Lecture 13: Introduction to Normal Forms and Normalization

CSX3006 DATABASE SYSTEMS

ITX3006 DATABASE MANAGEMENT SYSTEMS

Outline

- Measure of Quality in DB Schema
- Data Redundancy and Functional Dependency
- Definition and Types of Functional Dependencies
- Definition of Keys in terms of Functional Dependencies
- Prime Attributes vs. Non-Prime Attributes
- 1st Normal Form
- 2nd Normal Form and Decomposition
- 3rd Normal Forms
- BCNF

Good Design? or Bad Design?

- Assume the following business rule;
  1. An employee can work for A single department; a department may have MANY employees
  2. For each department, there is A manager of the department;
     - The manager must be one of the employees
- Which of the following designs below is better for managing employee data? Why?

Design #1: all fields in one relation (table)

EmployeeData = (<u>eid</u>, name, birthdate, address, dept_id, dept_name, manager_id)

Design #2: Separate fields into 2 relations

Employee = (<u>eid</u>, name, birthdate, address, *dept_id*)

Department = (<u>dept_id</u>, dept_name, *manager_id*)

What's Wrong? - 1

Design #1: **EmployeeData = (<u>eid</u>, name, birthdate, address, dept_id, dept_name, manager_id)**

<u>eid</u>	name	birthdate	address	dept_id	dept_name	manager_id
6001	Tom	11/6/1995	121 Bellaire	1	Accounting	6001
6002	Mary	5/5/1987	4 Nutty	1	Accounting	6001
6003	Harry	3/9/1993	12 Downtown	2	IT	6003
6004	Sara	12/7/1994	15 Harrison	2	IT	6003
…	…	…				

- **Data Redundancy → Update Anomalies → Data Inconsistency**
  - **Waste of space; more importantly causes Update Anomalies**

What’s Wrong? - 2

Design #1: **EmployeeData = (<u>eid</u>, name, birthdate, address, dept_id, dept_name, manager_id)**

<u>eid</u>	name	birthdate	address	dept_id	dept_name	manager_id
6001	Tom	11/6/1995	121 Bellaire	1	Accounting	6001
6002	Mary	5/5/1987	4 Nutty	1	Accounting	6001
6003	Harry	3/9/1993	12 Downtown	2	IT	6003

- **Insertion Anomalies**
  - Insertion of a **new employee** *requires data **for*** **department attributes or nulls**
  - Difficult to **insert a new department** *that has **no employees** as yet*
- **Deletion Anomalies**
  - **Deleting the last employee** of a department → ***losing that department info.***
- **Modification Anomalies**
  - Changing the dept_name, or manager_id of a department → multiple changes

What's Wrong? - 3

Design #2:

Employee = (eid, name, birthdate, address, *dept_id*)

Department = (dept_id, dept_name, *manager_id*)

- NO problems of
  - Insertion Anomalies,
  - Deletion Anomalies,
  - Modification Anomalies

Good Design? or Bad Design? - 1

- Is breaking up into multiple relations always better?

employee = (eid)	**empname = (name)**	**manager = (eid)**	**department = (dept_name)**

Good Design? or Bad Design? - 2

- Which of the following designs below is better for managing employee data? Why?

Design #1:

Emp = (<u>eid</u>, name, birthdate, address, phone)

eid	name	birthdate	address	phone
1	Jane	11/05/75	Blah blah	222
2	Greg	19/02/73	Blah blah	312
3	Jane	31/08/74	Blah blah	434

Q: What's the phone number of Jane (eid 1)?

Design #2:

Employee = (<u>eid</u>, name)

Emp_info = (<u>name, birthdate</u>, address, phone)

eid	name
1	Jane
2	Greg
3	Jane

name	birthdate	address	phone
Jane	11/05/75	Blah blah	222
Greg	19/02/73	Blah blah	312
Jane	31/08/74	Blah blah	434

Joined table

eid	name	birthdate	address	phone
1	Jane	11/05/75	Blah blah	222
1	Jane	31/08/74	Blah blah	434
2	Greg	19/02/73	Blah blah	312
3	Jane	31/08/74	Blah blah	434
3	Jane	11/05/75	Blah blah	222

Things to Avoid
(lead to a bad design)

- **Data Redundancy** → Update Anomalies → Data Inconsistency
- **NULL values** (as much as possible)
- **Bad Splitting** → Lossy Joins → Spurious tuples

(Informal) Guidelines for Database design

1. DO NOT combine attributes *from multiple entity sets or relationship sets* into a single relation schema

2. Design in a way that NO insertion, deletion, modification anomalies will occur

3. Avoid NULL values AS MUCH AS POSSIBLE

4. Avoid Bad Splitting that could results in spurious tuples when joined

How do we ‘measure’ the goodness?

- By *following the design approach discussed in this course*, it tends to end up with *a generally good relational database schema;*
  - Avoid most of update anomalies and the problems of data redundancy
- However, we do not have any formal measure of determining why one good design is better than another one.
  - How can we formally argue and be **assured** that a set of relation schema we have designed is **free from data redundancy, update anomalies, null values, and lossy joins**?
  - How can we **formally assess the ‘quality’ or ‘goodness’** of different alternative designs in an objective and systematic ways?

Normal Forms

- Formal way to define and assess the 'quality' or 'goodness' of database schema
- Successive Levels of Normal Forms (in the order of 'goodness'):
  - e.g.) 1st NF → 2nd NF → 3rd NF → BCNF → 4th NF → 5th NF
  - Each level has *a set of constraints* designed to *minimize data redundancy*.
- The **higher** the **normal form** →

  the **less** vulnerable it is to data **inconsistency** and **anomalies** →

  the **better 'quality'** it represents

Normalization

- **A design technique** to *minimize data redundancy*
  - Prevents the data inconsistency *that could result in update anomalies*

- Process of testing if a relation schema *meets all the requirements* of a certain NF and if necessary transforming the schema *into higher NF*
  - By ***decomposing*** the relation schema ***into multiple relation schemas***

Functional Dependencies (FDs)

- **A set of constraints** that define each normal form's **requirements**
  - We use FDs to define Normal forms.

