COMP3311 Database Management Systems

Spring 2022

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

# RDB Design

Prof Xiaofang Zhou

# In This Module…



- We already know how to design relational databases, but
  - How can we measure the “goodness” of database design?
- Functional dependency (FD)
  - What is a functional dependency (FD) constraint?
  - How do we reason with FDs?
- Database normalization
  - What is a normal form (NF)?
  - How do you achieve a NF?

# + Informal Design Guidelines



- Informal measures of relational database schema quality and design guidelines
  1. Keeping semantics of the attributes
  2. Reducing the redundant values in tuples
  3. Reducing the null values in tuples
  4. Disallowing 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

# + Redundant Values in Tuples

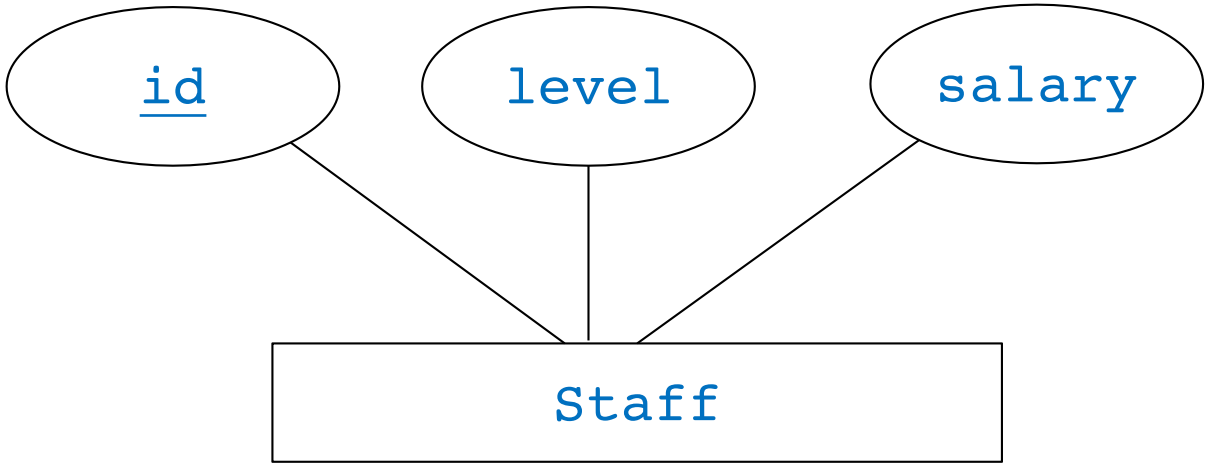


- One design goal is to minimize the storage space that base relations occupy
- The way for grouping attributes into relation schemas is done, has a significant effect on storage space
- In addition, an incorrect grouping may cause update anomalies which may result in inconsistent data or even loss of data

# + A Motivating Example



- Imagine that we have created the following entity for mailing addresses at a University




| id | level | salary |
|---|---|---|
| 100 | Lecturer | 113,728 |
| 104 | Professor | 181,964 |
| 104 | Lecturer | 113,728 |
| 105 | Associate Professor | 155,627 |

# + Update Anomalies



- Modification anomaly
  - Data inconsistency that results from data redundancy
  - E.g., Updating the salary of one lecturer (not both)
- Deletion anomaly
  - Loss of certain attributes because of the deletion of other attributes
  - E.g., Deleting 104 would lead to deletion of the salary of professor
- Insertion anomaly
  - Cannot insert some attributes without the presence of others
  - E.g., Storing the salary of senior lecturer

| id | level | salary |
|---|---|---|
| 100 | Lecturer | 113,728 |
| 104 | Professor | 181,964 |
| 104 | Lecturer | 113,728 |
| 105 | Associate Professor | 155,627 |

# + 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.
- Example:

```
Staff[id, level, salary]

StaffAppointment[id, level]
StaffIncome[level, salary]
```

- How should we decompose tables to remove anomalies?

