8. Relational Database Design

University Relational Schema

- Classroom (building, room number, capacity)
- Department (dept name, building, budget)
- Course (course_id, title, dept name, credits)
- Instructor (i_ID, name, dept name, salary)
- Section (course id, sec id, semester, year, building, room number, time slot id)
- Teaches (i_ID, course id, sec id, semester, year)
- Student (s_ID, name, dept name, tot cred)
- Takes (ID, course id, sec id, semester, year, grade)
- Advisor (s_ID, i_ID)
- time slot (time slot id, day, start time, end time)
- Prereq (course id, prereq id)

Goal

- To generate a set of relation schemas, that allows us to
  - Store information without unnecessary redundancy, yet
  - Also allows us to retrieve information easily
- To design schemas in an appropriate normal form
  - Need information of real-world enterprise information
  - Some of this information exists in a well-designed E-R diagram, but additional information about the enterprise may be needed as well
  - Require formal approach to relational database design based on the notion of functional dependencies to define normal forms in terms of functional dependencies and other types of data dependencies

Logical Database Design

- The process of deciding how to arrange the attributes of the entities in the business environment into database structures, such as the tables of a relational database
  - This consists of deciding which tables to create, what columns they will contain, as well as the relationships between the tables
- The goal is to create well structured tables that properly reflect the company's business environment

Design Alternative: : Larger Schemas

- Suppose instead of having two schemas instructor and department, have single schema: inst dept (ID, name, salary, dept name, building, budget)
  - Result of a natural join on the relations corresponding to instructor and department
  - Seems a good idea because some queries can be expressed using fewer joins
    - Need to repeat the department information ("building" and "budget") once for each instructor in the department, as many instructors are available in the departments – creating problem of redundancy
    - If we forget the redundancy problem, and still continue with design there is still another problem
      - When creating a new department in the university
      - We cannot represent directly the information concerning a department unless that department has at least one instructor for newly created department
      - In the old design, the schema department can handle this

Design Alternative: : Smaller Schemas

- A real-world database has
  - A large number of schemas, an even larger number of attributes and the number of tuples can be in the millions or higher
- Real university requires that every department must have only one building and one budget value, so two schemas

  Department (dept name, building, budget)

  Instructor (i_ID, name, dept name, salary)

Design Alternative: : Smaller Schemas

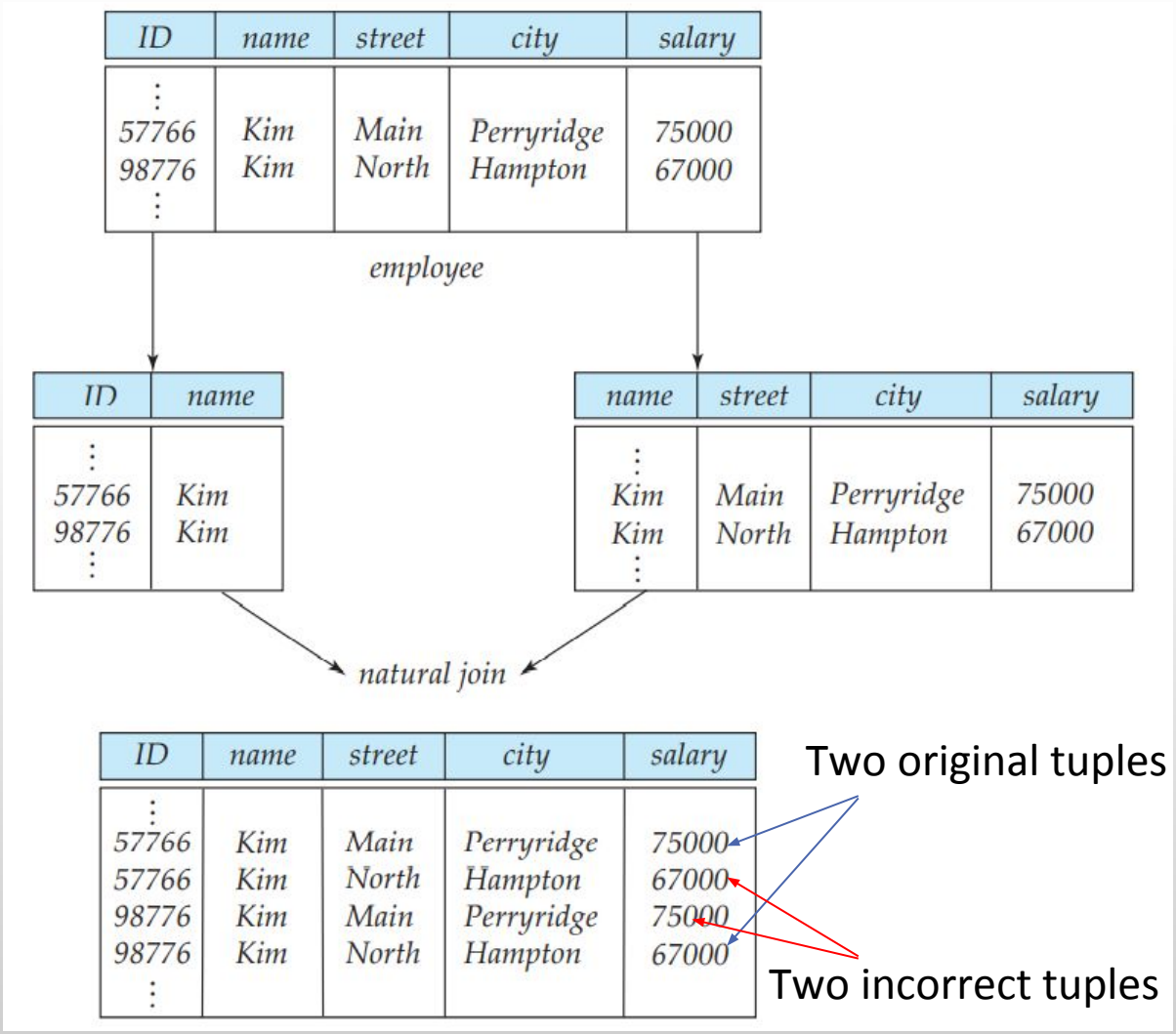
- Not all decompositions of schemas are helpful
- Consider an extreme case where all schemas consisting of one attribute
  - No interesting relationships of any kind could be expressed
- Consider a less extreme case
  - Decompose employee schema: employee (ID, name, street, city, salary)
  - Into two schemas:

    employee1 (ID, name) employee2 (name, street, city, salary)

Design Alternative: : Smaller Schemas

- If we attempt to regenerate the original tuples using a natural join
  - Although we have more tuples, we actually have less information
  - Unable to distinguish which of the Kims'
  - Decomposition is unable to represent certain important facts about the university employees
    - Need to avoid such decompositions
    - Such decompositions known as **lossy decompositions**
  - Opposite to this are known as lossless decompositions

Two employees with the same name 'Kim'