- **A constraint** between two sets of attributes
  - Require that the value for a certain set of attributes ***determines uniquely*** the value for another set of attributes.
    - e.g.) **X → Y**; (value of X ***determines uniquely*** the value of Y)

- Functional Dependency is based on semantics of data, BUSINESS LOGIC

Formal Definition of Functional Dependency - 1

- Let
  - R be a relation schema
  - $\alpha \subseteq R$
  - $\beta \subseteq R$
  - α and β are ***sets of attributes*** of the relation schema

- The functional dependency $\alpha \rightarrow \beta$ holds on *R if and only if*
  for any legal relation instances *r*(R),
  whenever any two tuples t_1 and t_2 of r agree on the attributes α,
  they also agree on the attributes β;

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

Formal Definition of Functional Dependency - 2

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

- The values of the α component of a tuple ***uniquely*** (or functionally) ***determine*** the values of the β component
  - In other words, the values of the β component of a tuple in *r* ***depend on*** the values of the α component
- There is a functional dependency from α to β
  - β is functionally dependent on α

Functional Dependency – 1
Remark

- A functional dependency is a property of the semantics (meaning) of the attributes
  - **Defined based on our understanding of the semantics of the attributes**
    - **Requires the understanding of the business logic and rules**

- A functional dependency is NOT a property of a particular relation instances
  - An FD **cannot be inferred automatically** from **given relation instances**
    - It may hold for a particular given instance, but can't be sure if the FD holds for all the legal values.
      - However, a single counter example is sufficient to disprove a FD

Functional Dependency – 2

Examples

- Consider R(A,B) with the following relation instances

A	B
1	4
2	5
3	7

1. What are the **functional dependencies** that hold?
   - Does A → B hold?
   - Does B → A hold?

A	B
1	4
3	7
3	9
1	5

2. What are the **functional dependencies** that hold?
   - Does A → B hold?
   - Does B → A hold?

Functional Dependency – 3

Trivial VS Non-trivial Functional Dependencies:

Trivial Functional Dependencies:

- **determinant** (LHS) is a **superset** of the **dependent** (RHS)
  - e.g.) X → X
  - XY → X
  - WXY → WX

- Consider R(A,B,C) with the following relation instances
  - What are the (non-trivial) functional dependencies satisfied by the relation instance below?

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c3

A → B

C → B

AC → B

Functional Dependency – 4
Example

- Consider the following relation schema

Student_grade = (student_id, name, course_id, course_title, grade)

- What are **the (non-trivial) functional dependencies** that **hold** in the schema?
  - NOTE: we **define the functional dependencies from our understanding of the meanings (semantics) of the attributes**, NOT from sample tuple instances

- **student_id → name**
- **course_id → course_title**
- **student_id, course_id → grade**

- **student_id, course_id → name**
- **student_id, course_id → course_title**

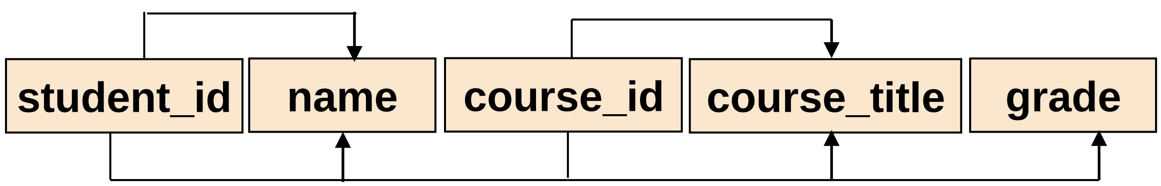
What should be the (primary) key of the schema?

- **student_id, course_id → name, course_title, grade**

Functional Dependency – 5
Example (Cont.)

Student_grade = (<u>student_id</u>, name, <u>course_id</u>, course_title, grade)

- **student_id → name**
- **course_id → course_title**
- **student_id, course_id → grade, name, course_title**



student_id	name	course_id	course_title	grade
4561234	James	SC3456	OS	A
4561234	James	SC4567	Compiler Design	B
4526322	Paula	SC3456	OS	C

Functional Dependency – 6

A generalization of notion of the key

- Allows the mapping constraints to be specified between
  1. key attributes and non-key attributes, and/or
  2. non-key attributes

Example 1:

Student_grade = (<u>student_id</u>, name, <u>course_id</u>, course_title, grade)

- **student_id, course_id → name, course_title, grade**

Example 2:

Technician = (<u>staff_id</u>, name, skill_level, hourly_wage, phone)

- **staff_id → name, skill_level, hourly_wage, phone**

Also allows us to express constraints that cannot be expressed using keys

- **student_id → name**
- **course_id → course_title**

- **skill_level → hourly_wage**

Types of Functional Dependency

1. Trivial Functional Dependency
2. Full Functional Dependency
3. Transitive Functional Dependency

1. Trivial Functional Dependency

- Determinant (LHS) is a superset of the dependent (RHS)
  - Example 1: X → X
    - grade → grade
  - Example 2: XY → X
    - student_id, name → name

2. Full Functional Dependency

- A set of attributes, Y, is *fully functionally dependent on* a set of attributes, X, ***if***