# The Join Operation



- Definition: $R_1 \bowtie R_2$ is the (natural) join of the two relations
  - Each tuple of $R_1$ is concatenated with every tuple in $R_2$ having the same values on the common attributes

$R_1$

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |
| 7 | 2 |

$R_2$

| B | C |
|---|---|
| 2 | 3 |
| 5 | 6 |
| 2 | 8 |

$R_1 \bowtie R_2$

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 2 | 8 |
| 4 | 5 | 6 |
| 7 | 2 | 3 |
| 7 | 2 | 8 |

# + Lossless Join Decomposition



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

$$R = R_1 \bowtie R_2$$

We will discuss F and r satisfying F later

- 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 | 8 |

Decompose

$R_1$

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |
| 7 | 2 |

$R_2$

| B | C |
|---|---|
| 2 | 3 |
| 5 | 6 |
| 2 | 8 |

Join

R’

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 2 | 8 |
| 4 | 5 | 6 |
| 7 | 2 | 3 |
| 7 | 2 | 8 |

- 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?

# + Guideline 4

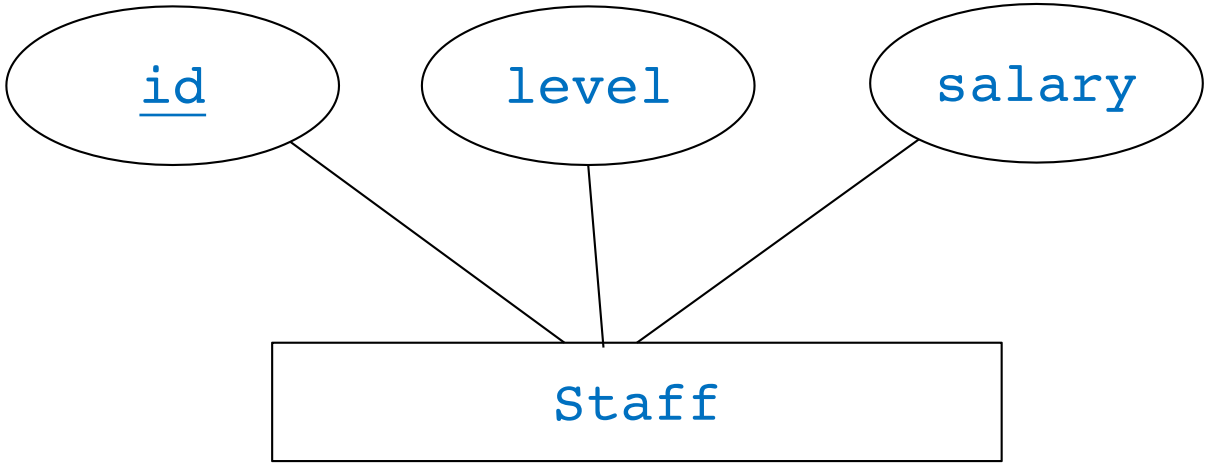


- Design the relation schemas so that they can be (relationally) joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated

# + Functional Dependency



- How do I know for sure if all staff 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`. But `level` is NOT a key.



# + Functional Dependency, Formally



- A functional dependency (FD) X→Y holds on relation R if for every legal instance of 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`
- 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

...X determines Y, X implies Y, or Y can be derived from X

# + 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 | level | salary |
|---|---|---|
| 100 | Lecturer | 113,728 |
| 104 | Professor | 181,964 |
| 104 | Lecturer | 113,728 |
| 105 | Associate Professor | 155,627 |

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

# + Question:



- 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 |

- What FDs cannot be true given the instance above?
  1. B→ C
  2. B→ D
  3. D→ B
  4. All of the above can be true
  5. None of the above can be true

*1 and 3 are fine, 2 cannot be true*

# + Fixing Anomalies



| id | level | salary |
|---|---|---|
| 100 | Lecturer | 113,728 |
| 104 | Professor | 181,964 |
| 104 | Lecturer | 113,728 |
| 105 | Associate Professor | 155,627 |

| id | level |
|---|---|
| 100 | Lecturer |
| 104 | Professor |
| 104 | Lecturer |
| 105 | Associate Professor |

| level | salary |
|---|---|
| Lecturer | 113,728 |
| Associate Professor | 155,627 |
| Professor | 181,964 |

# + Anomalies Fixed?



| id | level |
|---|---|
| 100 | Lecturer |
| 104 | Professor |
| 104 | Lecturer |
| 105 | Associate Professor |

| level | salary |
|---|---|
| Lecturer | 113,728 |
| Associate Professor | 155,627 |
| Professor | 181,964 |

