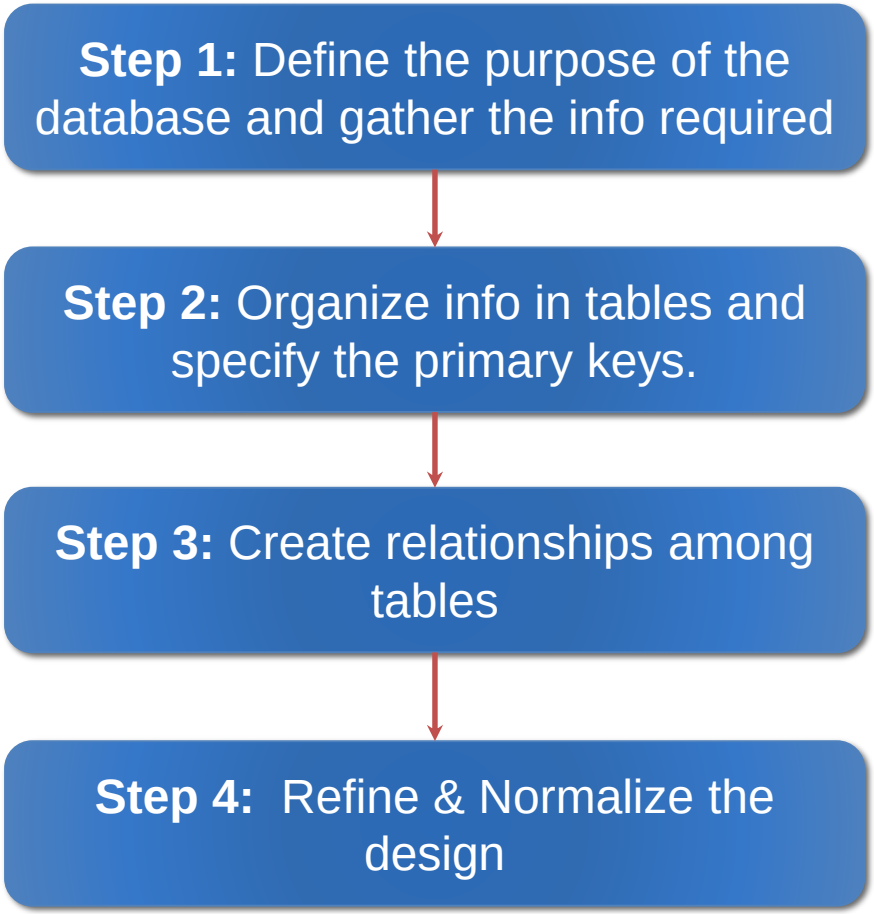
DATABASE NORMALIZATION

Chapter 19

Database Design

- Database Design is essentially the the process of creating the database tables and deciding which attributes should be placed in each table.
- Traditionally an Entity-Relational (ER) Diagram is used to capture the application requirements in a conceptual model, where each real-world object is modeled as an entity
- The database design is generated by transforming the entities in a ER diagram into tables.
- Prior to 1972, we had no formal method of deciding whether the defined tables were ‘well formed.’
- Edgar Codd introduced the concept of normalization to refine table design.

Database Design Process



- Understand the application and gather info

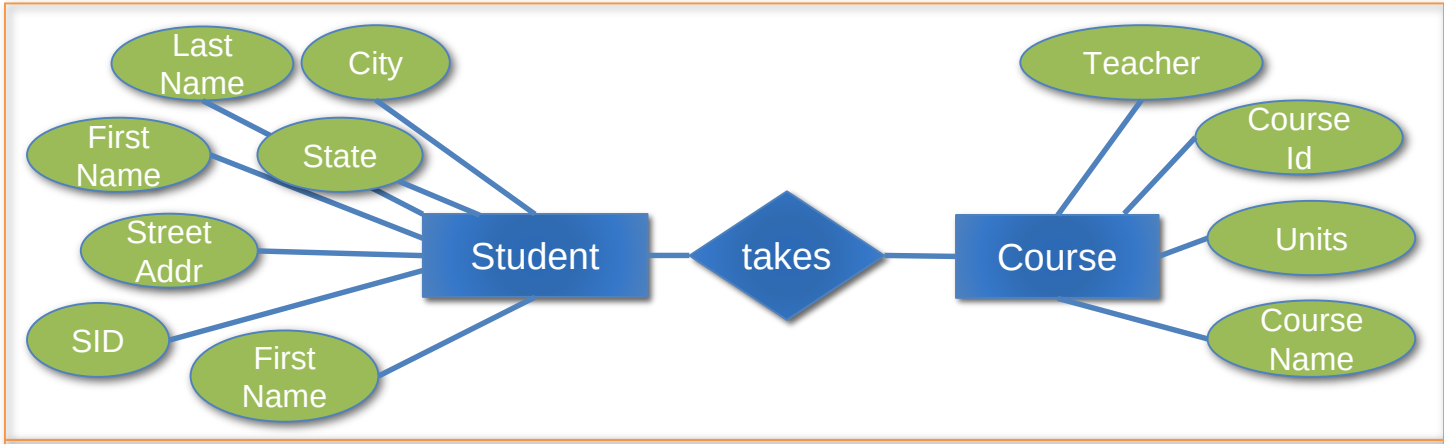
- Define the relevant attributes
- Organize the attributes into tables
- Identify the primary key for each table

- Examine each table and decide how one table is related to other table
- Create new tables (or fields) to clarify the relationships