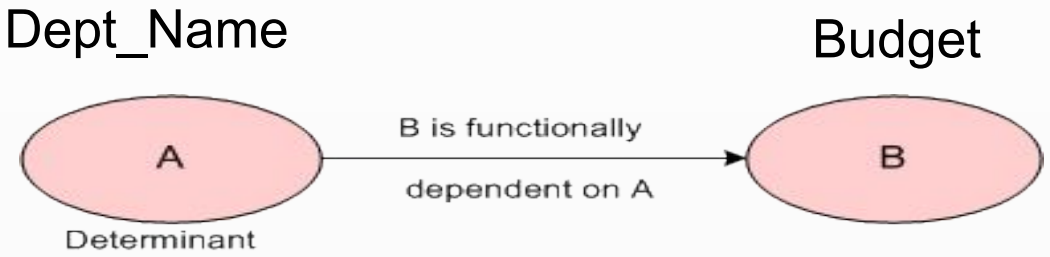
Functional Dependency

- Real university requires that every department must have only one building and one budget value, so two schemas

  Department (dept name, building, budget), Instructor (i_ID, name, dept name, salary)

- Need to specify rules such as “each specific value for dept_name corresponds to at most one budget” even in cases where dept_name is not the primary key for the schema in question
- A rule that says “if there were a schema (dept name, budget), then dept name is able to serve as the primary key”, specified as a functional dependency: **dept_name → budget**

Functional Dependency



- The value of Dept_Name *determines* the value of Budget
- Dept_Name is the **determinant**
- Budget is **functionally dependent** on Dept_Name

Functional Dependencies

- Rules help to recognize situations where a schema ought to be split, or decomposed, into two or more schemas
- Finding the right decomposition is much harder for schemas with a large number of attributes and several functional dependencies
- To deal with this, required to study a formal methodology

Keys and Functional Dependencies

- A database models a set of entities and relationships in the real world
- But, there are variety of constraints (rules) on the data also
  - For example, some of the constraints for a university database are:
  1. Students and instructors are uniquely identified by their ID
  2. Each student and instructor has only one name
  3. Each instructor and student is (primarily) associated with only one department
  4. Each department has only one value for its budget, and only one associated building

Keys and Functional Dependencies

- Legal instance of the relation
  - An instance of a relation that satisfies all such real-world constraints
- Legal instance of a database
  - An instance of a database where all the relation instances are legal instances
- Some of the most commonly used types of real-world constraints can be represented formally as Keys (super key, candidate key and primary key), or as functional dependencies

Design Guidelines for Relational Schemas

- Informal measures of quality for relation schema design
  - Semantics of the attributes
  - Reducing the redundant values in tuples
  - Disallowing the possibility of generating spurious tuples
- These measures are not always independent of one another

Design Guidelines for Relational Schemas

- Semantics of the attributes
  - How the attribute values in a tuple relate to one another
  - Do not combine attributes from multiple entity types and relationship types into a single relation
    - A relation schema corresponds to one entity type or one relation

Design Guidelines for Relational Schemas

- Reducing the redundant values in tuples
  - Redundant information in more than one place within a database, creating problems:
    - **Redundant Storage: Some information is stored repeatedly**
    - **Update Anomalies: If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated**
    - **Insertion Anomalies: It may not be possible to store certain information unless some other, unrelated, information is stored as well**
    - **Deletion Anomalies: It may not be possible to delete certain information without losing some other, unrelated, information as well**
  - Design the base relation schemas by replacing the schema so that no insertion, deletion, or modification anomalies are present in the relations
  - If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly

Design Guidelines for Relational Schemas

- Disallowing the possibility of generating spurious tuples
  - Design relation schemas so that they can be 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
  - Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations, because joining on such attributes may produce spurious tuples

The Data Normalization Process

- A methodology for organizing attributes into tables so that redundancy among the non-key attributes is eliminated

- **The output of the data normalization process is a properly structured relational database**

Data Normalization Technique

- Input:
  - All the attributes that must be incorporated into the database
  - A list of all the defining associations between the attributes (i.e., the **functional dependencies**)
    - A means of expressing that the value of one particular attribute is associated with a single, specific value of another attribute
    - If we know the value of A and we examine the relation that holds this dependency, we will find only one value of B in all of the tuples that have a given value of A, at any moment in time
    - However, that for a given value of B there may be several different values of A

Functional Dependencies (FDs)

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

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold
- Reason to identify FDs that hold for all possible values for attributes of a relation
  - These represent the types of integrity constraints that we need to identify
  - Such constraints indicate the limitations on the values that a relation can legitimately assume
  - In other words, they identify the legal instances which are possible

Functional Dependency - Check

Given the following relation instance.

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the instance?

(a) X -> Y (b) Y -> Z (c) X -> Z (d) Y -> X (e) Z -> X (f) Z -> Y

In option (a), its given X->Y, it means that the value of X uniquely determines the value of Y. But, here the value 1 of X, gives three different values of Y i.e. 4, 5 and 6. Therefore this FD is not satisfied by the instance.

(a) X -> Y (b) Y -> Z **(c) X -> Z** (d) Y -> X **(e) Z -> X** **(f) Z -> Y**

Functional Dependency - Check

Given the following relation instance.

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the instance?

(a) XY -> Z and Z -> Y (b) YZ -> X and X -> Z

(c) YZ -> X and Y -> Z (d) XZ -> Y and Y -> X

Functional Dependency - Check

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

- Option (a): XY -> Z and Z -> Y
  - Given Z->Y, it means that the value of Z uniquely determines the value of Y. But here the value 2 of Z, gives two different values of Y i.e. 4 and 2. Therefore this FD is not satisfied by the instance.
- Option (b): YZ -> X and X -> Z
  - Given X->Z, it means that the value of X uniquely determines the value of Z. But here the value 1 of X, gives two different values of Z i.e. 2 and 3. Therefore this FD is not satisfied by the instance.

Functional Dependency - Check

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

- Option (c): YZ -> X and Y -> Z
  - Given Y-> Z, here the value of Y uniquely determines the value of Z. Therefore this FD is satisfied by the instance.
  - Now take FD YZ-> X, here (4,2), (5,3), (6,3), (2,2) uniquely determine the value of X. Therefore this FD (YZ->X) is satisfied by the instance.

- Option (d): XZ -> Y and Y -> X
  - Given Y->X, here the value of Y uniquely determines the value of X. Therefore this FD is satisfied by the instance.
  - Now take FD XZ->Y, here (1,3) cannot uniquely determine the value of Y. (1,3) gives two values for Y i.e. 5 and 6. Therefore this FD (XZ->Y) is not satisfied by the instance.

Functional Dependency

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

- Possible FDs
  - STUD_NO is unique for each student
    - STUD_NO □ STUD_NAME
    - STUD_NO □ STUD_PHONE
    - STUD_NO □ STUD_STATE
    - STUD_NO □ STUD_COUNTRY
    - STUD_NO □ STUD_AGE
    - STUD_STATE □ STUD_COUNTRY

Check FD: STUD_STATE □ STUD_COUNTRY ?