- Modification anomaly fixed: Updating the salary only in one place does not cause inconsistencies
- Deletion anomaly fixed: Deleting the row with id 104 does not delete the salary of Professor
- Insertion anomaly fixed: It is now possible to store the salary for a Senior Lecturer without a staff appointed at this level

# General Anomaly Fixing

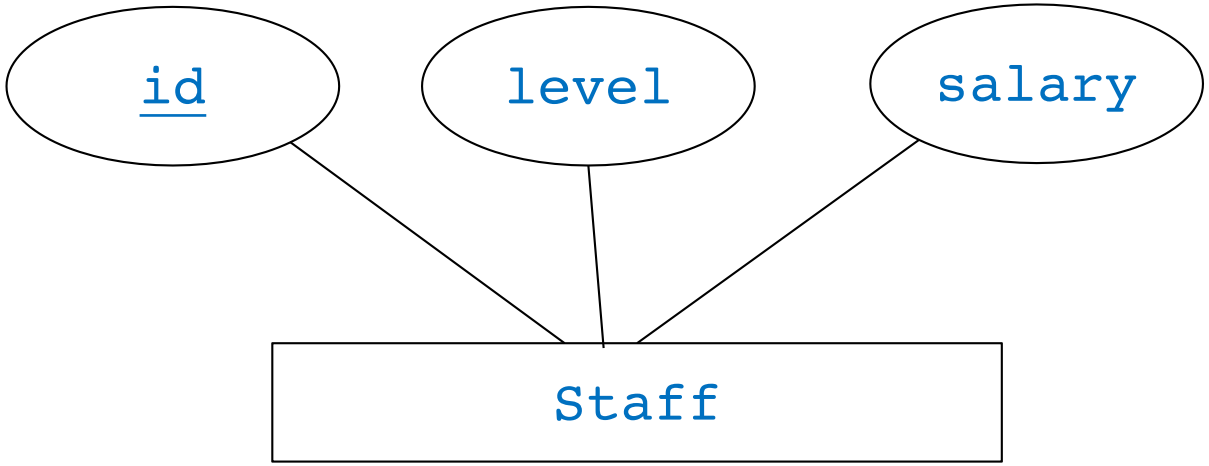


- We were able to fix the anomalies by splitting the `Staff` table into two tables
- In the rest of this module, we will discuss how anomalies can be addressed formally

# + Keys



- As a reminder, 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



# + Question: Possible Keys



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

B→C

C→B

D→ABC

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

1. B
2. C
3. BD
4. All of the above
5. None of the above

*5 is the answer, as B and C cannot determine all attributes, and BD is not minimum*

# Question: Possible Superkeys



Assume the same relation R(A,B,C,D) and FDs

B→C

C→B

D→ABC

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

1. D
2. BD
3. BCD
4. All are superkeys
5. None are superkeys

*4, as D is a key already*

# + Explicit and Implicit FDs



- Given a set of (explicit) functional dependencies, we can determine implicit ones
  - Explicit FDs: `staffID` → `level`, `level` → `salary`
  - Implicit FD: `staffID` → `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., a 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 $X \rightarrow Y$
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- 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
  - Union: if X → Y, X → Z then X → YZ
  - Pseudotransitivity: if X → Y, WY → Z then WX → Z

# The Six Inference Rules



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

- IR4 Decomposition: $X \rightarrow YZ \models X \rightarrow Y$
- IR5 Union: $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$
- IR6 Pseudo transitivity: $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

# + Proof of Decomposition Rule



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

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

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 → Y, X → Z} ⊨ X → YZ

IR1 Reflexivity: Y ⊆ X ⊨ X → Y
IR2 Augmentation: X → Y, ⊨ XZ → YZ
IR3 Transitivity: {X → Y, Y → Z} ⊨ X → Y

Proof:

1: X → Y: Given

2: X → Z: Given

3: XX → XY: Using IR2 on 1 (adding X)

4: X → XY: Since XX=X (not using IRI-R3)

5: XY → YZ: Using IR2 on 2 (adding Y)