- Refine the tables
- Split a large table into 2+ smaller tables

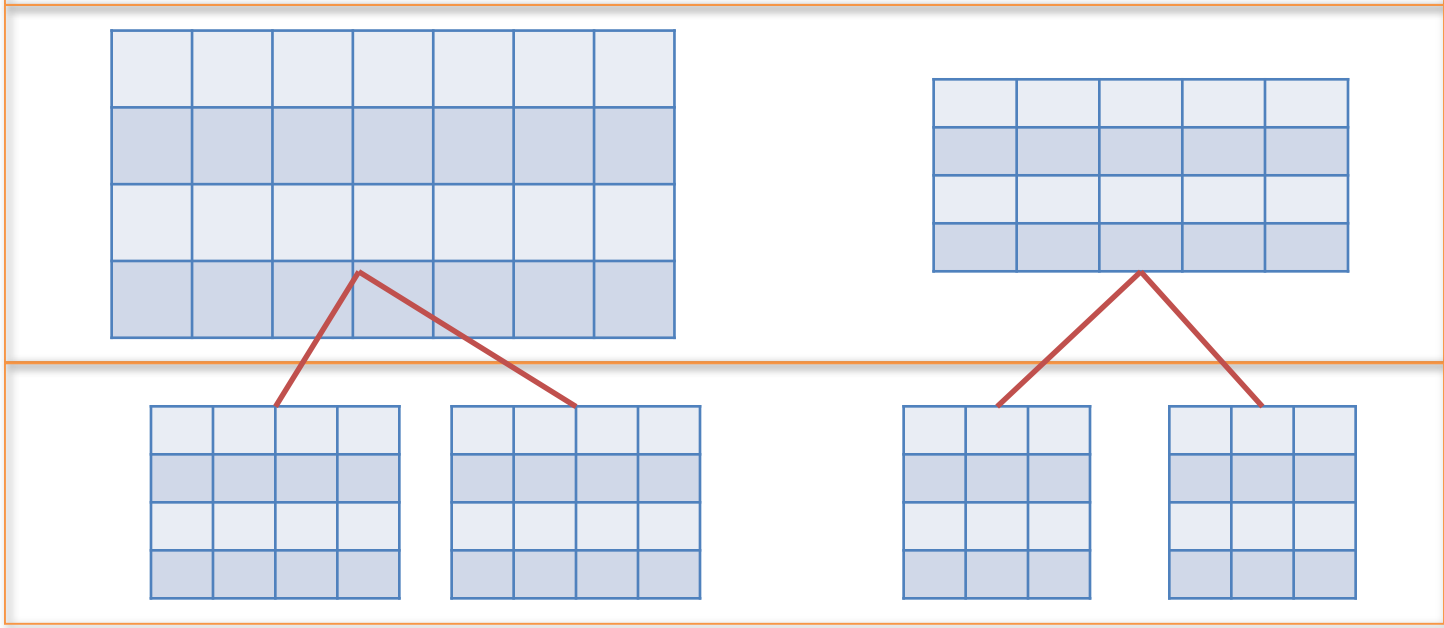
Database Design

Conceptual Model



Relational Model

Normalization
(to eliminate anomalies)



Outline

- Why consider database normalization
  - Data anomalies
    - redundant, update, insertion, deletion

- Informal rules for database design
- Functional Dependencies (FD)
- 1st Normal Form (1NF)
- 2nd Normal Form (2NF)
- 3rd Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- De-normalization

Normalization

What is normalization??

A technique for organizing the data into multiple related tables to minimize redundancy.

Database Design (Cont.)

- Let us see some examples of bad schema design.

- **Student** (SID, FirstName, LastName, StreetAddr, City, State, Zipcode, CourseId, CourseName)

SID	First Name	Last Name	StreetAddr	City	State	Zipcode	CourseId	CourseName
342	Dori	Smith	42 Wallaby Way	Riverside	CA	93940	CS 263	Intro to Database Systems
937	Andy	Lee	234 Elm Street	Riverside	CA	93940	CS 238	Intro to Data Structures
937	Andy	Lee	234 Elm Street	Riverside	CA	93940	CS 263	Intro to Database Systems
675	Karen	Wayne	1313 Mockingbird Ln	Riverside	CA	93940	CS 306	Game Engine Programming
173	David	Potter	4 Privet Drive	Riverside	CA	93940	CS 263	Intro to Database Systems
173	David	Potter	4 Privet Drive	Riverside	CA	93940	CS 282	Intro to Network Security

- Redundancy causes several problems
  - Redundant storage
  - Insertion/deletion/update anomalies

Database Design (Cont.)

- Before going over these informal database design guidelines, lets see some examples of bad schema design.

SID	First Name	Last Name	Street Addr	City	State	Zipcode	Course Id	Course Name
342	Dori	Smith	42 Wallaby Way	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 238	Intro to Data Structures
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
675	Karen	Wayne	1313 Mockingbird Ln	Monterey Bay	CA	93940	CS 306	Game Engine Programming
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 282	Intro to Network Security

The address, city, state, and zipcode are stored multiple times.

The course name is stored multiple times

Database Design (Cont.)

- Before going over these informal database design guidelines, lets see some examples of bad schema design.

SID	First Name	Last Name	Street Addr	City	State	Zipcode	Course Id	Course Name
342	Dori	Smith	42 Wallaby Way	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 238	Intro to Data Structures
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
675	Karen	Wayne	1313 Mockingbird Ln	Monterey Bay	CA	93940	CS 306	Game Engine Programming
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 282	Intro to Network Security

- **Update anomaly:** An update to an attribute may require multiple updates due to redundancy.
  - Ex. Changing the ‘Street Addr’ of the student 173 will require 2 updates.
  - Ex. Changing the ‘Course Name’ of CS 263 will require 3 updates.