Yes, possible FD, as if two records have same STUD_STATE, they will have same STUD_COUNTRY as well

Check FD: STUD_NAME □ STUD_ADDR ?

No, Not possible, as two students' having same name, determines two different States.

Functional Dependency

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

- FDs set: Set of all FDs present in the relation

{STUD_NO → STUD_NAME,
STUD_NO → STUD_PHONE
STUD_NO → STUD_STATE
STUD_NO → STUD_COUNTRY
STUD_NO → STUD_AGE
STUD_STATE → STUD_COUNTRY
}

Functional Dependency

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

- FD Set: ?
  {
  C→B
  AB→C
  AB→D
  CD→B
  }

- Check FDs?
  - A→B
    (Violating Tuples: 1 and 2)
  - B→A
    (Violating Tuples: 2 and 3)
  - D→C
    (Violating Tuples: 3 and 4)

Functional Dependencies (FDs)

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example*:*
    - *ID, name → ID*
    - *name → name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
  - Although trivial Fds are valid, they offer no additional information about integrity constraints for the relation
  - As far as normalization is concerned, trivial Fds are ignored

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F* which can be inferred or deduced from the Fds in F
- For example: If we know that Kristi is older than Debi and that Debi is older than Traci, we are able to infer that Kristi is older than Traci.
  - How did we make this inference?
  - Without thinking about it or maybe knowing about it, we utilized a transitivity rule to make this inference: Kristi > Debi, Debi > Traci, then Kristi > Traci

Closure of a Set of Functional Dependencies

- For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*
- We denote the *closure* of *F* by $\mathbf{F^+}$
- F^+ is a superset of *F*

Closure of a Set of Functional Dependencies

- To infer {*A → C}* from {{*A → B*}, {*B → C}}*
- *Require a* set of inference rules to infer the set of Fds in F+
- Armstrong provided a set of inference rules

Closure of a Set of Functional Dependencies

- We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - **Main** (sufficient set of inference rules for generating closure set of FDs)
    - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(Reflexivity)**
    - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(Augmentation)**
    - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(Transitivity)**
  - **Others** (Can be derived from main)
    - if $\alpha \rightarrow \gamma\ \beta$, then $\alpha \rightarrow \gamma$ and $\alpha \rightarrow \beta$ **(Projection)**
    - if $\alpha \rightarrow \gamma$ *and* $\alpha \rightarrow \beta$, then $\alpha \rightarrow \gamma\ \beta$ **(Additive)**
    - if $\alpha \rightarrow \beta$, and $\beta\ \gamma \rightarrow I$, then $\alpha\gamma \rightarrow I$ **(Pseudo Transitivity)**
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold)

Closure of a Set of Functional Dependencies

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Some members of F^+ : $\{A \rightarrow H, AG \rightarrow I\}$?
  - $A \rightarrow H$
    - Achieved by:
      1) transitivity from $A \rightarrow B$
      2) transitivity from $B \rightarrow H$

Closure of a Set of Functional Dependencies

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Some members of F^+ : $\{A \rightarrow H, AG \rightarrow I\}$?
  - $AG \rightarrow I$
    - Achieved by
      1) augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      2) transitivity with $CG \rightarrow I$

Closure of a Set of Functional Dependencies

- Given R = (A,B,C,D,E,F,G,H, I, J) and
- F = {AB → E, AG → J, BE → I, E → G, GI → H}
- Does F+ : {AB → GH} ?

Closure of a Set of FDs

if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(Reflexivity)**
if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$ **(Augmentation)**
if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(Transitivity)**
if $\alpha \rightarrow \gamma\,\beta$, then $\alpha \rightarrow \gamma$ and $\alpha \rightarrow \beta$ **(Projection)**
if $\alpha \rightarrow \gamma$ *and* $\alpha \rightarrow \beta$, then $\alpha \rightarrow \gamma\,\beta$ **(Additive)**

- Given R = (A,B,C,D,E,F,G,H, I, J) and F = {AB → E, AG → J, BE → I, E → G, GI → H}, Does F+: {AB → GH}?
- Proof

1. AB → E, given in F
2. AB → AB, reflexive rule IR1
3. AB → B, projective rule IR4 from step 2
4. AB → BE, additive rule IR5 from steps 1 and 3
5. BE → I, given in F
6. AB → I, transitive rule IR3 from steps 4 and 5
7. E → G, given in F
8. AB → G, transitive rule IR3 from steps 1 and 7
9. AB → GI, additive rule IR5 from steps 6 and 8
10. GI → H, given in F
11. AB → H, transitive rule IR3 from steps 9 and 10
12. AB → GH, additive rule IR5 from steps 8 and 11

– proven

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
    then add the resulting functional dependency to F^+
until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Hardware Company Database

- Salesperson is working at an office. Salesperson sells the products to the customers.

Office Number	Telephone	Size
OFFICE		

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number
SALESPERSON				

Product Number	Product Name	Unit Price
PRODUCT		

Salesperson Number	Product Number	Quantity
SALES		

Customer Number	Customer Name	Salesperson Number	HQ City
CUSTOMER			

Hardware Company: SALESPERSON and PRODUCT

Salesperson Number
Salesperson Name
Commission Percentage
Year of Hire
Department Number
Manager Name
Product Number
Product Name
Unit Price
Quantity

- List out the functional dependencies.

Hardware Company: SALESPERSON and PRODUCT

Salesperson Number
Salesperson Name
Commission Percentage
Year of Hire
Department Number
Manager Name
Product Number
Product Name
Unit Price
Quantity

Salesperson Number → Salesperson Name
Salesperson Number → Commission Percentage
Salesperson Number → Year of Hire
Salesperson Number → Department Number
Salesperson Number → Manager Name
Product Number → Product Name
Product Number → Unit Price
Department Number → Manager Name
Salesperson Number, Product Number → Quantity

Functional Dependency - Advantages

- Removes data redundancy where the same values should not be repeated at multiple locations in the same database table
- Maintains the quality of data in the database
- Allows clearly defined meanings and constraints of databases
- Helps in identifying bad designs of the database
- Expresses the facts about the database design

Attributes Closure

- An attribute B is functionally determined by α, if α → B
- To test whether a set α is a super key
  - Devise an algorithm for computing the set of attributes functionally determined by α
    - Using FDs
      - One way of doing this is to compute F^+, take all functional dependencies with as the left-hand side, and take the union of the right-hand sides of all such dependencies
    - Expensive, since F^+ can be large
  - Solution
    - Attributes Set Closure

Uses of Attributes Set Closure

- Testing for super key:
  - To test if α is a super key, compute α^+, and check if α^+ contains all attributes of *R*
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$
    - Compute α^+ by using attribute closure, and then check if it contains β
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, find the closure γ^+, and for each $S \subseteq \gamma^+$, output a functional dependency $\gamma \rightarrow S$