1. Y is **functionally dependent on** X, and
2. Y is **NOT functionally dependent on any of *proper subset*** of X.

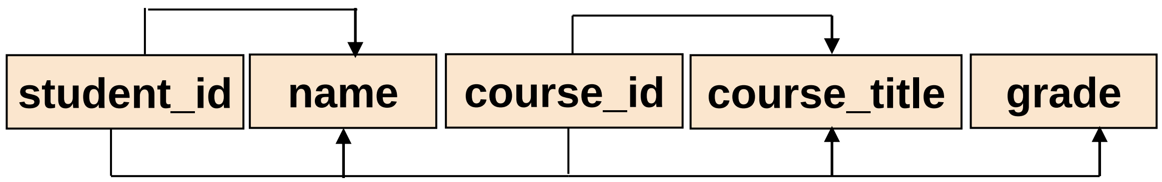
- E.g.,
  - student_id, course_id → grade (**full functional dependency**)
  - student_id, course_id → name (**partial functional dependency**)
    - since student_id → name

3. Transitive Functional Dependency

- IF there exist X → Y, and Y → Z THEN **X → Z**
  - The FD: **X → Z** is a **transitive** FD

- Examples,
  - employee_id → email_address
  - email_address → home_page_URL
  - **employee_id → home_page_URL**

FDs and Keys

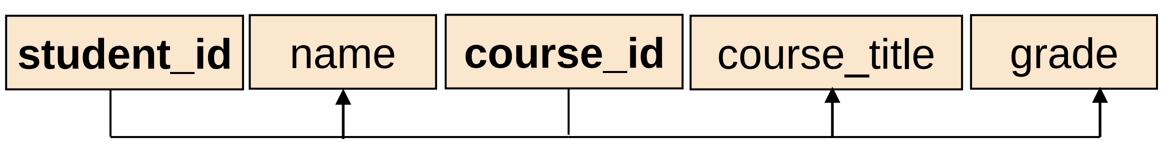


- What should be the **key** of the relation schema above? Why?

Association between FDs and Keys:
A superkey

- A superkey: a set of one or more attributes that, taken collectively, *allow us to identify uniquely a tuple in the relation*.

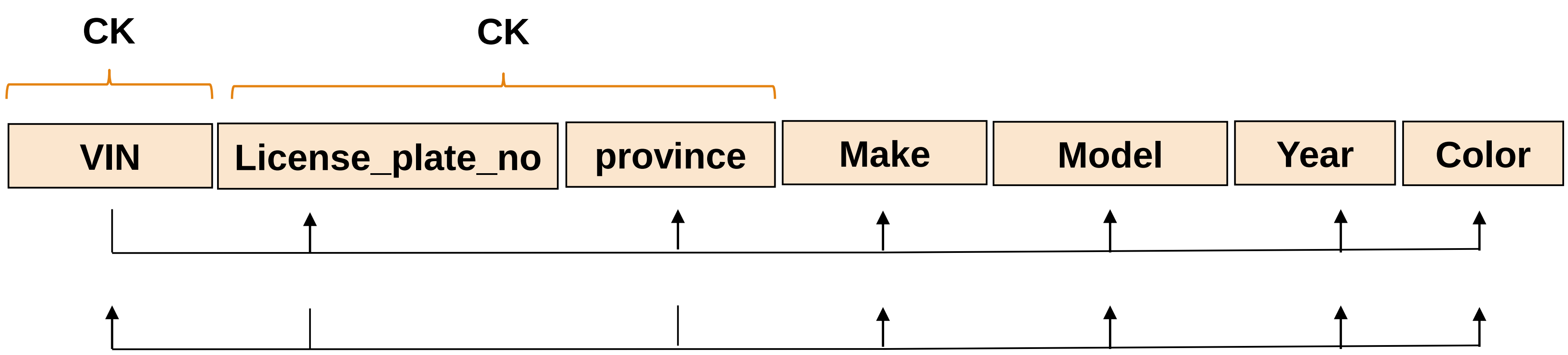
- There is an **FD from** the **super key to EVERY other attribute** in the schema



Association between FDs and Keys:
A candidate key (CK)

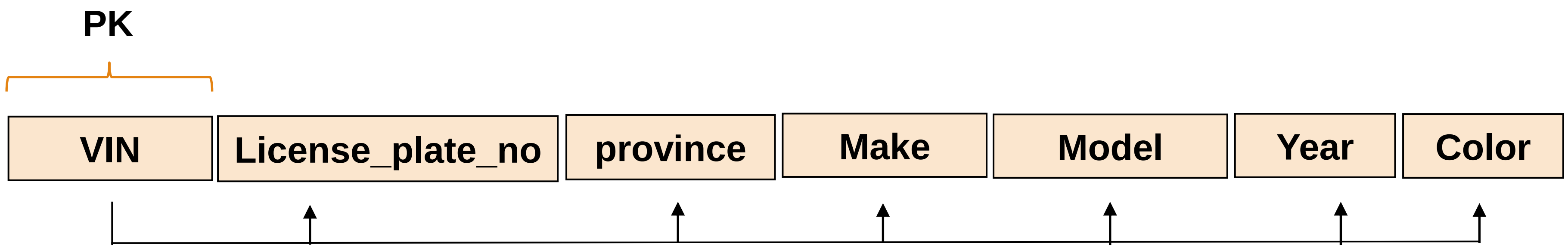
- *K* is a **candidate key** if *K* is a ***minimal*** superkey
  - No proper subset of K is a superkey

- There is **Full FD from** the **candidate key to EVERY other attribute** in the schema



Association between FDs and Keys:
A primary key (PK)

- A candidate key ***chosen** as the principal means of identifying tuples within a relation*
- There is **Full FD from** the **primary key to EVERY other attribute** in the schema



Recap: Functional Dependency - 1

- **What is a functional dependency?**
  - **A constraint specified between two sets of attributes in a relation**
    - **A → B**; values of A *functionally determines* values of B
  - A generalization of the notion of the keys
    - **Allows to express constraints that cannot be expressed using only keys**

Student_grade = (student_id, name, course_id, course_title, grade)

- **student_id → name**
- **course_id → course_title**

Technician = (staff_id, name, skill_level, hourly_wage, phone)

- **skill_level → hourly_wage**

Recap: Functional Dependency - 2

- **What is a functional dependency?**
  - **A property of the semantics of the attributes, not of a particular relation instances**
    - Cannot automatically infer from given relation instances
    - **Defined from business rules and logic (based on the meanings of attributes)**
  - **A formal way of defining the constraints that must be satisfied in each normal form**
    - Normal Forms are defined in terms of Functional Dependencies.