6: X → YZ: Using IR3 on 4 & 5

# + Proof of Pseudotransitive Rule



To prove IR6: {X → Y, WY → Z} ⊨ WX → Z

IR1 Reflexivity: Y ⊆ X ⊨ X → Y
IR2 Augmentation: X → Y, ⊨ XZ → YZ
IR3 Transitivity: {X → Y, Y → Z} ⊨ X → Y

Proof

1: X → Y: Given

2: WY → Z: Given

3: WX → WY: Using IR2 on 1 (adding W)

4: WX → Z: Using IR3 on 3 & 2

# + $F^+$ and $X^+$



- $F^+$ is all FDs which can be derived from F
  - Too many and too time-consuming to compute
  - And not necessary
- $X^+$ is the closure of a set of attributes X under F
  - Computing $X^+$ is easy

```
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 $X^+$ Computation



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

X = {pNumber}

F = {pNumber → {pName, pLocation, dNum},

dNum → {dName, mgrSSN, mgrStartDate}}

Step 1: $X^+$ := {pNumber}

Step 2: $X^+$ := {pNumber, pName, pLocation, dNum}

Step 3: $X^+$ := {pNumber, pName, pLocation, dNum, dName, mgrSSN, mgrStartDate}

# Question: $X^+$ Computation



Given R(ABCDE) and F = {A→B, BC→E, ED→A}

Determine $(ACD)^+$

*ACD, ACDB, ACDBE (stop now, as all attributes are included already)*

# + Superkey Test



Step 1: Database designers first specify the set of FDs F that can easily be determined from semantics

Step 2: For a given set of attributes, it is a superkey iff the closure includes all attributes in the relation

# + Example: Supplier-Part DB



- Suppliers supply parts to projects
  - `SupplierPart(sName,city,status,pNum,pPame,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`
- Exercise
  - Show that (`sName`, `pNum`) is a superkey
  - Show that this pair of attributes is a key

# + 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 $S'^+ = R$

Notes:

- If R has n attributes, there exist $2^n$ subsets
- 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

..set, subset/superset, proper subset
…can you design an algorithm to find all keys?

# + Question: Finding Key



Which of the following is a key of Relation R(ABCDE) with F = {D → C, CE → A, D → A, AE → D}

1. ABDE
2. BCE
3. CDE
4. All of these are keys
5. None of these are keys

*1 is a superkey as D -> C,*
*2 is a key*
*3 is not a key, as its closure does not include B*

# + 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)

# + Review of "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!
  - An attribute that is not a member of any candidate key is a non-prime attribute

# First Normal Form (1NF)



- A relation schema is in 1NF if domains of 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

# Second Normal Form (2NF)



- R is a relation schema, with the set F of FDs
- R is in 2NF iff
  - For all X: $X \subset R$
  - and, for all $A \in R$
  - such that there exists a FD: $X \rightarrow A$ in $F^+$
- Then
  1. $A \in X$ (i.e., $X \rightarrow A$ is trivial), or
  2. X is NOT a proper subset of a candidate key for R, or
  3. A is a prime attribute

In other words, a relation is in 2NF iff for any non-trivial FD $X \rightarrow A$ where A is a non-primary attribute, X is not a proper subset of any key