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*
- Algorithm to compute α^+, the closure of α under *F*

```
        result := α;
    while (changes to result) do
        for each β → γ in F do
            begin
                if β ⊆ result then  result := result ∪ γ
            end
```

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, \quad A \rightarrow C, \quad CG \rightarrow H, \quad CG \rightarrow I, \quad B \rightarrow H\}$
- $(AG)^+ = ?$

1. *result = AG*

2. *result = ABCG* (*$A \rightarrow B$ and* A $\subseteq$ AG, so result U= B,

   $A \rightarrow C$ and A $\subseteq$ AG, so result U= C*)*

3. *result = ABCGH* (*$CG \rightarrow H$* and *$CG \subseteq AGBC$, so result U= H)*

4. *result = ABCGHI* (*$CG \rightarrow I$* and *$CG \subseteq AGBCH$, so result U= I)*

- No new attributes are added to *result after this, and the algorithm terminates*

Uses of the Attribute Closure

- To test if α is a superkey, we compute α+, and check if α+ contains all attributes in *R*
- We can check if a functional dependency α → β holds (or, in other words, is in *F+), by checking if* β ⊆ α*+. That is, we compute* α*+ by using attribute* closure, and then check if it contains β
- It gives us an alternative way to compute *F+: For each* γ ⊆ *R, we find the closuer* γ+ and for each *S* ⊆ γ *+, we output a functional dependency* γ → *S.*

Attribute Set Closure

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

- FDs set:{

  STUD_NO → STUD_NAME,
  STUD_NO → STUD_PHONE
  STUD_NO → STUD_STATE
  STUD_NO → STUD_COUNTRY
  STUD_NO → STUD_AGE
  STUD_STATE → STUD_COUNTRY
  }

Attribute Closure:
(STUD_NO)+ = {
STUD_NO,
STUD_NAME,
STUD_PHONE,
STUD_STATE,
STUD_COUNTRY,
STUD_AGE
}
(STUD_STATE)+ = {STUD_STATE,
STUD_COUNTRY}

Find Candidate Keys and Super Keys using Attribute Closure

- If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation
- If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well
- For Example, using FD set of STUDENT table ,
- (STUD_NO, STUD_NAME)+ =

  {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}
- (STUD_NO)+ =

  {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}
- (STUD_NO, STUD_NAME) will be super key but not candidate key because its subset (STUD_NO)+ is equal to all attributes of the relation
- So, STUD_NO will be a candidate key

Example of Attribute Set Closure

- Let R= (A, B, C, D, E, F) be a relation Relation with the following dependencies: C->F, E->A, EC->D, A->B. Which of the following is a key for R?

(a) CD (b) EC

(c) AE (d) AC

FDs: C->F, E->A, EC->D, A->B
Find closure set for CD.
X = CD
= CDF {C->F}

No more attributes can be added to X. Hence closure set of CD = CDF

Find closure set for EC.
X = EC
= ECF {C->F}
= ECFA {E->A}
= ECFAD {EC->D}
= ECFADB {A->B}

Closure set of EC covers all the attributes of the relation R.
If any closure covers all the attributes of the relation R then that is the key.

Attribute Closure

- Given the relation Relation R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R.
- What is the key for R?: i) {E, F}, ii) {E, F, H}, iii) {E, F, H, K, L}, iv) {E}

Attribute Closure

- Given the relation Relation R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R.
- What is the key for R?: i) {E, F}, ii) {E, F, H}, iii) {E, F, H, K, L}, iv) {E}
- Now, find the attribute closure of all given options, receive:
  {E,F}+ = {EFGIJ}
  {E,F,H}+ = {EFHGIJKLMN}
  {E,F,H,K,L}+ = {{EFHGIJKLMN}
  {E}+ = {E}
  {EFH}+ and {EFHKL}+ results in set of all attributes, but EFH is minimal. So it will be candidate key. So correct option is (ii).

Canonical Cover

- Whenever a user updates the database, the system must check whether any of the functional dependencies are getting violated in this process
  - If there is a violation of dependencies in the new database state, the system must roll back
  - Working with a huge set of functional dependencies can cause unnecessary added computational time
  - This is where the canonical cover comes into play
- A canonical cover of a set of functional dependencies F is a simplified set of functional dependencies that has the same closure as the original set F

Canonical Cover

- A Canonical Cover for a set of functional dependencies F is another set of functional dependencies Fc such that all the functional dependencies in F logically imply all the functional dependencies in Fc and vice versa
- Fc should meet the following requirements;
  1. F should logically imply all FDs in Fc, [F = Fc]
  2. Fc should logically imply all FDs in F,
  3. Functional dependencies of Fc should not contain any **Extraneous attribute**.
  4. The left side of all the functional dependencies in Fc should be unique.

Canonical Cover

- So, a Canonical cover Fc is a ***minimal*** set of FDs that is equivalent to F, and ***have no redundant FDs or redundant attributes as part of FDs***
- In other words, every functional dependency of Fc is very much needed and it is as small as possible when compared to the size of F

Canonical Cover

Example:

- **Redundant Functional Dependency:**
- A → C is redundant in {A → B, B → C, A → C}
- Here, A → B and B → C will automatically include A → C as a result of Transitivity
- Hence, we do not need to check whether C is uniquely determined by A or not [in other words, A uniquely determines C or not]. **Hence, A → C is redundant**
- And, the set of functional dependencies {A → B, B → C} is semantically equivalent to given set of functional dependencies {A → B, B → C, A → C}

Canonical Cover Algorithm

ALGORITHM CanonicalCover (FD set F)

BEGIN

REPEAT UNTIL STABLE

1. Wherever possible, apply UNION rule from Armstrong's Axioms

(e.g., A → BC, A → CD becomes A → BCD)

2. Remove "extraneous attributes", if any, from every FD

(e.g., AB→C, A→B becomes A→B, B→C i.e., A is extraneous in AB→C)

END

Canonical Cover

- For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \quad B \rightarrow C, A \rightarrow C\}$
- Parts of a functional dependency may be redundant
  - E.g.: on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to

    $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
  - E.g.: on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to

    $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$

Extraneous Attributes

- **Redundant Attributes or Redundant Part of Set of Attributes:**
- ***Attribute C is redundant on the Right Hand Side (RHS) of FD***

$$A \rightarrow CD \text{ in } \{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$$

  - Here, C is already determined by B
  - Hence, we do not need to include in another FD to check the dependency
  - So, the given set of functional dependencies can be simplified as

$$\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$$

  - And, this is equivalent

Extraneous Attributes

- **Redundant Attributes or Redundant Part of Set of Attributes:**
- ***Attribute C is redundant on the Left Hand Side (LHS) of FD***

$$AC \rightarrow D \text{ in } \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$$

- Here, if we know A, intuitively we know C as well through Transitivity rule
- Hence, A → D is suffice to represent. So, the given set of functional dependencies can be simplified as

$$\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$$

- And, this is equivalent to the given set of FDs

Extraneous Attributes - Algorithm

- ***Case 1 (LHS)***: To find if an attribute A in α is extraneous or not. That is, to test if an attribute of Left Hand Side of a functional dependency is Extraneous or not.
- **Step 1**: Find $(\{\alpha\} - A)^+$ using the dependencies of F.
- **Step 2**: If $(\{\alpha\} - A)^+$ contains all the attributes of β, then A is extraneous.