- **The types of Functional Dependencies:**
  - Trivial vs. Non-Trivial
  - Full vs. Partial
  - Transitive

Normal Forms and Normalization

- **1st NF**
  - **Every attribute has atomic values** ;
    - therefore, **every legal relation** is already in 1st NF;
- **2nd NF**
  - 1st NF and **every non-prime attribute is fully functionally dependent on every candidate key**
- **3rd NF**
  - 2nd NF and every non-prime attribute is non-transitively dependent on every candidate key
- **BCNF (Boyce-Codd Normal Form)**
  - 3rd NF and every determinant is a super key
- **4th NF**
  - Based on the concept of multi-valued dependency
- **5th NF (PJ/NF)**
  - Based on the concept of Join Dependency; hence the alias Project Join Normal Form

Normal Forms and Normalization

- Codd originally defined the first three normal forms (1^{st}, 2^{nd}, and 3^{rd})
- Boyce and Codd later identified a redundancy that can happen is certain cases of the 3^{rd} NF, and made a stronger version called, BCNF, to counter those cases.

Every non-key attribute* must *be functionally dependent

upon the key ← **1^{st} NF + requirements of Key**

the whole key ← **2^{nd} NF**

and nothing but the key ← **3^{rd} NF**

Terminology Reminder

- **Superkey**
- **Candidiate key**
- **Primary key**
- **Prime Attribute**
  - **An attribute which is *a part of any candidate* key**
- **Non-Prime Attribute**
  - **An attribute which is *not* a prime attribute;**
    - ***Not a part of any candidate keys***
- **Trivial Functional Dependency**
- **Full Functional Dependency**
- **Partial Functional Dependency**
- **Transitive Functional Dependency**

1^{st} NF - 1

- **Domains of attributes must include *only atomic* (simple, indivisible) values** and the value of any attribute in a tuple must be a *single value* from the domain
  - By definition, **every legal relation is already in 1^{st} NF.**
    - Since a legal relation only allows for atomic values in its attributes

1st NF - 2

dept_id	dept_name	manager_id	locations
1	Finance	4123123	Bangkok, Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok, Chiangmai, Phuket

dept_id	dept_name	manager_id	**location**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

2^{nd} NF - 1

- **Every *non-prime attribute* is fully functionally dependent on every candidate key**
  - In other words, every non-prime attribute is NOT partially dependent on **any** candidate key

  - **NOTE: IF there is only one candidate key and it is a single attribute (not composite key)**, THEN the schema is *in 2^{nd} NF by virtue of the definition*.

- **Prime Attribute**
  - **An attribute which is *a part of any candidate* key**
- **Non-Prime Attribute**
  - **An attribute which is *not* a prime attribute;**
    - ***Not a part of any candidate keys***

2^{nd} NF - 2

dept_id	dept_name	manager_id	**location**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- What are the functional dependencies?

2^{nd} NF - 3

dept_id	dept_name	manager_id	**location**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- What are the functional dependencies?
  - **dept_id, location → dept_name, manager_id**
  - **dept_id → dept_name, manager_id**
  - dept_name → manager_id (maybe? Need to confirm from business logic)
  - manager_id → dept_name, dept_id (maybe? Again confirmation required)

2^{nd} NF - 4

dept_id	dept_name	manager_id	**location**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- **Assume only** the followings are functional dependencies that hold in the schema
  - **dept_id, location → dept_name, manager_id**
  - **dept_id → dept_name, manager_id**
- What are the ***candidate keys*** in the schema?

2nd NF - 5

dept_id	dept_name	manager_id	**location**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- **Assume only** the followings are functional dependencies that hold in the schema
  - **dept_id, location → dept_name, manager_id**
  - **dept_id → dept_name, manager_id**
- What are the ***candidate keys*** in the schema?
  - (dept_id, locations)
- Is there any other candidate keys?

2nd NF - 6

<u>dept_id</u>	dept_name	manager_id	**<u>location</u>**
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- **Assume only** the followings are functional dependencies that hold in the schema
  - **dept_id, location → dept_name, manager_id**
  - **dept_id → dept_name, manager_id**
- What is (are) the ***candidate key(s)*** in the schema?
  - (dept_id, locations)
- Is there any other candidate key(s)? No

2nd NF - 7

- **Every non-prime attribute is <u>fully functionally dependent</u> on every candidate key**
  - **Every non-prime attribute is NOT partially dependent on any candidate key**

Department = (<u>dept_id</u>, dept_name, manager_id, <u>location</u>)

- **Only candidate key and the Primary key : (dept_id, location)**
  - **Prime attributes**; dept_id, location
  - **Non-prime attributes**: dept_name, manager_id
- **FDs that hold**
  - **dept_id, location → dept_name, manager_id**
    - This is obvious since (dept_id, location) is a candidate and primary key
  - **dept_id → dept_name, manager_id**
- **Is the schema in 2nd NF given the functional dependencies above?**
  - dept_name, manager_id are ***partially dependent on*** (dept_id, location)
    - Since dept_id → dept_name, manager_id

