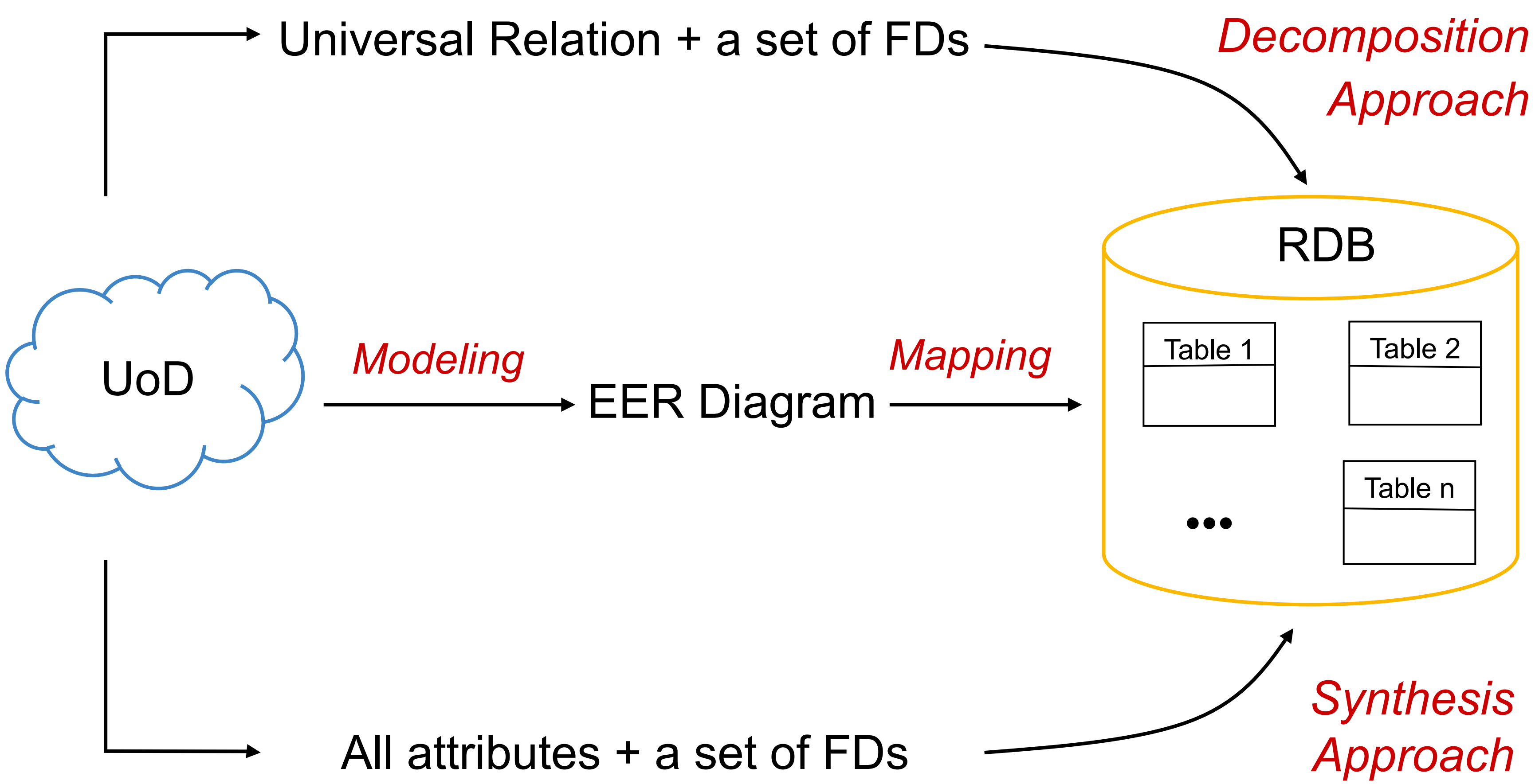
- Design by Analysis
- Start from a universal relations and a set of FDs
- Analyze and decompose into a set of BCNF relations
- Removes all anomalies ✓
- Does not preserve all FDs ✗

Bottom-Up Approach

- Design by Synthesis
- Start from individual attributes and a set of FDs
- Synthesize into a set of 3NF relations
- Does not removes all anomalies ✗
- Preserve all FDs ✓

For both approaches, the results are non-deterministic. Missing FDs can lead to sub-optimal or incorrect design

From UoD to Relational Schema



Denormalization

Given that good decomposition addresses anomalies, it is tempting to decompose the relation as much as possible

- When a table is decomposed, several relations may need to be combined to find the answers for a query

Process of intentionally violating a normal form to gain performance improvements is called denormalization

- Fewer joins (better query processing time)
- Reduces number of foreign keys (less storage and maintenance)

Useful in data analysis or if certain frequent queries require joined results

- Must be a controlled process

Summary

FDs represent data semantics

- Some FDs can be specified by database designers, others can be inferred
- FDs can be used to enforce constraints on data that should hold on all instances of a given schema

FD concepts and normalization techniques are formal theories to measure the “goodness” or quality of a relational schema design

- They can restructure (decompose) schemas, such that the resulting relations possess desirable properties and are said to be in a given normal form
- A higher NF is generally better, however other factors such as performance and decomposition need to be considered

Review

Do you know …

- How can we measure the quality of database design?
- What is a functional dependency (FD) constraint?
- What is a normal form (NF)?
- How do you achieve a (higher) NF?

Reading

- Chapters 14 (up to 14.6) and 15 (up to 15.5) in Elmasri & Navathe

Next Module

- Module 5: Database Security