Database Design (Cont.)

- Before going over these informal database design guidelines, lets see some examples of bad schema design.

SID	First Name	Last Name	Street Addr	City	State	Zipcode	Course Id	Course Name
342	Dori	Smith	42 Wallaby Way	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 238	Intro to Data Structures
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
675	Karen	Wayne	1313 Mockingbird Ln	Monterey Bay	CA	93940	CS 306	Game Engine Programming
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 282	Intro to Network Security
654	Madi	Kim	5 Buttonwood	Monterey Bay	CA	93940	?	?

Insertion anomaly: If we add a new student, he/she has to be registered for a course, or we must place ‘NULL’ for fields that are not relevant.

Deletion anomaly: If student ‘Dori Smith’ drops the course ‘263’, we will delete the whole row and lose all the information we have for that student.

Informal Guidelines for Database Design

GUIDELINE 1:

- Each tuple/row in a relation should represent one entity or relationship instance.
- Attributes of different entities (Student, Course, Faculty) should not be mixed in the same relation.
- Only foreign keys should be used to refer to other entities.

SID	First Name	Last Name	Street Addr	City	State	Zipcode	Course Id	Course Name
342	Dori	Smith	42 Wallaby Way	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 238	Intro to Data Structures
937	Andy	Lee	234 Elm Street	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
675	Karen	Wayne	1313 Mockingbird Ln	Monterey Bay	CA	93940	CS 306	Game Engine Programming
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 263	Intro to Database Systems
173	David	Potter	4 Privet Drive	Monterey Bay	CA	93940	CS 282	Intro to Network Security

In this example, we have two real-world entities being represented, ‘student’ and ‘course’, and their attributes should not be in the same relation.

Database Design

- Measures of quality
  - Making sure attribute semantics are clear
  - Reducing redundant information in tuples
  - Reducing NULL values in tuples
  - Disallowing possibility of generating spurious tuples

- Database designers generally decide the database design using informal guidelines.

Informal Guidelines for Database Design

- **GUIDELINE 2:** Design a schema that does not suffer from the insertion, deletion and update anomalies.

- **GUIDELINE 3:** Relations should be designed such that their tuples will have as few NULL values as possible

  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)

  - Reasons for nulls:
    - attribute not applicable or invalid
    - attribute value unknown
    - value known to exist, but unavailable

Informal Guidelines for Database Design

- **GUIDELINE 4:** The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join of any relations.

Car(ID, Color, Make)

ID	Color	Make
123	Gray	Honda Accor
456	Red	Toyota Camry
567	Red	Lexus Rx
789	Blue	Toyota RV4

Car1 (ID, Color)

ID	Color
123	Gray
456	Red
567	Red
789	Blue

Car2 (Color, Make)

Color	Make
Gray	Honda Accor
Red	Toyota Camry
Red	Lexus Rx
Blue	Toyota RV4

Car_Joint(ID, Color, Make)

ID	Color	Make
123	Gray	Honda Accor
456	Red	Toyota Camry
456	Red	Lexus Rx
567	Red	Toyota Camry
567	Red	Lexus Rx
789	Blue	Toyota RV4

Normalization

- Codd discussed several Normal Forms (1NF, 2NF, 3NF, BCNF)
- If a relation is in a certain *normal form*, it is known that certain kinds of problems are avoided or minimized.
- *Normalization* is a process of analyzing a relation to ensure it is well formed and applying refinement if required.
- The main refinement technique for normalization is decomposition.
  - i.e. decomposing a table into 2 or more smaller tables.
  - Ex. Given a table with attributes A, B, C, D we may decide to compose this table into AB, BCD, ACD, etc.

- Codd introduced the concept of *functional dependencies* to decide which *normal form* hold over the relation, and then decide whether it needs to be normalized (or decomposed).

Functional Dependencies

- *Functional dependencies* (FDs) are **constraints** that are derived from the *meaning* and *relationships* of the attributes.
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y
- X → Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y.
- For any two tuples t_1 and t_2 in any relation instance r(R):
  If $t_1[X]=t_2[X]$, *then* $t_1[Y]=t_2[Y]$

Functional Dependencies Example

Car(ID, Color, Make)

ID	Color	Make
123	Gray	Honda Accor
456	Red	Toyota Camry
567	Red	Lexus Rx
789	Blue	Toyota RV4

We can say that if we know the ID of the car, we can derive the Color and Make

ID ⟶ Color, Make

or written as

ID ⟶ Color

ID ⟶ Make

Properties of functional dependencies

Armstrong's axioms

If A, B, C, and D are sets of attributes, then we have the following properties:

- **Transitive:** If A → B and B → C, then A → C
- **Reflexive:** For every subset S of A, A → S
- **Union:** If A → B and A → C, then A → BC
- **Augmentation:** If A → B, then AC → B
- **Decomposition:** If A → BC, then A → B
- **Psudo-Transitive:** If A → B and CB → D, then CA → D.
- If K is a key of a given relation, then K → A, for every set of attributes A of the given relation.

Inference Rules for FDs

- **Closure** of a set F of FDs is the set $\mathbf{F^+}$ of all FDs that can be inferred from F.
- **Closure** of a set of attributes X with respect to F is the set $\mathbf{X^+}$ of all attributes that are functionally determined by X.
- The closure can be calculated by repeatedly applying **Reflexive**, **Augmentation**, **Transitive** inference rules using the FDs in F.