# + A non-2NF Example



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

F = {`houseNum`, `street`, `postCode` → `state`, `value`,
`postCode` → `state`}

Consider X → A where X = {`houseNum`, `street`, `postcode`} and A = `state`

1) `state` is not a primary attribute, and
2) `postCode` → `state` is not a trivial FD, and
3) `postCode` is a proper subset of X, which is a candidate key

It is a problem because `state` does not directly depend on a key in entirety. It depends on a key partially (i.e., `postCode`).

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

# + Boyce-Codd Normal Form (BCNF)



- R is a relation schema, with the set F of FDs
- R is in BCNF iff
  - For all X: $X \subset R$
  - and, for all $A \in R$
  - such that there exists a FD: $X \rightarrow A$ in $F^+$
- Then
  - $A \in X$ ($X \rightarrow A$ is trivial), or
  - X is a superkey for R

In other words, a relation is in BCNF iff for any non-trivial FD $X \rightarrow A$, X must be a superkey

…informally: whenever a set of attributes of R determine another attribute, it should determine all the attributes of R

# + A Non-BCNF Example



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

Is R in BCNF?

- Candidate keys: AD and BD
- B $\rightarrow$ A is a non-trivial FD, B is not a superkey
- So, B $\rightarrow$ A violates BCNF

Note: A is a primary attribute

# + Another BCNF Example



`Address(houseNum, street, city, state, postCode)`

F = {{`houseNum`, `street`, `postcode`} → {`city`, `state`},

`postCode` → {`city`, `state`}}

Is `Address` in BCNF?

- Only one key: {`houseNum`, `street`, `postCode`}
- {`postCode`}$^+$ = {`postCode`, `city`, `state`}, but `postCode` is not a superkey
- So, `postCode` → `city` violates BCNF

Note: `city` is not a primary attribute

# + Third Normal Form (3NF)



- R is a relation schema, with the set F of FDs
- R is in 3NF if and only if
  - For all X: $X \subset R$
  - and, for all $A \in R$
  - such that there exists a FD: $X \rightarrow A$ in F+
- Then
  - $A \in X$ ($X \rightarrow A$ is trivial), or
  - X is a superkey for R, or
  - 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-primary attribute, X must be a superkey

… both BCNF & 3NF can be said informally to have no transitive dependency

# + BCNF ⊂ 3NF ⊂ 2NF



- BCNF:
  - Trivial, or
  - X is a superkey for R
- 3NF:
  - Trivial, or
  - X is a superkey for R, or
  - A is a primary attribute
- 2NF:
  - Trivial, or
  - X is not a proper subset of a candidate key for R, or
  - A is a primary attribute

For a relation R with FD set F,
we check for all possible $X \rightarrow A$ in $F^+$

…2NF: every things depends on a key and entire key
…3NF/BCNF: everything depends on a key and nothing else

# An Example of 2NF But Not 3NF



- If it is not in 2NF, it cannot be in BCNF/3NF

`Address(houseNum, street, postcode, state, value)`

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

postCode → state}

- If it is in 2NF, it is not necessarily in 3NF

`Address(houseNum,street,postcode,state,value,premier)`