- ***Case 2 (RHS)***: To find if an attribute A in β is extraneous or not. That is, to test if an attribute of Right Hand Side of a functional dependency is Extraneous or not.
- **Step 1**: Find α^+ using the dependencies in F' where $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{ \alpha \rightarrow (\beta - A) \}$.
- **Step 2**: If α^+ contains A, then A is extraneous.

Finding Extraneous Attributes

- **<u>Example 1 for LHS:</u>**
- Given F = {P→Q, PQ→R}. Is Q extraneous in PQ→R?
- Looking for a LHS attribute, Hence, let us use Case 1 discussed above

Step 1: Find $(\{\alpha\} - A)^+$ using the dependencies of F.

- Here, α is PQ. So find $(PQ - Q)^+$, i.e., P+ (closure of P).
- From F, if you know P, then you know Q (from P→Q).
- If you know both P and Q then you know R (from PQ→R).
- Hence, the closure of P is PQR.

Step 2: If $(\{\alpha\} - A)^+$ contains all the attributes of β, then A is extraneous.

- $(PQ - Q)^+$ contains R. Hence, Q is extraneous in PQ→R.

Finding Extraneous Attributes

- **Example 2 for RHS:**
- Given F = {P→QR, Q→R}. **Is R extraneous in P→QR?**
- Looking for a RHS attribute. Hence, let us use the Case 2 given above.

Step 1: Find α^+ using the dependencies in F' where F' = (F – {α → β}) U { α → (β – A) }.

- Let us find F' as stated above.
- F' = (F – {α → β}) U { α → (β – A) } = ({P→QR, Q→R} – {P→QR}) U {P→(QR-R)}
- = ({Q→ R} U {P→Q})
- F' = {Q→R, P→Q}
- Here, α is P. So find $(P)^+$, i.e., closure of P using the F' which we found.
- From F', if you know P, then you know Q (from P→Q).
- If you know Q then you know R (from Q→R).
- Hence, the closure of P is PQR.

Step 2: If α^+ contains A, then A is extraneous.

- P+ contains R. Hence, R is extraneous in P→QR.

Extraneous Attributes

- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$?
  - $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$?
  - Since AB $\rightarrow C$ can be inferred even after deleting C

Computing a Canonical Cover

- $R = (A, B, C)$ *and* $F = \{A \rightarrow BC, \quad B \rightarrow C, \quad A \rightarrow B, \quad AB \rightarrow C\}$

Canonical Cover Algorithm - Revise

ALGORITHM CanonicalCover (FD set F)

BEGIN

REPEAT UNTIL STABLE

1. Wherever possible, apply UNION rule from Armstrong's Axioms

(e.g., A → BC, A → CD becomes A → BCD)

2. Remove "extraneous attributes", if any, from every FD

(e.g., AB→C, A→B becomes A→B, B→C i.e., A is extraneous in AB→C)

END

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

Computing a Canonical Cover

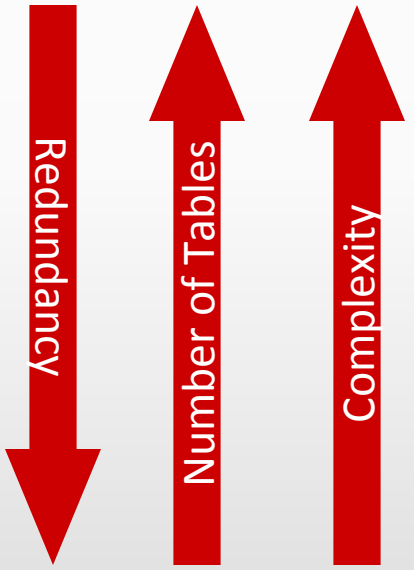
- *R = (A, B, C) and F = {A → BC, B → C, A → B, AB → C}*
- Combine *A → BC* and *A → B* into *A → BC*
  - Set is now *{A → BC, B → C, AB → C}*
- *A* is extraneous in *AB → C*
  - Check if the result of deleting A from *AB → C* is implied by the other dependencies
    - Yes: in fact, *B → C* is already present!
  - Set is now *{A → BC, B → C}*
- *C* is extraneous in *A → BC*
  - Check if *A → C* is logically implied by *A → B* and the other dependencies
    - Yes*:* using transitivity on *A → B and B →* C.
      - Can use attribute closure of *A* in more complex cases
- The canonical cover is: *A → B, B → C*

Normalization

- The process which allows you to remove redundant data within your database.
- The applications required to maintain the database are simpler.
- This involves restructuring the tables to successively meeting higher forms of Normalization.
- A design that has a lower normal form than another design has more redundancy. Uncontrolled redundancy can lead to data integrity problems.
- A properly normalized database should have the following characteristics
  - Scalar values in each fields.
  - Absence of redundancy.
  - Minimal use of null values.
  - Minimal loss of information.

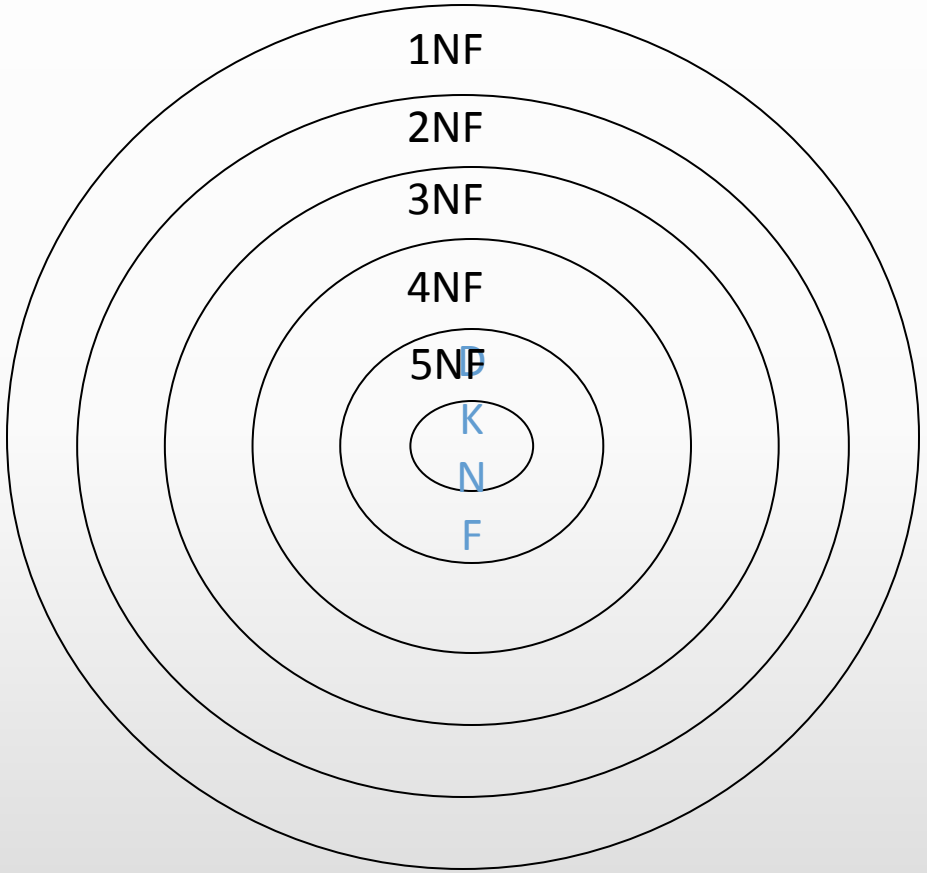
Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.