1st Normal Form (1NF)

- **1NF:** A relation is in 1NF if each attribute is atomic (simple structure, i.e., a string, integer, double, boolean, etc.). It disallows multi-valued and composite attributes.
- Example

Not in 1NF

SID	First Name	Last Name	Course Id
342	Dori	Smith	CS 263
937	Andy	Lee	CS 238, CS263
675	Karen	Wayne	CS 306
173	David	Potter	CS 263, CS 282

In 1NF

SID	First Name	Last Name	Course Id
342	Dori	Smith	CS 263
937	Andy	Lee	CS 238
937	Andy	Lee	CS 263
675	Karen	Wayne	CS 306
173	David	Potter	CS 263
173	David	Potter	CS 282

- 1NF suffers from
  - Redundancy and insertion/deletion/update anomalies

2nd Normal Form (2NF)

- **2NF:** A relation is in 2NF if it is in 1NF and has no partial dependencies.

- A dependency **A** → **B** is a *partial dependency* if
  - B is a not subset of A
  - A is a proper subset of some key for the relation
  - B is not part of any candidate key of the table (relation) .

- Essentially, a table is not in 2NF if an attribute is partially dependent on a subset of the a candidate key.

2NF Example

Student_Activities (SID, Activity, Fee)
SID, Activity → Fee
Activity → Fee This is a partial dependency

Decompose into 2 relations

SID	Activity	Fee
100	Skiing	200
100	Golf	65
150	Swimming	50
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

In 1NF, but not in 2NF. WHY?
Fee is determined by Activity, but the primary key is {SID, Activity}. Thus this is a partial dependency

Student(SID, Activity)

SID	Activity
100	Skiing
100	Golf
150	Swimming
175	Squash
175	Swimming
200	Swimming
200	Golf

Activity(Activity, Fee)

Activity	Fee
Skiing	200
Swimming	50
Squash	50
Golf	65

3rd Normal Form (3NF)

- **3NF:** A relation schema <R, F>, where R is the relational schema and F is the set of functional dependencies for this schema, is in 3NF if it is in 2NF and has no transitive dependencies.

- A *transitive dependency* is a functional dependency on a non-primary key attribute.
- Essentially, for a relation to be in 3NF
  - For any dependency X → A in F+ at least one of the following holds:
    - ***X*** is **super key**, ***or***
    - ***A*** is **key attribute, i.e. A-X is contained in a key**

Transitive dependency example

The functional dependency {Book} → {Author Nationality} applies; that is, if we know the book, we know the author's nationality. Furthermore:

- {Book} → {Author}
- {Author} does not → {Book}
- {Author} → {Author Nationality}

Therefore {Book} → {Author Nationality} is a transitive dependency.

Transitive dependency occurred because a non-key attribute (Author) was determining another non-key attribute (Author Nationality).

3NF Example 1

SID	Building	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

Housing (SID, Building, Fee)
SID → Building → Fee

In 2NF, but not in 3NF.
WHY?
Fee is a functionally dependency on a non-prime key attribute Building.

Solution :

Non-key determinate with transitive dependencies goes into a new table

Non-key determinate becomes primary key in the new table and remains as a foreign key in the old table.

3NF Example 1 (Cont.)

SID	Building	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

Student-Housing (SID, Building)
Key: SID

SID	Building
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

Building-Fee (Building, Fee)
Key: Building

Building	Fee
Randolph	1200
Ingersoll	1100
Pitkin	1100

3NF Example 2

Order(OrderNo, PizzaCode, Quantity), where OrderNo,PizzaCode -> Quantity

Order No	Pizza Code	Quantity
123	P	1
235	V	2
242	PP	3
732	PC	1
346	S	1

In 3NF?

Yes, we have only 1 attribute (Quantity) is is a non-prime attribute, and it is dependent on the key.

Pizza(PizzaCode, PizzaName, Pricey), where PizzaCode -> PrizzaName, Price

Pizza Code	Pizza Name	Price
P	Pepperoni	10
V	Veggie	5
PP	Double Pepperoni	10
PC	Pepperoni with extra cheese	16
S	Sausage	10

In 3NF?

Yes, both non-prime attributes (PizzaName and Price) are dependent on the key.

Minimal Cover

Definition: Minimal Cover of a set of FDs is a minimal set of dependencies that are equivalent to the given set of dependencies, i.e it does not contain any redundant dependency. The following procedure is basically examines every attribute in the left parts of every dependency to remove it if it is redundant, and then examine every dependency for the same purpose.

Procedure for Finding a Minimal Cover F for a Set E of FDs

1. Set F= E
2. Replace each FD $X \rightarrow A_1A_2 \ldots A_n$ in F by the n FDs
   $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$
3. Repeat the following until no more attributes can be removed:
   for each FD $X \rightarrow A$ in F **then**
   for each attribute B that is an element of X **then**
   if $\{F-\{X \rightarrow A\}\} \cup \{X-\{B\} \rightarrow A\}$ is equivalent to F **then**
   replace $X \rightarrow A$ with $X-\{B\} \rightarrow A$ in F
4. Repeat the following until no more dependencies can be removed:
   for each remaining FD $X \rightarrow A$ in F
   if$\{ F- \{X \rightarrow A\}$ is equivalent to F**, then**
   remove $X \rightarrow A$ from F.

