Principles of Database Systems (CS307)

Lecture 10: Normalization - Part 1

Zhong-Qiu Wang

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result
  - Form the basis for the widely used SQL query language
  - Can express SQL statements in a mathematical way
- 6 basic operators:
  - select: σ
  - project: $\prod$
  - rename: ρ
  - union: $\cup$
  - set difference: $-$
  - Cartesian product: x
- Unary (operating on one relation) and binary (operating on pairs of relations)

Select Operation

- The select operation selects tuples that satisfy a given predicate (谓词)
  - Notation: $\sigma_p(r)$
  - p is called the selection predicate （选择谓词）
  - Lowercase Greek letter - σ
- Example
  - Select those tuples of the *instructor* relation where the instructor is in the “Physics” department

$$\sigma_{dept\_name = \text{“Physics”}}(instructor)$$

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Select Operation

- We allow comparisons using $=, \neq, >, \geq. <. \leq$ in the selection predicate
- We can combine several predicates into a larger predicate by using the connectives:

$$\wedge \textbf{(and)}, \vee \textbf{(or)}, \neg \textbf{(not)}$$

  - E.g., find the instructors in Physics with a salary greater $90,000, we write:

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90,000} (instructor)$$

- The select predicate may include comparisons between two attributes
  - E.g., find all departments whose name is the same as their building name:

$$\sigma_{dept\_name=building} (department)$$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out

Notation: $\prod_{A_1, A_2, A_3 \ldots A_k} (r)$

where A_1, A_2, …, A_k are attribute names, r is a relation name, and $\prod$ is uppercase Greek letter pi

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Composition of Relational Operations

- The result of a relational-algebra operation is relation
  - … and therefore, relational-algebra operations can be composed together into a relational-algebra expression

- Example: Find the names of all instructors in the Physics department

$$\Pi_{name}(\sigma_{dept\_name = \text{“Physics”}}(instructor))$$

  - Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation

Cartesian-Product Operation

- The Cartesian-product operation (denoted by a cross, ×) allows us to combine information from any two relations
  - E.g., the Cartesian product of the relations instructor and teaches is written as:

$$\text{instructor} \times \text{teaches}$$

- We construct a tuple of the result out of each possible pair of tuples
  - … one from the instructor relation and one from the teaches relation (see next slide)
  - Since the instructor ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came
    - instructor.ID and teaches.ID
  - Relation schema of $\text{instructor} \times \text{teaches}$
    - (instructor.ID, instructor.name, instructor.dept name, instructor.salary, teaches.ID, teaches.course id, teaches.sec id, teaches.semester, teaches.year)
    - (instructor.ID, name, dept name, salary, teaches.ID, course id, sec id, semester, year)

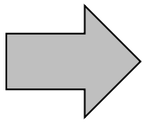
Can remove the prefix if not producing ambiguity

The "instructor × teaches" table

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

×

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4



instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Join Operation

- To get only those tuples of "instructor × teaches" that pertain to instructors and the courses that they taught, we write:

$$\sigma_{instructor.id = teaches.id}\ (instructor \times teaches)$$

  - We get only those tuples of "instructor × teaches" that pertain to instructors and the courses that they taught
    - i.e., those tuples where instructor.id = teaches.id

Join Operation

- The join operation allows us to combine a select operation and a Cartesian-Product operation into a single operation
  - Consider relations $r(R)$ and $s(S)$:
  - Let "theta (θ)" be a predicate on attributes in the schema R "union" S. The join operation $r \bowtie_\theta s$ is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

  - Thus, $\sigma_{instructor.id = teaches.id} (instructor \times teaches)$ can equivalently be written as:

$$instructor \bowtie_{Instructor.id = teaches.id} teaches$$

- When theta (θ) is not provided, perform natural join

Union Operation

- The union operation allows us to combine two relations
  - Notation: $r \cup s$
- For $r \cup s$ to be valid:
  - r, s must have the same arity (same number of attributes)
  - The attribute domains must be compatible
    - Example: 2nd column of r deals with the same type of values as does the 2nd column of s

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations
  - Notation: $r \cap s$
- Assume (same as Union):
  - r, s have the same arity
  - Attributes of r and s are compatible

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another
  - Notation: $r - s$
- Assume (same as Union and Set Intersection):
  - r, s have the same arity
  - Attributes of r and s are compatible

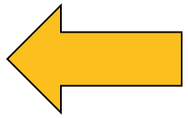
Normalization: A First Look

Design Alternatives

- In designing a database schema, we must ensure that we avoid two major pitfalls
  - **Redundancy**: a bad design may result in repeated information
    - E.g., store course identifier and title of a course for each course offering
      - Only store course identifier is sufficient
    - Redundant representation of information may lead to data inconsistency among the various copies of information
      - E.g., update is not performed on all the copies
  - **Incompleteness**: a bad design may make certain aspects of the enterprise difficult or impossible to model
    - E.g., only have entity for course offering, but without entity for courses
      - Impossible to model new courses that are not offered yet

Design Alternatives

- Avoiding bad designs is not enough
  - There may be many good designs from which we must choose
- For example, a customer who buys a product
  - The sale activity is a relationship between the customer and the product?
  - The sale activity is a relationship among the customer, the product, and the sale itself?
    - i.e., the sale can be considered as an entity
- Database design can be difficult
  - When #entities and #relationships are large
- Do we have any guidelines on how to get a good design?
  - Normal Forms (范式)!