F = {{houseNum, street, postcode} → {state, value, premier},

state → premier}

# An Example of 3NF But Not BCNF



- If it is not in 3NF, it cannot be in BCNF
  - 3NF considers only primary attributes, BCNF only considers any attributes (thus more strict)
- A 3NF is not necessarily in BCNF

`Study`(`studentID`, `courseID`, `lecturer`)

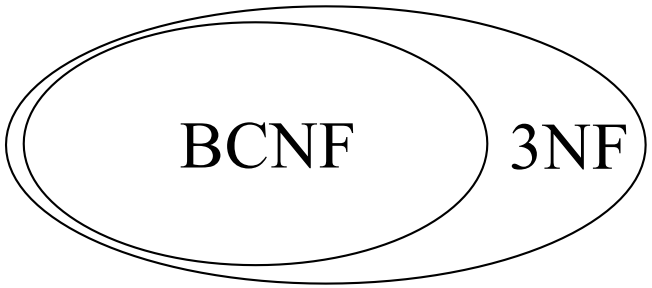
F = {{`studentID`, `courseID`} → `lecturer`,
`Lecturer` → `courseID`}

This relation is in 3NF as `courseID` is a primary attribute (so we don't check it for 3NF). However, for BCNF, we need check all attributes including `coursID`. In this case `lecturer` is not a superkey

# BCNF & 3NF



- BCNF guarantees removal of all anomalies
- 3NF has some anomalies, but preserves all dependencies
- If a relation R is in BCNF it is in 3NF
- A 3NF relation R may not be in BCNF
- Most organizations go to BCNF or 3NF



# + 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

*Any key must have A, and $A^+ = A$*
*So we need to check AB, AC and AD*
*$AB^+ = AB$, $AC^+ = ACDB$, $AD^+ = ADCB$*
*Now we only need to check by expanding AB*
*ABC, ABD: no need to check*

*So keys are: AC, AD*

*D → C (D not key so not BCNF)*
*C is part of a candidate key*
*so R is in 3NF*

# The Good News…



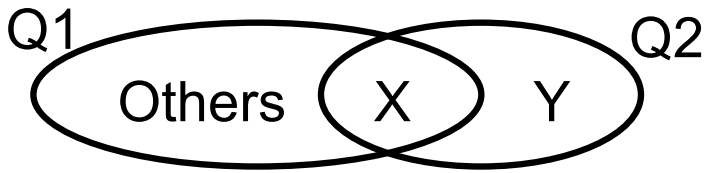
- The normalization process sounds complicated?
  - When you design a database, we can simply check for BCNF or 3NF
- This is quite simple
  - To check for BCNF, just take each attribute and see if any subset of the attributes that can determine it is a super key
  - To check for 3NF follows the same process, but can skip all the primary attributes
- The hard part for both cases is that we need to computer closures for all attribute subsets
  - This step finds all keys too
  - You need to be systematic about generating all subsets
  - This is easy for a computer program, and not too hard for us to do if the number of attributes in a relation is small

# + BCNF Decomposition



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

Let D := R;
While (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);
};




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

# + Why is This Algorithm Correct?



- For an offending FD X→Y, Q is replaced by Q1(Q-Y) and Q2(X ∪ Y)
  - X→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→Y in Q
- It fixes the problems caused by all offending FDs in the end

… X is the join attributes between Q1 and Q2

# + BCNF Decomposition Example



Given relation R(ABCD) and F = {B→C, D→A}

- Closure and keys
  - $BD^+$ = 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 R2 to R3(DB), R4(DA)
- Final answer: R2(BC), R3(DB), R4(DA)

# + Determining Which FDs Apply



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?
  - No

# + Question:



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

- Which of the following hold in S?
  1. A→B
  2. AB→E
  3. AE→B
  4. BCD→A
  5. None of the above

*1) A+ = {A}, so not hold*
*2) E is not in S, no hold*
*3) As above*