Decomposition to 3NF

Given a set of attributes R and a set of dependencies F

Procedure Decompose3NF(R, F)

1. Find a minimal cover G for F, i.e. $F^+ = G^+$.
2. Partition F into groups such that the dependencies in each group share the same left-hand side.
3. For each group of dependencies, create one relation.
4. If none of the created relations contains a key for the original relation scheme, then add a new relation scheme that consists only the attributes of some key of the original relation.
5. Create a relation that contains all attributes which have not been placed in any relation.
6. Eliminate any redundant relations (if relation A is subset of relation B, then A is redundant)

You may remove the last two steps

End Procedure

Example – Decompose to 3NF (Cont.)

- **Example:**
  - Empoyee_Project (E_Ssn, E_Salary, E_Phone, P_No ,P_Location, D_No)
  - Given Dependencies:
    - E_ssn → E_salary, E_phone, D_no
    - P_no → P_name, P_location
    - E_ssn, P_no → E_salary, E_phone, D_no, P_name, P_location

1. Find the Minimal Cover
   - E_ssn → E_salary
   - E_ssn → E_phone
   - E_ssn → D_no
   - P_no → P_name
   - P_no → P_location
   - E_ssn, P_no → E_salary
   - E_ssn, P_no → E_phone
   - E_ssn, P_no → D_no
   - E_ssn, P_no → P_name
   - E_ssn, P_no → P_location

Step 1 - Split the dependencies into multiple dependencies (only one attribute on the right side)

Example – Decompose to 3NF (Cont.)

- **Example:**
  - Empoyee_Project (E_Ssn, E_Salary, E_Phone, P_No ,P_Location, D_No)
  - Dependencies:
    - E_ssn → E_salary, E_phone, D_no
    - P_no → P_name, P_location
    - E_ssn, P_no → E_salary, E_phone, D_no, P_name, P_location

1. Find the Minimal Cover
   - E_ssn → E_salary
   - E_ssn → E_phone
   - E_ssn → D_no
   - P_no → P_name
   - P_no → P_location
   - E_ssn, P_no → E_salary
   - E_ssn, P_no → E_phone
   - E_ssn, P_no → D_no
   - E_ssn, P_no → P_name
   - E_ssn, P_no → P_location

Step 2 - Get rid of redundant attributes on the left hand side(reduce left sides)

Example – Decompose to 3NF (Cont.)

- **Example:**
  - Empoyee_Project (E_Ssn, E_Salary, E_Phone, P_No ,P_Location, D_No)
  - Dependencies:
    - E_ssn → E_salary, E_phone, D_no
    - P_no → P_name, P_location
    - E_ssn, P_no → E_salary, E_phone, D_no, P_name, P_location

1. Find the Minimal Cover
   - E_ssn → E_salary
   - E_ssn → E_phone
   - E_ssn → D_no
   - P_no → P_name
   - P_no → P_location
   - E_ssn → E_salary
   1. E_ssn → E_phone
   - E_ssn → D_no
   - E_ssn → P_name
   - E_ssn → P_location

Step 3 - Get rid of redundant dependencies

Example – Decompose to 3NF (Cont.)

2. Partition F into groups such that the dependencies in each group share the same left-hand side.
   - E_ssn → E_salary
   - E_ssn → E_phone
   - E_ssn→ D_no

   P_no → P_name
   P_no → P_location

2. For each group of dependencies, create one relation.
   - E_ssn → E_salary, E_phone, D_no
   - P_no → P_name, P_location

2. If none of the created relations contains a key for the original relation scheme, then add a new relation that consists only the attributes of some key of the original relation.
3. Create a relation that contains all attributes which have not been placed in any relation.
4. Eliminate any redundant relations. remove the last two steps

After applying the procedure, we get:
Employee(E_ssn, E_salary, E_phone, D_no) and

- Project(P_no,P_name,P_location). Each relation of the relations is in 3NF.

Example 2- Decompose to 3NF

- Given the relation schema (C,T,H,R,S,G) decompose to 3NF

$C \rightarrow T$
$HR \rightarrow C$
$HT \rightarrow R$
$CS \rightarrow G$
$HS \rightarrow R$
$CH \rightarrow R$

C = Course,
T = Teacher,
H = Hour (Time),
R = Room#,
S = Student,
G = Grade

1. Find the minimal cover

$C \rightarrow T$
$HR \rightarrow C$
$HT \rightarrow R$
$CS \rightarrow G$
$HS \rightarrow R$
~~$CH \rightarrow R$~~

2. After applying the procedure, we get (CT), (CHR), (HRT), (CGS), and (HSR)

The resulting relations are in 3NF, but there are still problems. Ex. A 'Teacher' cannot be added unless he is teaching a course.

Schedule for Week 10

Lecture

- Monday
  - Normalization example
  - Query Optimization
- Wednesday
  - Triggers
- Friday
  - Review

Lab

- Project demos
- Mini lab on triggers

Lets review

- An attribute that does not occur in ANY candidate key is called a **non-prime attribute**.

2nd Normal Form

It should not have any partial dependencies.

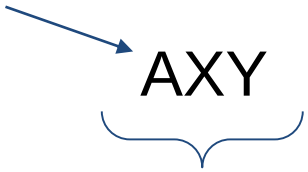
i.e. For a functional dependency A → B

prime attribute

non-prime attribute

There should not exists an instance where

A is part of the key



candidate key

B depends on A

B doesn't need to depend on AXY

3rd Normal Form

It should not have any transitive dependencies.

i.e. if we have functional dependency

A → B

non-prime attribute

non-prime attribute

This is a transitive dependency

Boyce-Codd Normal Form (BCNF)

It must satisfy two conditions:

- Must be in 3rd normal form
- For any dependency of the form A $\rightarrow$ B,
  A should be a super key.