Levels of Normalization



Each higher level is a subset of the lower level

The data normalization process is progressive.

For example, if a group of tables is in second normal form, it is also in first normal form.

Normalization

- With the exception of 1NF, all these normal forms are based on Functional dependencies among the attributes of a table.
- A group of tables is said to be in a particular normal form if every table in the group is in that normal form.

First Normal Form

- A relation is in **1NF**, if all values stored in the relation are single-valued and atomic (as opposed to list of values).

- 1NF places restrictions on the structure of relations. Values must be simple.

  - All the fields contain only scalar values
    - Having scalar values also means that all instances of a record type must contain the same number of fields.

First Normal Form (1NF)

Example: 1 NF?

ISBN	Title	AuName	AuPhone	PubName	PubPhone	Price
0-321-32132-1	Balloon	Sleepy, Snoopy, Grumpy	321-321-1111, 232-234-1234, 665-235-6532	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Jones, Smith	123-333-3333, 654-223-3455	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Joyce	666-666-6666	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Roman	444-444-4444	Big House	123-456-7890	$25.00

Author and AuPhone columns are not scalar

Not 1NF

A table not in first normal form is called un normalized

1NF - Decomposition

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced (the primary key of the original table concatenated with one or more data items from the new table)
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

Example 1: Converted to 1NF

Primary Key

ISBN	**Title**	**PubName**	**PubPhone**	**Price**
0-321-32132-1	Balloon	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Big House	123-456-7890	$25.00

Primary Key

ISBN	**AuName**	**AuPhone**
0-321-32132-1	Sleepy	321-321-1111
0-321-32132-1	Snoopy	232-234-1234
0-321-32132-1	Grumpy	665-235-6532
0-55-123456-9	Jones	123-333-3333
0-55-123456-9	Smith	654-223-3455
0-123-45678-0	Joyce	666-666-6666
1-22-233700-0	Roman	444-444-4444

First Normal Form

The relation is **not** in 1NF

EmpNum	**EmpPhone**	**EmpDegrees**
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

EmpDegrees is a multi-valued field:

employee 679 has two degrees: *BSc* and *MSc*

employee 333 has three degrees: *BA, BSc, PhD*

First Normal Form

EmpNum	EmpPhone	EmpDegrees
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

To obtain 1NF relations

Without loss of information, replace the above with two relations - see next slide

First Normal Form

Employee

EmpNum	EmpPhone
123	233-9876
333	233-1231
679	233-1231

EmployeeDegree

EmpNum	EmpDegree
333	BA
333	BSc
333	PhD
679	BSc
679	MSc

An outer join between Employee and EmployeeDegree will produce the information we saw before

Second Normal Form

- A *key attribute* is any attribute that is part of a key; any attribute that is not a key attribute, is a *non-key attribute*.
- A relation is in **2NF** if it is in 1NF, and every non-key attribute is fully dependent on each candidate key.
- A relation in 2NF will not have any partial dependencies

Second Normal Form (2NF)

For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be functionally dependent on the entire primary key

Example 1: 2 NF?

Relation □ {**Title, PubId, AuId**, Price, AuAddress}

1. Key □ {Title, PubId, AuId}
2. {**Title, PubId, AuID**} □ {Price}
3. {**AuID**} □ {AuAddress}
4. AuAddress does not belong to a key
5. AuAddress functionally depends on AuId which is a subset of a key

Not 2NF

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1: Converted to 2NF

Old Relation □ {**Title, PubId, AuId,** Price, AuAddress}

New Relation □ {**Title, PubId, AuId**, Price}

New Relation □ {**AuId**, AuAddress}

Third Normal Form (3NF)

- 2NF (and 3NF) both involve the concepts of key and non-key attributes.
- This form dictates that all **non-key** attributes of a table must be functionally dependent on a candidate key i.e. there can be no interdependencies among non-key attributes.
- For a table to be in 3NF, there are two requirements
  - The table should be second normal form
  - No attribute is transitively dependent on the primary key

Example 1: 3NF ?

Relation □ {**BuildingID**, Contractor, Fee}

1. Primary Key □ {BuildingID}
2. {**BuildingID**} □ {Contractor}
3. {**BuildingID**} □ {Fee}
4. Fee transitively depends on the BuildingID
5. {Contractor} □ {Fee}
6. Both Contractor and Fee depend on the entire key hence 2NF

BuildingID	Contractor	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

Not in 3NF

In this relation 3NF is violated since a non-key field is dependent on another non-key field and is transitively dependent on the primary key.

3NF - Decomposition

1. Move all items involved in transitive dependencies to a new entity.
2. Identify a primary key for the new entity.
3. Place the primary key for the new entity as a foreign key on the original entity.

Example 1: Converted to 3NF

Old Relation → {**BuildingID**, Contractor, Fee}

New Relation → {**BuildingID**, Contractor}

New Relation → {**Contractor**, Fee}

BuildingID	Contractor	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

BuildingID	Contractor
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

Contractor	Fee
Randolph	1200
Ingersoll	1100
Pitkin	1100

Check

Example A: 2NF and 3 NF?

Relation □ {**City, Street, HouseNumber**, HouseColor, CityPopulation}

1. key □ {City, Street, HouseNumber}
2. {City, Street, HouseNumber} □ {HouseColor}
3. {City} □ {CityPopulation}
4. CityPopulation does not belong to any key.
5. CityPopulation is functionally dependent on the City which is a proper subset of the key

Not 2NF, so not 3NF also

Example B: 2NF and 3 NF?

Relation □ {**studio, movie**, budget, studio_city}

6. Key □ {studio, movie}
7. {studio, movie} □ {budget}
8. {studio} □ {studio_city}
9. studio_city is not a part of a key
10. studio_city functionally depends on studio which is a proper subset of the key

Not 2NF, so not 3NF also

Convert

Example A: Convert to 2NF

Old Relation □ {**City**, **Street**, **HouseNumber**, HouseColor, CityPopulation}

New Relation □ {**City**, **Street**, **HouseNumber**, HouseColor}

New Relation □ {**City**, CityPopulation}

Example B: Convert to 2NF

Old Relation □ {**Studio**, **Movie**, Budget, StudioCity}

New Relation □ {**Movie**, **Studio**, Budget}

New Relation □ {**Studio**, StudioCity}

Check

Example C: 2 NF and 3NF ?