2nd NF – 8 (Normalization)

- **What are the *problems of not meeting 2nd NF requirements*?**
  - **Data Redundancy** → Update Anomalies → Data Inconsistency
    - dept_name, manager_id are ***repeated***

<u>dept_id</u>	dept_name	manager_id	<u>location</u>
1	Finance	4123123	Bangkok
1	Finance	4123123	Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok
3	Marketing	6234233	Chiangmai
3	Marketing	6234233	Phuket

- **How do we transform the schema that adheres to 2nd NF? (Normalization)**

Department = (<u>dept_id</u>, dept_name, manager_id, <u>location</u>)

dept_id → dept_name, manager_id

Decompose the relation by

Creating a new relation from the FD that causes Partial FD;

- PK is the LHS of the FD

department = (<u>dept_id</u>, dept_name, manager_id)

dept_locations = (<u>dept_id, location</u>)

2^{nd} NF - 9

dept_id	dept_name	manager_id	locations
1	Finance	4123123	Bangkok, Phuket
2	R&D	3562342	Bangkok
3	Marketing	6234233	Bangkok, Chiangmai, Phuket

- **Note that we can directly transform above (illegal) relation into 2^{nd} NF without going through the 1^{st} NF first.**
  - **dept_id is the primary key but there are multiple locations**
    - So, *split* the *non-atomic attribute into* its own table together with the primary key

dept_id	dept_name	manager_id
1	Finance	4123123
2	R&D	3562342
3	Marketing	6234233

dept_id	**location**
1	Bangkok
1	Phuket
2	Bangkok
3	Bangkok
3	Chiangmai
3	Phuket

Exercise on 2nd NF - 1

Employee_project = (<u>e_id, p_number</u>, e_name, p_title, p_budget, hours_worked)

- **e_id, p_number → hours_worked, e_name, p_budget, p_title**
- **e_id → e_name**
- **p_number → p_title, p_budget**

- **Is the schema above with the given functional dependency in 2nd NF?**
  - **Identify *partial Functional Dependencies***
    1. Focus on **non-prime attribute** on the **RHS** of FD
    2. Find 2 or more FDs with the **same RHS attributes** and
    3. **Subset-superset relationship** on the **LHS** attributes (of the same FDs in 2)

- **Transform into 2nd NF by decomposing**

Exercise on 2nd NF - 2

Employee_project = (e_id, p_number, e_name, p_title, p_budget, hours_worked)

- e_id, p_number → hours_worked, e_name, p_budget, p_title
- e_id → e_name
- p_number → p_title, p_budget

- **Is the schema above with the given functional dependency in 2nd NF?**
  - Identify the FD(s) that causes *partial Functional Dependencies*
    - e_id → e_name
    - p_number → p_title, p_budget
- **Transform into 2nd NF by decomposing**

employee = (e_id, e_name)

project = (p_number, p_title, p_budget)

Employee_project = (e_id, p_number, hours_worked)

Exercise on 2^{nd} NF - 3

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

- Assume that there are no other functional dependencies apart from the obvious ones from the primary key to other attributes
- **Is the schema in 2^{nd} NF?**
  - If there is only one candidate key and it is a single attribute (not composite key), and no other FDs exist, then the schema is in 2^{nd} NF by virtue of the definition.

Exercise on 2^{nd} NF - 4

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

title, year, studio_id → movie_id, studio_name

- (title, year, studio_id) is another candidate key
- Still in 2^{nd} NF. WHY?
  - The only **non-prime attribute**, studio_name, **fully depends on both candidate keys**
    - movie_id → studio_name
    - title, year, studio_id → studio_name

Exercise on 2^{nd} NF - 5

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

title, year, studio_id → movie_id, studio_name

studio_id → studio_name

- Still in 2^{nd} NF?

Exercise on 2nd NF - 6

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

title, year, studio_id → movie_id, studio_name

studio_id → studio_name

- Still in 2nd NF?
  - Now the non-prime attribute, studio_name, is ***partially dependent*** *on a candidate key*
    - Transform into 2nd NF by decomposing

Studio = (*<u>studio_id</u>*, studio_name)

Movie = (<u>movie_id</u>, title, year, studio_id)

Need for 3rd NF - 1

Movie = (movie_id, title, year, studio_id, studio_name)

studio_id → studio_name

- Consider the above schema and **assume that there are no other functional dependency** than the one specified; **Is the relation in 2nd NF?**
  - **Yes; There is NO PARTIAL functional dependency**
  - **2nd NF doesn't say anything about non-prime attribute depending on another non-prime attributes!** (Will restrict it in the 3rd NF.)
- **We have verified that above table confirms to 2nd NF.**
  - **But, is the table free from Data Redundancy?**
- **2nd NF still can suffer from**
  - **Data Redundancy** → Update Anomalies → Data Inconsistency
    - **What's the problem?**

Need for 3rd NF - 2

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

studio_id → studio_name

<u>movie_id</u>	title	year	studio_id	Studio_name
14	Gone with the wind	1973	2	Universal Pictures
22	God Father	1979	5	Warner Brothers
28	Scar Face	1986	5	Warner Brothers
32	Forest Gump	1989	2	Universal Pictures

- **2nd NF still can suffer from**
  - **Data Redundancy** → Update Anomalies → Data Inconsistency
  - **What's the problem?**
    - Transitive Dependency!
- movie_id → studio_name
  - studio_name is transitively dependent on movie_id
    - movie_id → studio_id; studio_id → studio_name

3rd NF - 1

- A relation schema is in 3rd NF ***if*** it is in 2nd NF and **every non-prime attribute is NON-transitively dependent *(i.e. directly dependent)* on *every candidate key***

- **Equivalent Definition for 3rd NF**
  - **Whenever X → Y holds in R, *either***
    - **X is a superkey of R, *or***
    - **Y is a prime attribute of R**
  - If **BOTH** conditions are **NOT satisfied** for any FD in R, then R is **not in 3rd NF**.