Boyce-Codd Normal Form (BCNF)

- **Boyce-Codd Normal Form (BCNF)** – R is in BCNF if for every FD X → A in F, either
  - A is a subset of X(trivial dependency) or
  - X is a superkey

Advisor (SID, Major, Advisor_name)
Key(primary): SID, Major
Key(candidate): SID, Advisor_name

The relation has a problem because an 'advisor' cannot be added unless he is advising a student.

SID	Major	Advisor_name
100	Math	Cauchy
150	Psychology	Jung
200	Math	Riemann
250	Math	Cauachy
300	Psychology	Perls
300	Math	Riemann

Functional Dependency:
Advisor_name → Major

This relation is in 3NF, because there is no transitive dependency, and we have three dependencies , two of them their left part is a super key and the left part of the third dependency is a prime attribute.

Definitions

- Before proceeding to BCNF decomposition procedure, lets define some terms.

- **F+**: Let F be a set of dependencies, the closure of F, referred to as F^+, is the set of non-trivial dependencies that can be inferred from F using Armstrong's axioms.

- **Projection:**
  - Let X be a set of attributes and F be a set of dependencies among the attributes in X.
  - For a given Y subset X, the projection π(F,Y) is the set of all dependencies in F^+ that only include the attributes of Y and is a minimal cover.

Boyce-Codd Normal Form (BCNF)

- Decomposing a relation R into BCNF
  1. Suppose R is not in BCNF. For an FD $X \rightarrow A$ that causes a violation of BCNF, decompose R into R-A and $X \bigcap A$.
  2. If either R-A or $X \bigcap A$ is not in BCNF, decompose them further by a recursive application of this algorithm.
     - **R-A** denotes the attributes in R minus those in A
     - **$X \bigcap A$** is the union on attributes in X and A
  3. For each decomposition Y, compute $\pi(F,Y)$

- Note, if several dependencies violate BCNF, depending on which of these dependencies we consider to decompose first, we may arrive at different relations.

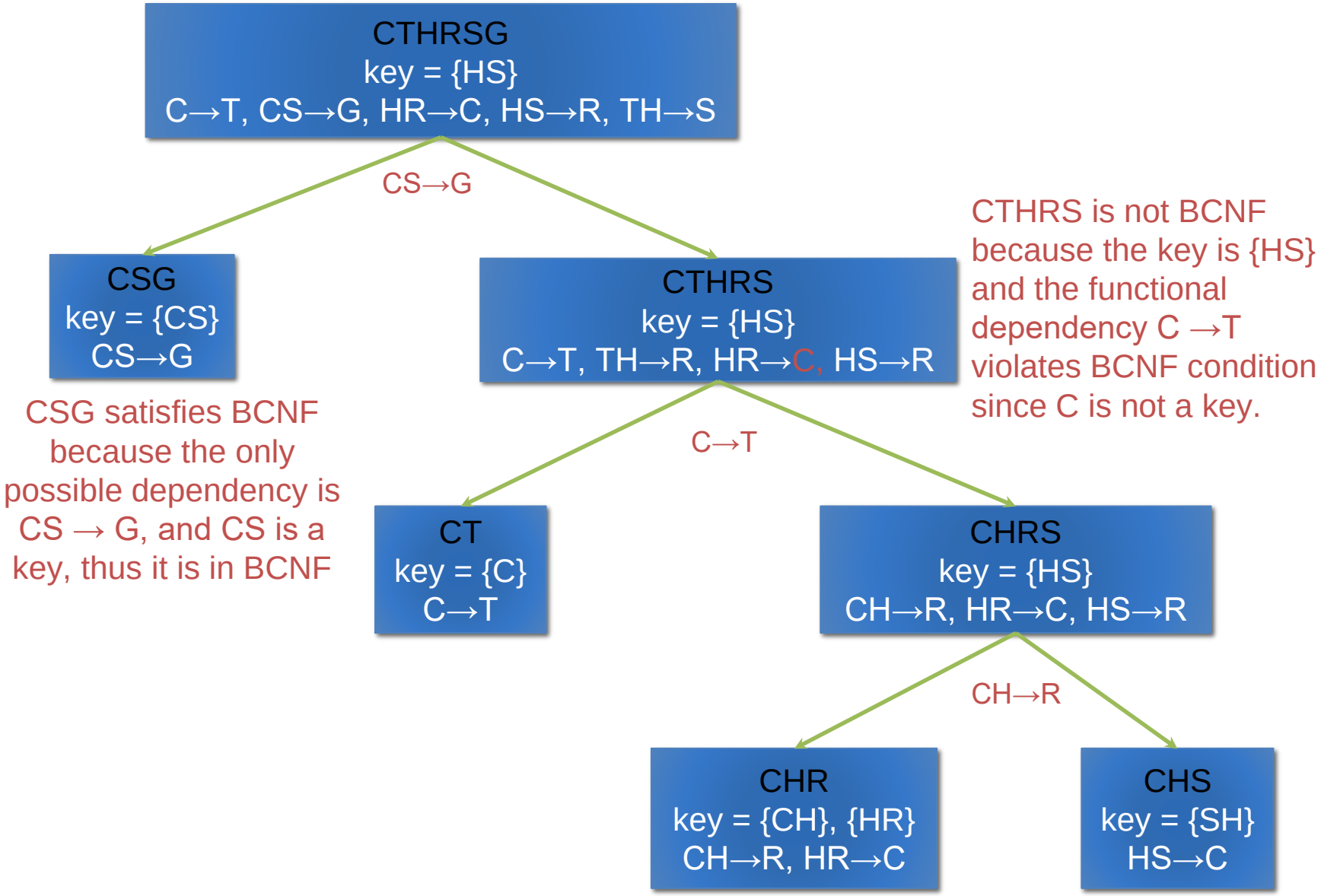
- BCNF guarantees loss-less join decomposition, however, it does not guarantee dependency preserving decomposition (while 3NF does.)