Relation □ {**Studio**, StudioCity, CityTemp}

In 2NF, but NOT 3NF

1. Primary Key □ {Studio}
2. {**Studio**} □ {StudioCity}
3. {**Studio**} □ {CityTemp}
4. Both StudioCity and CityTemp depend on the entire key hence 2NF
5. But, {StudioCity} □ {CityTemp}
6. CityTemp transitively depends on Studio hence violates 3NF

Convert

Example C: Convert to 3NF

Old Relation → {**Studio**, StudioCity, CityTemp}

New Relation → {**Studio**, StudioCity}

New Relation → {**StudioCity**, CityTemp}

Student Group Project

- Start the design of the ER-Diagram, relations, normalization

Normalization - BCNF

- Once the attributes are arranged in third normal form, the group of tables that they comprise is a well-structured relational database with no data redundancy.
- R. Boyce and E. F. Codd introduced a stronger definition of 3NF called Boyce-Codd Normal Form (BCNF).

Boyce-Codd Normal Form (BCNF)

- BCNF is a refinement of 3NF
  Drops the restriction of a non-key attribute from the 3rd normal form.
- BCNF does not allow dependencies between attributes that belong to candidate keys.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

- BCNF is defined very simply:

A relation is in BCNF,

if it is in 1NF and **if every determinant is a candidate key**.

Boyce-Codd Normal Form (BCNF)

- Third normal form and BCNF are not same if the following conditions are true:
  - The table has two or more candidate keys
  - At least two of the candidate keys are composed of more than one attribute
  - The keys are not disjoint i.e. the composite candidate keys share some attributes

Example 1 - Relation □ {City, Street, ZipCode }

1. Key1 □ {**City, *Street***}
2. Key2 □ {**ZipCode, *Street***}
3. No non-key attribute hence 3NF
4. {**City, Street**} □ {ZipCode}
5. {ZipCode} □ {City}
6. Dependency between attributes belonging to a key

Address (Not in BCNF)

BCNF - Decomposition

1. Place the two candidate primary keys in separate relations.
2. Place each of the remaining data items in one of the resulting relations according to its dependency on the primary key.

Example 1 (Convert to BCNF)

Old Relation □ {**City, Street**, ZipCode }

New Relation1 □ {Street, ZipCode}

New Relation2 □ {City, ZipCode}

- **Loss-less and Dependency preserving decomposition**
  - If decomposition does not cause any loss of information it is called a **lossless** decomposition.
  - If a decomposition does not cause any dependencies to be lost it is called a **dependency-preserving** decomposition.

Boyce-Codd Normal Form (BCNF)

Example 2: Suppose there is a company wherein employees work in **more than one department**.

Store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Candidate key: {**emp_id, emp_dept**}

Functional dependencies in the table above:

emp_id → emp_nationality

emp_dept → {dept_type, dept_no_of_emp}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

Boyce-Codd Normal Form (BCNF)

Example 2: Suppose there is a company wherein employees work in **more than one department**.

The data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

To make the table comply with BCNF we can break the table in three tables like this:

Table 3: emp_dept

emp_id	**emp_dept**
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Table 1: emp_nationality

emp_id	emp_nationality
1001	Austrian
1002	American

Table 2: emp_dept_mapping

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

This is now in BCNF as in both the functional dependencies left side part is a key.

Decomposition – Loss of Information

1. Any table Relation can be decomposed in a lossless way into a collection of smaller schemas that are in BCNF form. However the dependency preservation is not guaranteed.
2. Any table can be decomposed in a lossless way into 3rd normal form that also preserves the dependencies.
   - **3NF may be better than BCNF in some cases**
3. If our database will be used for OLTP (on line transaction processing), then BCNF is our target. Usually, we meet this objective.
   - However, we might denormalize (3NF, 2NF, or 1NF) for performance reasons

.

Use your own judgment when decomposing schemas

Normalization

Also,
  any relation that is in BCNF, is in 3NF;
  any relation in 3NF is in 2NF; and
  any relation in 2NF is in 1NF.

There is a sequence to normal forms:
  1NF is considered the weakest,
  2NF is stronger than 1NF,
  3NF is stronger than 2NF, and
  BCNF is considered the strongest

If our database will be used for OLTP (On Line Transaction Processing), then BCNF is our target (Usually, we meet this objective)

However, we might denormalize (3NF, 2NF, or 1NF) for performance reasons.

Other Higher Normal Forms

- Higher normal forms that go beyond BCNF were introduced later such as Fourth Normal Form (4NF) and Fifth Normal Form (5NF). However these later normal forms deal with situations that are very rare.

Fourth Normal Form (4NF)

- Fourth normal form eliminates independent many-to-one relationships between columns.
- To be in Fourth Normal Form,
  - A relation must first be in Boyce-Codd Normal Form.
  - A given relation may not contain more than one multi-valued attribute.

Example 1: 4NF ?

Relation □ {MovieName, ScreeningCity, Genre)

Primary Key: {MovieName, ScreeningCity, Genre)

1. All columns are a part of the only candidate key, hence BCNF
2. Many Movies can have the same Genre
3. Many Cities can have the same movie
4. Violates 4NF

Not 4NF

Movie	ScreeningCity	Genre
Hard Code	Los Angles	Comedy
Hard Code	New York	Comedy
Bill Durham	Santa Cruz	Drama
Bill Durham	Durham	Drama
The Code Warrier	New York	Horror

4NF - Decomposition

1. Move the two multi-valued relations to separate tables
2. Identify a primary key for each of the new entity.

Example 1: Converted to 3NF

Old Relation □ {MovieName, ScreeningCity, Genre}

New Relation □ {MovieName, ScreeningCity}

New Relation □ {MovieName, Genre}

Movie	**Genre**
Hard Code	Comedy
Bill Durham	Drama
The Code Warrier	Horror

Movie	**ScreeningCity**
Hard Code	Los Angles
Hard Code	New York
Bill Durham	Santa Cruz
Bill Durham	Durham
The Code Warrier	New York

Check

Example D - Movie BCNF or 4NF?

Relation □ {MovieTitle, MovieID, PersonName, Role, Payment }

1. Key1 □ {MovieTitle, PersonName}
2. Key2 □ {MovieID, PersonName}
3. Both role and payment functionally depend on both candidate keys thus 3NF
4. {MovieID} □ {MovieTitle}
5. Dependency between MovieID & MovieTitle Violates BCNF

Not in BCNF,
so also Not in 4NF

Example E - Consulting (BCNF ?)