*4) {BCD}+ = {ABCDE} (Note: we use all FDs to find the closure, including DC -> E, even E is not in S)*

# BCNF Decomposition Again



R(ABCDE) and F = {AB → C, D → E}

- Find closure of the following
  - 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)

# + 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)

# + Exercise: More BCNF



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?

1. {AB, AC, BD}
2. {AB, AC, CD}
3. {AB, AC, BCD}
4. All of the above
5. None of the above

# + Excise: Do the Work



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

- Find the keys
  - There are 3 keys: $\{AD\}^+$ = ADBC, $\{AC\}^+$ = ACBD, $\{BC\}^+$ = BCAD
- A → B violates BCNF
  - R1(ACD), R2 (AB)
- C → D violates BCNF for R1
  - R3(AC), R4(CD)
- So the final answer is {AB, AC, CD}

# + Excise: Check the Answers



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?

1. {AB, AC, BD}
2. {AB, AC, CD}
3. {AB, AC, BCD}
4. All of the above
5. None of the above

# + Exercise: Option ‘A’ Exposed



R(ABCD), F = {A → B, C → D, AD → C, BC → A}

Is {AB, AC, BD} a lossless join?

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

→

| A | B |
|---|---|
| **1** | **2** |
| 8 | 2 |

| A | C |
|---|---|
| 1 | 5 |
| **1** | **3** |
| 8 | 9 |

| B | D |
|---|---|
| 2 | 6 |
| 2 | 7 |
| **2** | **4** |

→

| A | B | C | D |
|---|---|---|---|
| 1 | 2 | 5 | 6 |
| 1 | 2 | 3 | 7 |
| 8 | 2 | 9 | 4 |
| **1** | **2** | **3** | **4** |
| … | … | … | … |

… please check the join results based on the correct decomposition

# 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 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 $F_1$, $F_2$, … $F_n$ accordingly
  - $F_i$ contains all FDs in F with the attributes completely in $R_i$
- If $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$, then this decomposition of R is dependency preserving
- 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

# An Illustrative BCNF Example

| studentID | courseID | lecturer |
|---|---|---|
| | | |
| | | |

{studentID, courseID} → lecturer
lecturer → courseID

Is lecturer a key?

| studentID | lecturer |
|---|---|
| | |
| | |

| lecturer | courseID |
|---|---|
| | |
| | |

The first FD is no longer preserved!

# So What's the Problem?



| lecturer | courseID |
|---|---|
| Xiaofang | INFS1200 |
| Shazia | INFS1200 |

| studentID | lecturer |
|---|---|
| James | Xiaofang |
| James | Shazia |

`lecturer` → `courseID`

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

| studentID | courseID | lecturer |
|---|---|---|
| James | INFS1200 | Xiaofang |
| James | INFS1200 | Shazia |

Violates the FD: `studentID`, `courseID` → `lecturer`

# + 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 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→B, B→C, A→BC}, G = {A→B, B→C}

… a similar concept is called canonical cover which allows to have more than one attribute on the right-hand side

# + Finding Minimal Covers of FDs



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

… in this course our introduction to minimum cover-based approach is very brief