- **Prime Attribute**
  - **An attribute which is *a part of any candidate*** key
- **Non-Prime Attribute**
  - **An attribute which is *not* a prime attribute;**
    - ***Not a part of any candidate keys***

3^{rd} NF - 2

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

studio_id → studio_name

studio_name is transitively dependent on movie_id:
movie_id → studio_id; studio_id → **studio_name**

Transitive FD: **movie_id → studio_name**

- If ***BOTH*** conditions are ***NOT* satisfied** for any FD in R, then R is ***not* in 3^{rd} NF**.
  - Given **studio_id → studio_name**
    - **Cond. 1: Is studio_id a superkey of Movie?** ✗
    - **Cond. 2: Is studio_name a prime attribute of Movie?** ✗
  - Therefore, the **FD: studio_id → studio_name violates the rule of 3^{rd} NF**

3^{rd} NF - 3

- **If there is only one candidate key and the key is not composite (single attribute) in R, then the definition of the 3^{rd} NF can be simplified as**
  - No non-prime attribute determines another non-prime attribute

- How to transform into a 3^{rd} NF?

Movie = (movie_id, title, year, studio_id, studio_name)

studio_id → studio_name

Studio = (studio_id, studio_name)

Movie = (movie_id, title, year, studio_id)

3rd NF - 4

- We can also transform directly into 3rd NF WITHOUT going through 2nd NF evaluation

Employee_project = (<u>e_id, p_number</u>, e_name, p_title, p_budget, hours_worked)

e_id, p_number → hours_worked ✔

e_id → e_name ✘

p_number → p_title, p_budget ✘

- We can evaluate the schema directly in terms of 3rd NF, without first considering if it meets the 2nd NF requirements

employee = (<u>e_id,</u> e_name)

project = (<u>p_number,</u> p_title, p_budget)

Employee_project = (<u>e_id, p_number</u>, hours_worked)

Exercise on 3rd NF - 1

Movie = (movie_id, title, year, studio_id, studio_name)

studio_id → studio_name

- Is the schema in 3rd NF? (Notice the second candidate key)

Exercise on 3rd NF - 2

Movie = (movie_id, title, year, studio_id, studio_name)

studio_id → studio_name

- Is the schema in 3rd NF? (Notice the second candidate key)
  - **No**

- **Why?**
  - **studio_id** is **NOT a superkey of Movie**
    - studio_id is a prime attribute, **but NOT a key attribute**
  - **studio_name** is **NOT a prime attribute of Movie**
    - **studio_name** is a **non-prime attribute** (and only one which is non-prime)

Exercise on 3rd NF - 3

- How about the schema below? Is the schema in 3rd NF?

Reviewer = (reviewer id, name, email, city, affiliation)

Assume no other FDs

reviewer_id → name, email, city, affiliation

email→ reviewer_id, name, city, affiliation

- Any partial FD? Any Transitive FD?

Exercise on 3rd NF - 4

- How about the schema below? Is the schema in 3rd NF?

Reviewer = (reviewer id, name, email, city, affiliation)

Assume no other FDs

review_id → name, email, city, affiliation

email→ review_id, name, city, affiliation

- Any partial FD? Any Transitive FD?

No

Yes

reviewer_id → name, city, affiliation

because

reviewer_id → email

email →name, city, affiliation

Exercise on 3^{rd} NF - 5

Reviewer = (reviewer id, name, email, city, affiliation)

reviewer_id → city, affiliation

Transitive functional dependency through

reviewer_id → email

email → city, affiliation

- Is the schema above in 3^{rd} NF? WHY?

Exercise on 3rd NF - 6

Reviewer = (reviewer id, name, email, city, affiliation)

reviewer_id → city, affiliation

Transitive functional dependency through

reviewer_id → email

email → city, affiliation

?

- Is the schema above in 3rd NF?
  - **YES**
- WHY?
  - Because the transitivity is through another candidate key (email)
    - **reviewer_id ←→ email; reviewer_id → All Others; email → All Others**
    - Alternatively, the **FD: email → city, affiliation** satisfies the first requirement saying the left hand side is a superkey.

Exercise on 3rd NF - 7

- Is the relation schema below with the given FDs in 3rd NF?

departmentKPIonProject = (department_id, project_id, score, dept_leader_emp_id)

- **Performance of each department on each project it has participated**
  - Realize that the *primary key implies that M-M between department and project*.
  - **dept_leader_emp_id** is the employee id who worked on the project as the group leader of the department
- The **business rule says an employee can work for only one department**;
  - So, the following FD holds on the relation,

dept_leader_emp_id → department_id

- **Is the schema with the given FD in 3rd NF? WHY?**

Exercise on 3rd NF - 8

payReport = (<u>e_id</u>, e_name, <u>p_number</u>, p_title, p_budget, hours_worked, skill_level, hourly_rate)

e_id, p_number → hours_worked
e_id → e_name, skill_level
p_number → p_title, p_budget
skill_level → hourly_rate

- **Is the schema in 3rd NF?**
- **If not, transform into 3rd NF.**

Exercise on 3rd NF - 9

payReport = (e_id, e_name, p_number, p_title, p_budget, hours_worked, skill_level, hourly_rate)

e_id, p_number → hours_worked
e_id → e_name, skill_level
p_number → p_title, p_budget
skill_level → hourly_rate

- Is the schema in 3rd NF?
- If not, transform into 3rd NF.

employee = (e_id, e_name, skill_level)

project = (p_number, p_title, p_budget)

payrate = (skill_level, hourly_rate)

payReport = (e_id, p_number, hours_worked)

- Notice that the original schema was only in 1st NF.
  - It's not even in 2nd NF since it has partial functional dependencies
- Notice that it's NOT necessary to go through from 1st → 2nd → 3rd
  - We can directly test a schema to see if it complies with 3rd NF
    - And can transform directly to 3rd NF if it is not without going through 2nd NF

Lossless Decomposition - 1

- The **purpose** of **Normalization** is to *eliminate data redundancy* and the possible *data inconsistency* caused by update anomalies.