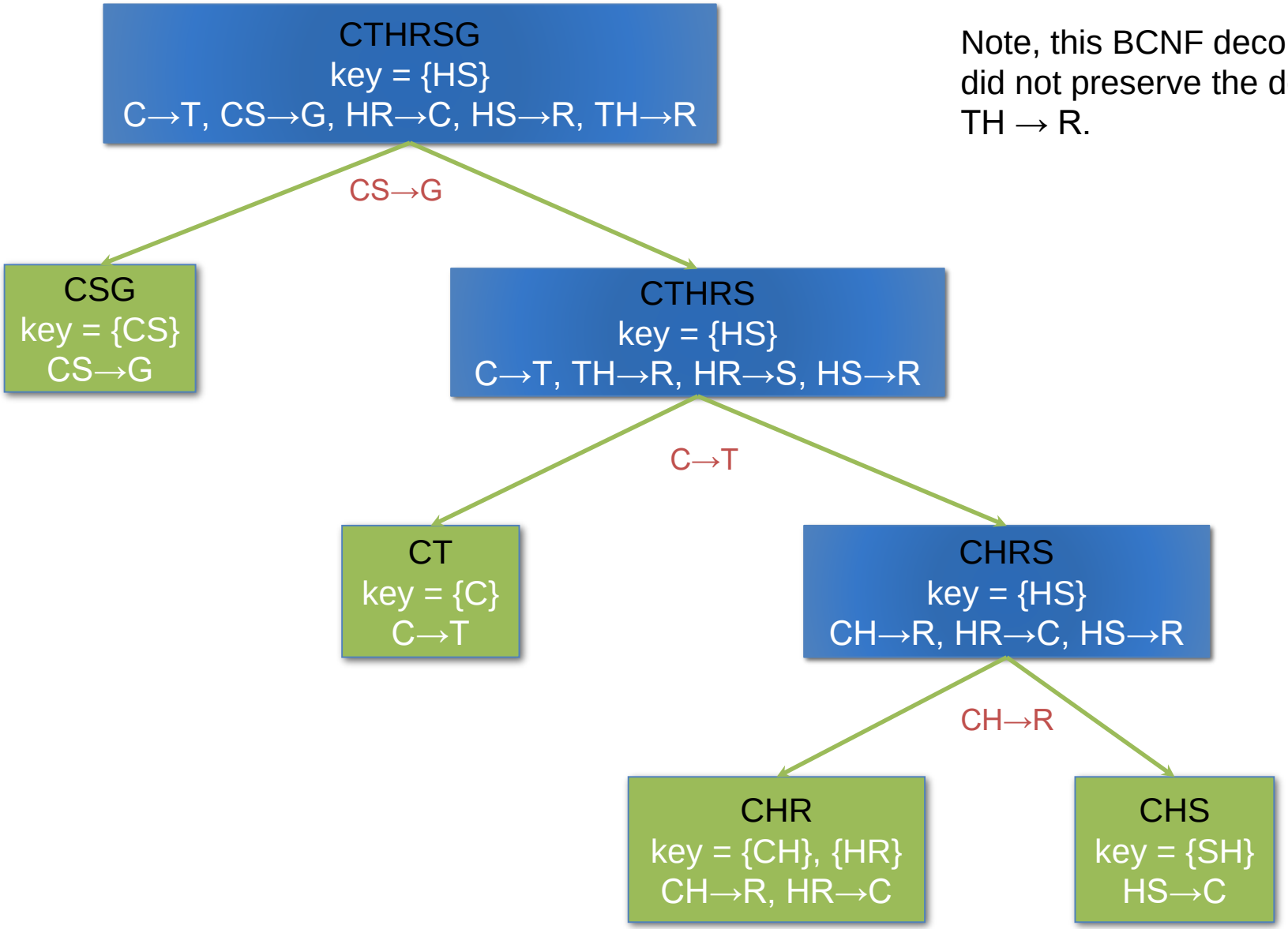
Example

- Given the relation schema (C,T,H,R,S,G) with key (H,S), decompose BCNF
  C = Course, **T** = Teacher, **H** = Hour, **R** = Room#, **S** = Student, **G** = Grade

C → T	each course has only one teacher
HR → C	at most one course can meet in a room at one time
HT → R	each student has at most one grade in each course
CS → G	a student can be in only one room at one time
HS → R	a student can attend at most one course at a given time

- Is this relation in BCNF?
  - No, because dependency CS→G violates BCNF since CS does not contain a key.


CTHRSG
key = {HS}
C→T, CS→G, HR→C, HS→R, TH→S
CS→G
CSG
key = {CS}
CS→G
CTHRS
key = {HS}
C→T, TH→R, HR→C, HS→R
CSG satisfies BCNF because the only possible dependency is CS → G, and CS is a key, thus it is in BCNF
CTHRS is not BCNF because the key is {HS} and the functional dependency C →T violates BCNF condition since C is not a key.
C→T
CT
key = {C}
C→T
CHRS
key = {HS}
CH→R, HR→C, HS→R
CH→R
CHR
key = {CH}, {HR}
CH→R, HR→C
CHS
key = {SH}
HS→C



Note, this BCNF decomposition did not preserve the dependency TH → R.