# + Minimum Cover Example: Step 1



{A→B, ABCD→E, EF→G, EF→H, ACDF → EG}

- Replace last rule with
  - ACDF → E
  - ACDF → G

# + Minimum Cover Example: Step 2



{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

… 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 (i.e., X →A is replaced by (X-B) →A)

…alternatively, you can simply check if X-B → B in order to drop B

# + Minimum Cover Example: Step 3



{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

… this is a very simple example to illustrate ideas

# + 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}

# + Synthesis into 3NF



Give R and F, to design a database in 3NF

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

for each $X \rightarrow Y$ in G {
    if (no relation in S contains $X \cup Y$)
        a relation with schema $X \cup Y$ to S;
}

if (no relation in S contains a candidate key for R)
    add a relation with any candidate key as the schema;

# + Synthesis Example



- R(ABCDE), F = {AB → C, C → D}
  - Key: ABE, derived FD: AB → D
- Decomposition from C → D
  - R1(ABCE), R2(CD)
  - R3(ABE), R4(ABC), R2(CD)
- Decomposition from AB → C
  - R1(ABDE), R2(ABC)
  - R2(ABE), R3(ABD), R2(ABC) [Note: C → D not preserved]
- Synthesis
  - F is already minimal
  - R1(ABC), R2(CD), R3(ABE) [Note: same as design 1]

# + Another Synthesis Example



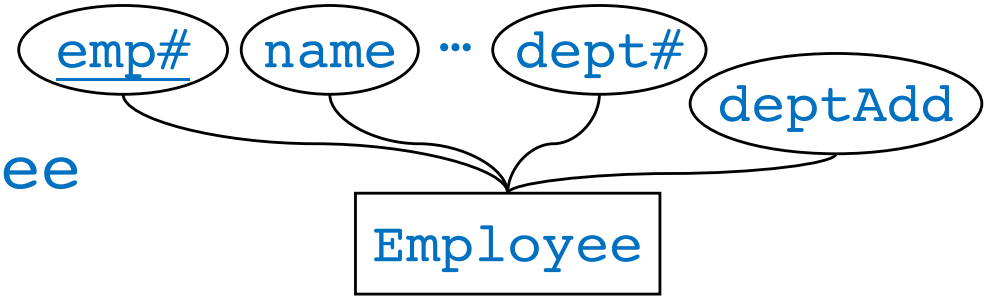
R(CSJDPQV), F = {SD→P, JP→C, J→S}

- Key: JDQV, derived FD: JD → C (as $JD^+$ = JDSPC)
- Decomposition 1
  - R1(CSJDQV), R2(SDP) [removing SD→P]
  - R3(CJDQV), R4(JS), R2(SDP) [removing J→S]
  - R5(JDQV), R6(JDC), R4(JS), R2(SDP) [removing JD→C]
- Decomposition 2
  - R1(SJDPQV), R2(JPC) [removing JP→C]
  - R3(JDPQV), R4(JS), R2(JPC) [removing J→S]
  - R5(JDQV), R6(SDP), R4(JS), R2(JPC) [removing SD→P]
- Synthesis
  - F is already minimal
  - R1(SDP), R2(JPC), R3(JS), R4(JDQV) [same as decomposition 2]

# + Normalization And The ER Model



- When an E-R diagram is well designed, the relation schemas generated from it should not need further normalization
- However, in an imperfect design there can be FDs from non-key attributes of an entity to some other attributes of the entity
- For example, an `Employee` entity with an FD
  `dept#` → `deptAdd`
- A good design would have made `Department` a separate entity



# + 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 that require joined results
  - Must be a controlled process

# + 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

(covered in this course)

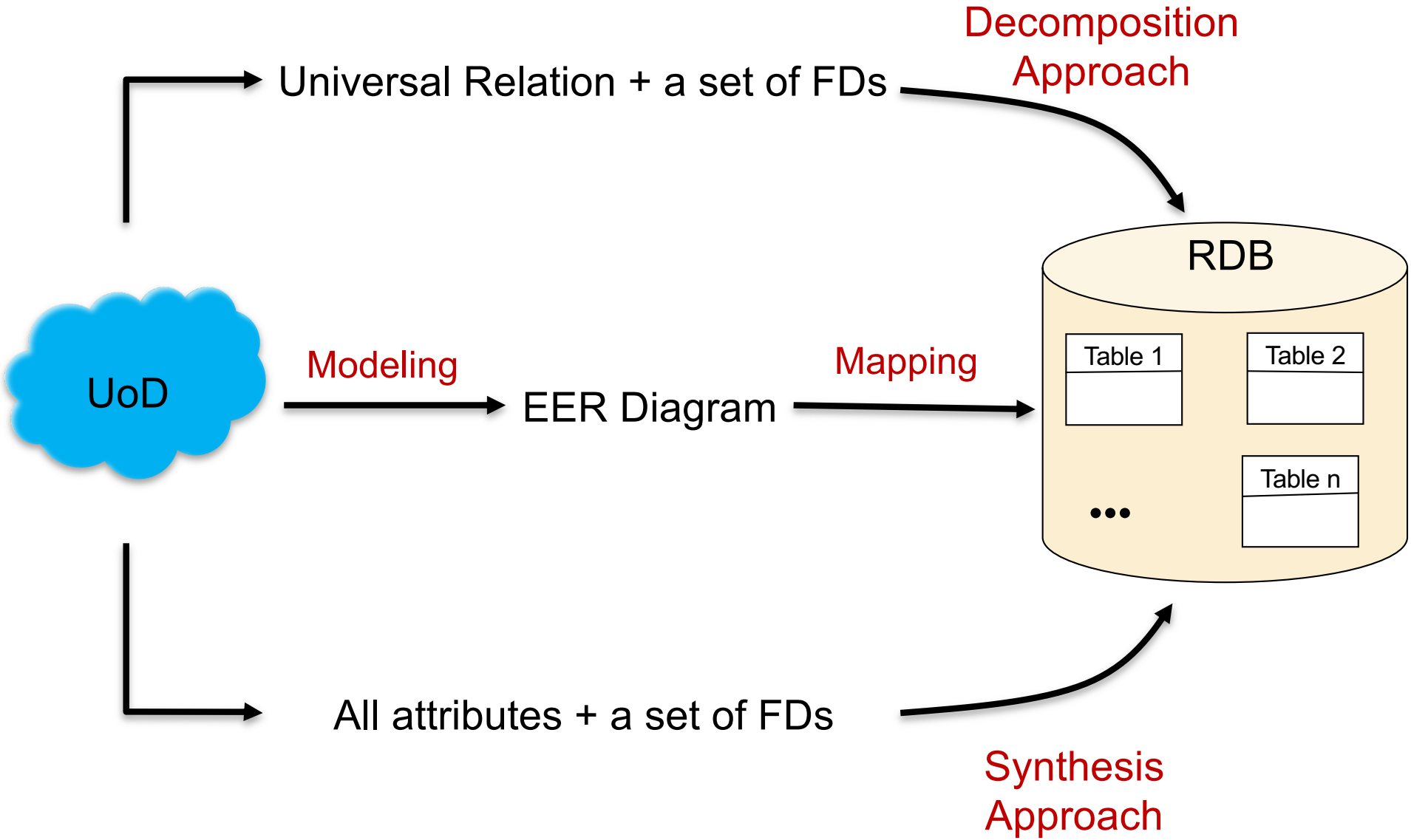
**Bottom-Up Approach**

- Design by Synthesis
- Start from individual attributes and a set of FDs
- Synthesize into a set of 3NF relations

(briefly illustrated in this course)

… for both approaches, the results are non-deterministic
… missing FDs can lead to sub-optimal or incorrect design

# From UoD to Relational Schema



# + Summary



- FD 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

# + Readings



- Textbook:  Ch 8 (ed 6)/Ch 7 (ed 7)