- Each **normal form** is the *standard measure of 'likelihood' for data redundancy'*
  - **Higher** the normal form, the **less chance** for data redundancy
- Normalization is done by
  - **Decomposing** the initial schema **into multiple schemas**

- How to ensure to get the same set of data that was available in the original schema from the decomposed schemas?

Lossless Decomposition - 2

Emp = (<u>eid</u>, name, birthdate, address, phone)

eid	name	birthdate	address	phone
1	Jane	11/05/75	Blah blah	222
2	Greg	19/02/73	Blah blah	312
3	Jane	31/08/74	Blah blah	434

Employee = (<u>eid</u>, name)

Emp_info = (<u>name, birthdate</u>, address, phone)

eid	name
1	Jane
2	Greg
3	Jane

name	birthdate	address	phone
Jane	11/05/75	Blah blah	222
Greg	19/02/73	Blah blah	312
Jane	31/08/74	Blah blah	434

eid	name	birthdate	address	phone
1	Jane	11/05/75	Blah blah	222
1	Jane	31/08/74	Blah blah	434
2	Greg	19/02/73	Blah blah	312
3	Jane	31/08/74	Blah blah	434
3	Jane	11/05/75	Blah blah	222

- **Lossy Decomposition** problem:
  - Because '**name**' is ***not a key*** of the emp_info.
    - **There can be many tuples with the same name**
    - **When** it is **joined** with employee relation on the '**name**' attribute, it will **generate spurious tuples.**

Lossless Decomposition - 3

- How to ensure '**lossless decomposition**' (avoid lossy decomposition)?
  - ***Decomposition*** of R into R1 and R2 ***is lossless if***
    - R1 ∩ R2 ***forms a superkey*** of either R1 or R2

Movie = (<u>movie_id</u>, title, year, studio_id, studio_name)

studio_id → studio_name

Movie = (<u>movie_id</u>, title, year, studio_id)

Studio = (<u>studio_id</u>, studio_name)

- **Movie ∩ Studio is studio_id** and the **studio_id is a superkey** *(in fact primary key)* of **Studio**.
  - Therefore *the decomposition* is the *lossless*.

Lossless decomposition - 4

- If a decomposition of a relation schema into 2nd and 3rd NFs is based on FDs that hold on the schema, then the decomposition is ***guaranteed to be lossless*!**

Recap: Normalization and Normal Forms

<u>Every non-prime attribute</u> must be *functionally dependent*

upon the key

the whole key

and nothing but the key

1st NF + requires a relation to have a key so that each tuple can be uniquely identified

2nd NF: *No partial* functional dependency

3rd NF: *No transitive* functional dependency

Need for BCNF - 1

Supplier_Part = (<u>supplier_id</u>, supplier_name, <u>part_no</u>, quantity)

- The relation **keep track of parts and the quantities of the parts supplied by a particular supplier**
  - **Assume that no two suppliers' names are the same.**
- **What are the *functional dependencies* that hold on the relation?**

supplier_id, part_no → quantity, supplier_name

supplier_id → supplier_name

supplier_name → supplier_id

supplier_name, part_no → quantity, supplier_id

Supplier_Part = (<u>supplier_id</u>, supplier_name, <u>part_no</u>, quantity)

- **It shows that we have *two* candidate keys**
  - **(supplier_id, part_no)** (chosen as the primary key)
  - **(supplier_name, part_no)**
- **Is the schema in 3rd NF?**

Need for BCNF - 2

Supplier_Part = (supplier_id, supplier_name, part_no, quantity)

supplier_id, part_no → quantity, supplier_name
supplier_id → supplier_name
supplier_name → supplier_id
supplier_name, part_no → quantity, supplier_id

- Is it 3rd NF?
  - Every **non-prime attribute** is *fully dependent* on every candidate key (2NF)
  - Every **non-prime attribute** is *non-transitively* dependent on every candidate key (3NF)
    - For every FD, either one of the following two must be true:
      - **Left hand side** is a **superkey**
      - **Right hand side** is a **prime attribute**
- **So the above schema is in 3rd NF** (therefore also in 2nd and 1st NF)
- Are we free of any *data redundancy* and *update anomalies*?