Lossless-Join Decomposition

- **Definition:** The lossless-join decomposition is another important property for the decomposition defined as follows*:*

- *A relation schema R has lossless-join decomposition R1, R2 if for every instance r of R we have:*
  - *r = (projection of r on R1) x (projection of r on R2), where x denotes join operation*

Example

R (branch_name, loan_name, customer_name, amount)

branch_name	loan_name	customer_name	amount
Downtown	17	Jones	1000
Redwood	23	Smith	2000
Perryridge	15	Hayes	1500
Downtown	14	Jackson	1500
Mianus	94	Curry	500

R1 ∩ R2 = {amount} does not imply
R1 – R2 = {branch_name, loan_name}

R1 ∩ R2 = {amount} does not imply
R2 – R1 = {cusomer_name}

We can state that the decomposition is lossy.

R1 (branch_name, loan_name, amount)

branch_name	loan_number	amount
Downtown	17	1000
Redwood	23	2000
Perryridge	15	1500
Downtown	14	1500
Mianus	94	500

R2 (amount, customer_number)

amount	customer_name
1000	Jones
2000	Smith
1500	Hayes
1500	Jackson
500	Curry

Denormalization

- Creation of normalized relations is an important database design goal.
- However, as tables are decomposed to conform to normalization requirements, the number of created database tables grows.
- Joining a larger number of tables takes additional input/output (I/O) operations, thereby reducing system speed.
- Thus, there is a tradeoff between good design and processing speed which can be resolved through denormalization.

Denormalization Example

Give the following attributes:

FullName, Street Number, StreetName, City, State, and ZipCode.

Obvoiusly we have the following dependencies:

FullName → StreetNumber,StreetName,City,State,ZipCode

ZipCode → City, State

Obviously the second dependency violates BCNF. Thus we decompose the attributes into two tables:

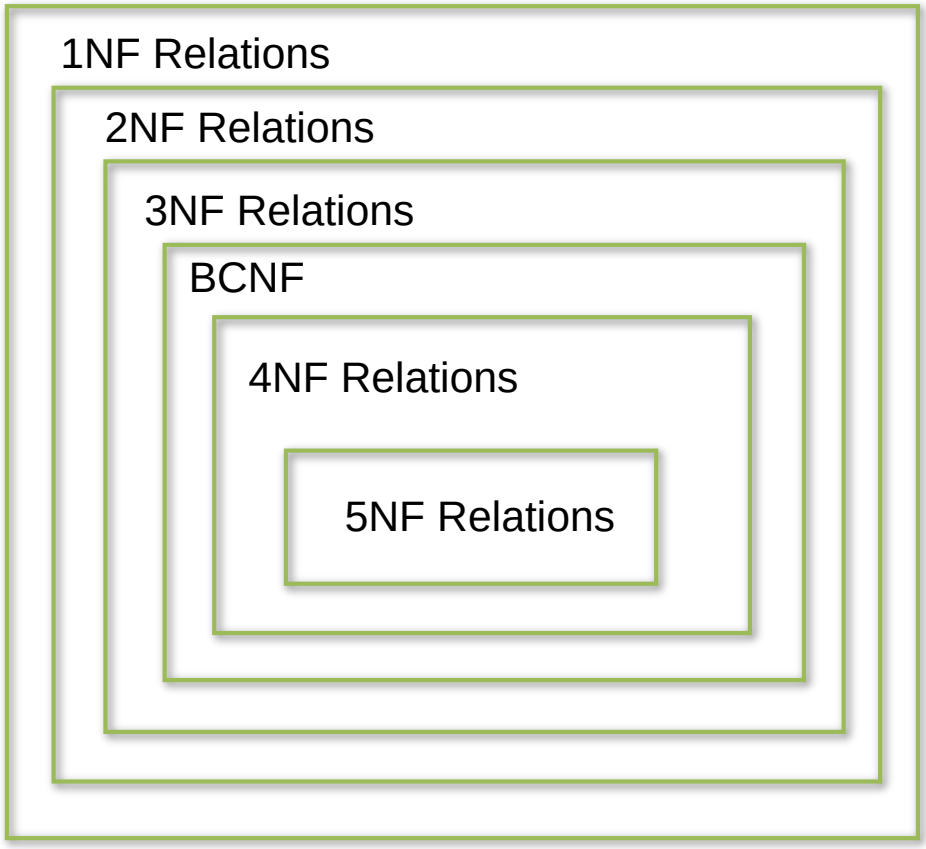
Table1:ZipCode,City,State

Table2: FullName,StreetNumber,StreetName,ZipCode

This mean every time we print the full name and address . We have to perform the join between the above two tables which is an expensive operation especially if it is frequently needed.

Higher Normal Forms

- BCNF is as far as we can go with FDs
- Higher normal forms are based on other sorts of dependency
- Fourth normal form removes multi-valued dependencies
- Fifth normal form removes join dependencies
- In practice 3NF and BCNF are used.



Summary

- Discussed anomalies (insertion/deletion/update) that can occur due to bad database design.
- Introduced the concept of ‘normalization’ to refine tables through decomposition to eliminate anomalies.
- Discussed ‘functional dependencies’
- Normal Forms

1NF	Attributes are atomic
2NF	1NF and no partial dependency
3NF	2NF and no transitive dependency
BCNF	Every determinant is a primary key

- Denormalization may be applied when application requirements require query speed-up and accepts the introduction of anomalies.