Relation □ {Client, Problem, Consultant}

6. Key1 □ {Client, Problem}
7. Key2 □ {Client, Consultant}
8. No non-key attribute hence 3NF
9. {Client, Problem} □ {Consultant}
10. {Client, Consultant} □ {Problem}
11. Dependency between attributes belonging to keys violates BCNF

Not in BCNF,
so also Not in 4NF

BCNF - Decomposition

Example 2 (Convert to BCNF)

Old Relation → {MovieTitle, MovieID, PersonName, Role, Payment }

New Relation → {MovieID, PersonName, Role, Payment}

New Relation → {MovieTitle, PersonName}

- Loss of relation {MovieID} □ {MovieTitle}

New Relation → {MovieID, PersonName, Role, Payment}

New Relation → {MovieID, MovieTitle}

- We got the {MovieID} □ {MovieTitle} relationship back

Example 3 (Convert to BCNF)

Old Relation → {Client, Problem, Consultant}

New Relation → {Client, Consultant}

New Relation → {Client, Problem}

Fourth Normal Form (4NF)

Example F: 4NF ?

Relation □ {Manager, Child, Employee}

1. Primary Key □ {Manager, Child, Employee}
2. Each manager can have more than one child
3. Each manager can supervise more than one employee
4. 4NF Violated

Manager	Child	Employee
Jim	Beth	Alice
Mary	Bob	Jane
Mary	NULL	Adam

Not in 4NF

Example G: 4NF ?

Relation □ {Employee, Skill, ForeignLanguage}

5. Primary Key □ {Employee, Skill, Language }
6. Each employee can speak multiple languages
7. Each employee can have multiple skills
8. Thus violates 4NF

Not in 4NF

Employee	Skill	Language
1234	Cooking	French
1234	Cooking	German
1453	Carpentry	Spanish
1453	Cooking	Spanish
2345	Cooking	Spanish

4NF - Decomposition

Example F: Converted to 4NF

Old Relation → {Manager, Child, Employee}

New Relation → {Manager, Child}

New Relation → {Manager, Employee}

Manager	Child
Jim	Beth
Mary	Bob

Manager	Employee
Jim	Alice
Mary	Jane
Mary	Adam

Example G Converted to 4NF

Old Relation → {Employee, Skill, ForeignLanguage}

New Relation → {Employee, Skill}

New Relation → {Employee, ForeignLanguage}

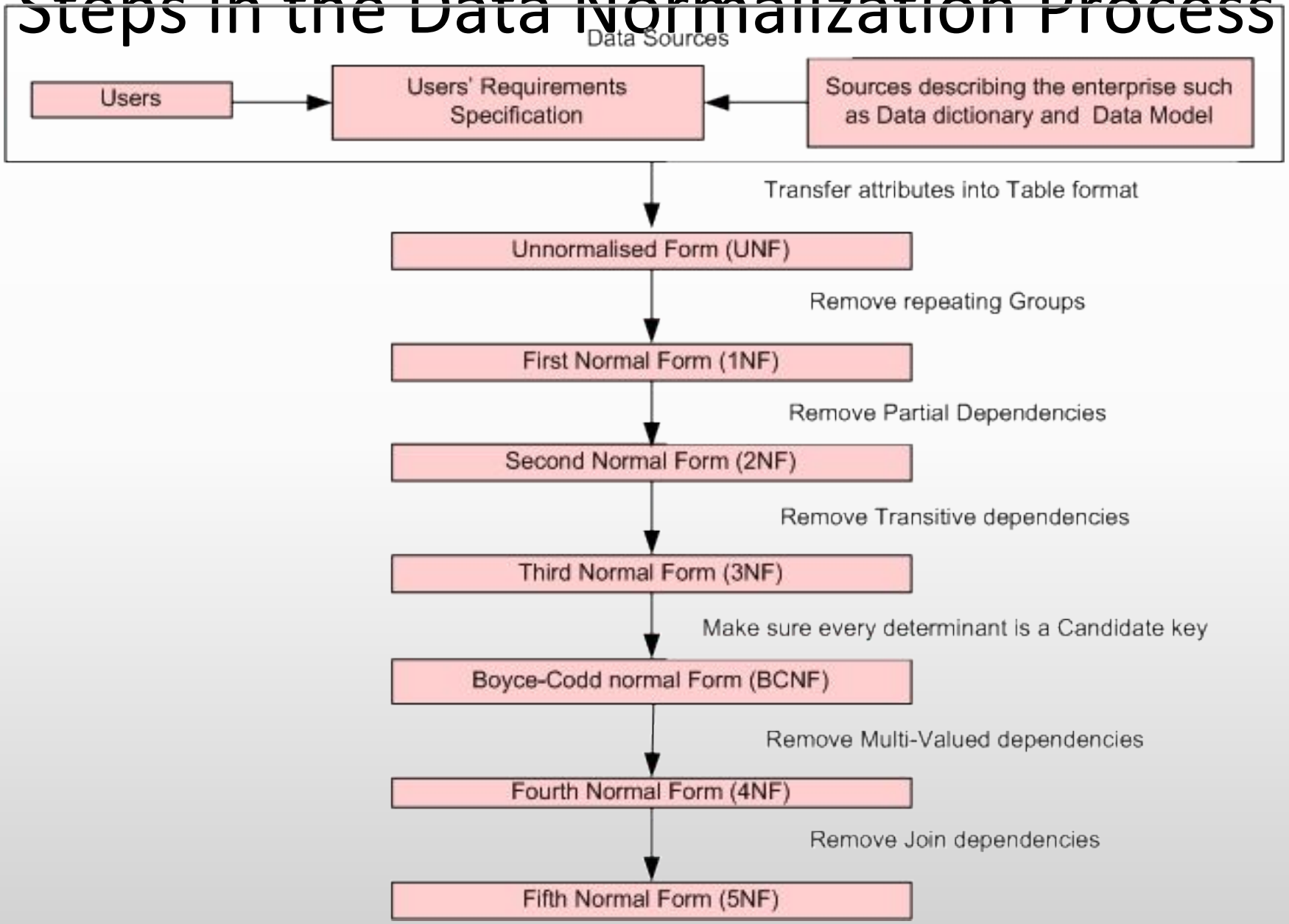
Employee	Skill
1234	Cooking
1453	Carpentry
1453	Cooking
2345	Cooking

Employee	Language
1234	French
1234	German
1453	Spanish
2345	Spanish

Fifth Normal Form (5NF)

- Fifth normal form is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy. Once it is in fifth normal form it cannot be broken into smaller relations without changing the facts or the meaning.

Steps in the Data Normalization Process



Domain Key Normal Form (DKNF)

- The relation is in DKNF when there can be no insertion or deletion anomalies in the database.

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - Normalization breaks *R* into smaller relations.
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity
- E.g. *employee* entity with attributes *department-number* and *department-address*, and an FD *department-number* → *department-address*
  - Good design would have made department an entity
- FDs from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- E.g. displaying *customer-name* along with *account-number* and *balance* requires join of *account* with *depositor*
- Alternative 1: Use denormalized relation containing attributes of *account* as well as *depositor* with all above attributes
  - Faster lookup
  - Extra space and extra execution time for updates
  - Extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as
  account ⋈ depositor
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

End of Presentation