3rd NF, but NOT BCNF - 1

Supplier_Part = (supplier_id, supplier_name, part_no, quantity)

supplier_id, part_no → quantity, supplier_name

supplier_id → supplier_name

supplier_name → supplier_id

supplier_name, part_no → quantity, supplier_id

supplier_id	**supplier_name**	part_no	quantity
101	**A**	P001	500
101	**A**	P002	100
102	**B**	P001	200
102	**B**	P002	300

- Where does the **problem of update anomalies** stem from?
- **Grouping** of attributes of **supplier** together with attributes of **parts**
  - Fail to realize that they *represent two different entities*
- **Why hasn't this design fault detected by 3rd NF?**
  - 3rd NF ensures that **non-prime attributes** are ***dependent only on a whole key***, but nothing else. (no other non-key attributes)
  - **But, it doesn't say anything about functional dependencies among prime attributes** (that are not key).
    - e.g.) supplier_id and supplier_name are *prime attributes of two different keys*, but then have a *functional dependency* defined between them

BCNF - 1

- BCNF handles certain (rather rare) situations *which 3rd NF cannot handle*.
- The problem of 3rd NF, but not BCNF can only happen in a relation that
  - **Has multiple candidate keys** (at least two)
  - **Those candidate keys are composite** (consists of multiple prime attributes)
  - **The candidate keys overlap** (i.e. share at least one common attribute)
    - A prime attribute (but not a key by itself) functionally determines another prime attribute

BCNF - 2

- If a relation contains *only one candidate key*, 3rd NF and BCNF *are equivalent*
- A relation is in BCNF *if and only if* ***every determinant (LHS) is a candidate key***
  - For every FD, the left side of the FD must be a key
    - The second condition of 3rd NF is removed.

BCNF Example - 1

Supplier_Part = (supplier_id, supplier_name, part_no, quantity)

supplier_id, part_no → quantity, supplier_name

supplier_name, part_no → quantity, supplier_id

supplier_id → supplier_name

supplier_name → supplier_id

- Is the schema above with the given FDs in BCNF?

BCNF Example - 2

Supplier_Part = (<u>supplier_id</u>, supplier_name, <u>part_no</u>, quantity)

supplier_id, part_no → quantity, supplier_name

supplier_name, part_no → quantity, supplier_id

supplier_id → supplier_name

supplier_name → supplier_id

- Is the schema above with the given FDs in BCNF?
  - No, a relation is in BCNF if and only if *every determinant is a super key*
    - For every FD, ***the left side of the FD must be a candidate key***
      - *not just a part of the key (being prime attribute)*

BCNF Example - 3

Supplier_Part = (<u>supplier_id</u>, supplier_name, <u>part_no</u>, quantity)

supplier_id, part_no → quantity, supplier_name

supplier_name, part_no → quantity, supplier_id

supplier_id → supplier_name

supplier_name → supplier_id

- How do we decompose into BCNF compliant schemas?

BCNF Example - 4

Supplier_Part = (<u>supplier_id</u>, supplier_name, <u>part_no</u>, quantity)

supplier_id, part_no → quantity, supplier_name

supplier_name, part_no → quantity, supplier_id

supplier_id → supplier_name

supplier_name → supplier_id

supplier_id	**supplier_name**	part_no	quantity
101	**A**	P001	500
101	**A**	P002	100
102	**B**	P001	200
102	**B**	P002	300

- How do we decompose into BCNF compliant schemas?

Supplier = (<u>supplier_id</u>, supplier_name)

Parts = (<u>supplier_id, part_no</u>, quantity)

The update anomalies is now removed.

supplier_id	part_no	quantity
101	P001	500
101	P002	100
102	P001	200
102	P002	300

supplier_id	supplier_name
101	A
102	B

Exercise#1 on BCNF - 1

departmentKPIonProject = (<u>department_id, project_id</u>, score, dept_leader_emp_id)

Performance of each department on each project it has participated

- Realize that the primary key implies that M-M between department and project.
- dept_leader_emp_id is the id of the employee who worked on the project as the group leader of the department

The **business rule says an employee can work for only one department**; Therefore, the following FD hold on the relation

dept_leader_emp_id → department_id

- Is the schema with the given FD in 3rd NF? If yes, WHY?
- Is the scheme BCNF? (If not, decompose it into BCNF)

Exercise#1 on BCNF - 2

- Is the schema with the given FD in 3rd NF? ← **Yes**
- WHY?
  - (department_id, project_id) is a candidate key
  - dept_leader_emp_id → department_id
  - Therefore (dept_leader_emp_id, project_id) is another candidate key
  - Only non-prime attribute is “score”
  - And score depends fully and directly on the two candidate keys
- Is the scheme BCNF? (If not, decompose it into BCNF) ← **No**

departmentKPIonProject = (<u>dept_leader_emp_id, project_id</u>, score)

leaderDept = (<u>dept_leader_emp_id</u>, department_id)

Exercise#2 on BCNF

- We can transform directly into BCNF without going through 2^{nd} and 3^{rd} NF first

payReport = (e_id, e_name, p_number, p_title, p_budget, hours_worked, skill_level, hourly_rate)

e_id, p_number → hours_worked
e_id → e_name, skill_level
p_number → p_title, p_budget
skill_level → hourly_rate

- **Is the schema in BCNF?**
- **What are FDs violating the BCNF rule?**

- A relation is in BCNF if and only if every determinant is a candidate key
  - **For every FD, the left side of the FD must be a candidate key**, not just a prime attribute.

e_id → e_name, skill_level
p_number → p_title, p_budget
skill_level → hourly_rate

employee = (e_id, e_name, skill_level)

payrate = (skill_level, hourly_rate)

project = (p_number, p_title, p_budget)

payReport = (e_id, p_number, hours_worked)

- **Confirm that all resulting schemas are in BCNF**

Exercise#3 on BCNF

Lending = (<u>customer_id, loan_number</u>, amount, branch_name, branch_city, assets)

branch_name → branch_city, assets

loan_number → amount, branch_name

- Is the relation in BCNF? Why NOT? (Is it in 3NF?, is it in 2NF?)
- Decompose into a set of schemas, each of which complies with BCNF

branch_name → branch_city, assets

branch = (<u>branch_name</u>, branch_city, assets) **in BCNF?**

loan_number → amount, branch_name

loan = (<u>loan_number</u>, amount, branch_name) **in BCNF?**

Lending = (<u>customer_id, loan_number</u>) **in BCNF?**