Normalization （规范化）

- In practice, we usually just satisfy 1NF, 2NF and 3NF

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells cannot have tables as values)[5]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[8]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

https://en.wikipedia.org/wiki/Database_normalization

First Normal Form (1NF，第一范式)

- A relational schema *R* is in 1NF if the domains of all attributes of R are atomic
  - Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS307 that can be broken up into parts
      - However, in practice, we can also consider it atomic
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data

First Normal Form (1NF)

- Example: Non-atomic attribute

station_id	name	location
1	Luohu(罗湖)	114.11833 , 22.53111
2	Guomao(国贸)	114.11889 , 22.54
3	Laojie(老街)	114.11639 , 22.54444
4	Grand Theater(大剧院)	114.10333 , 22.54472
5	Science Museum(科学馆)	114.08972 , 22.54333
6	Huaqiang Rd(华强路)	114.07889 , 22.54306
7	Gangxia(岗厦)	114.06306 , 22.53778
8	Convention and Exhibition Center Station(会展中心)	114.05472 , 22.5375
9	Shopping Park(购物公园)	114.05472 , 22.53444
10	Xiangmihu(香蜜湖)	114.034 , 22.5417

First Normal Form (1NF)

- Fix it by splitting the names into two columns

station_id	english_name	chinese_name	longitude	latitude
1	Luohu	罗湖	114.11833	22.53111
2	Guomao	国贸	114.11889	22.54
3	Laojie	老街	114.11639	22.54444
4	Grand Theater	大剧院	114.10333	22.54472
5	Science Museum	科学馆	114.08972	22.54333
6	Huaqiang Rd	华强路	114.07889	22.54306
7	Gangxia	岗厦	114.06306	22.53778
8	Convention and Exhibition Cent…	会展中心	114.05472	22.5375
9	Shopping Park	购物公园	114.05472	22.53444
10	Xiangmihu	香蜜湖	114.034	22.5417

First Normal Form (1NF)

- Another example: Starring
  - Problems: 1) Redundant names; 2) difficulties in updating/deleting a specific person; 3) extra cost in splitting names; 4) difficulties in making statistics

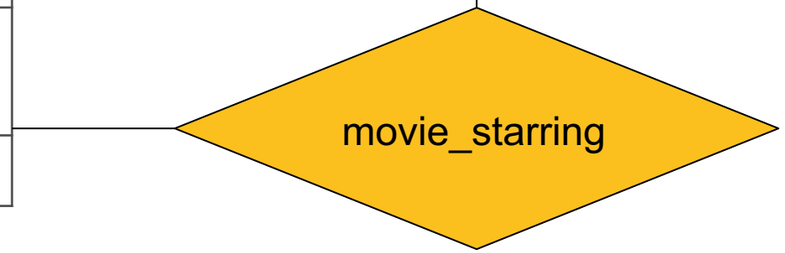
Movie ID	Movie Title	Country	Year	Director	Starring
0	Citizen Kane	US	1941	welles, o.	Orson Welles, Joseph Cotten
1	La règle du jeu	FR	1939	Renoir, J.	Roland Toutain, Nora Grégor, Marcel Dalio, Jean Renoir
2	North By Northwest	US	1959	HITCHCOCK, A.	Cary Grant, Eva Marie Saint, James Mason
3	Singin' in the Rain	US	1952	Donen/Kelly	Gene Kelly, Debbie Reynolds, Donald O'Connor
4	Rear Window	US	1954	Alfred Hitchcock	James Stewart, Grace Kelly

First Normal Form (1NF)

- Fix it by treating the column as a multi-valued attribute
  - *movie_starring* table has two foreign keys, *movid_id* and *star_id*

Movie ID	Movie Title	Country	Year	Director
0	Citizen Kane	US	1941	welles, o.
1	La règle du jeu	FR	1939	Renoir, J.
2	North By Northwest	US	1959	HITCHCOCK, A.
3	Singin' in the Rain	US	1952	Donen/Kelly
4	Rear Window	US	1954	Alfred Hitchcock

Star ID	Firstname	Lastname	Born	Died
1				
2				
3				



Second Normal Form (2NF，第二范式)

- A relation satisfying 2NF must:
  - be in 1NF
  - not have any non-prime attribute that is dependent on any proper subset of any candidate key of the relation
    - A non-prime attribute of a relation is an attribute that is not a part of any candidate key of the relation
  - 不 包含只依赖于 主键中部分属性 的非主属性
    - “非主属性”是指 不属于 任何候选键的属性

Second Normal Form (2NF)

- Example: consider this table with the composite primary key (*station_id*, *line_id*)

station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - The columns *line_color* and *operator* are not related to *station_id*
    - They are only related to *line_id*, which is only part of (a subset of) the primary key
  - Similarly, *english_name*, *chinese_name*, and *district* are not related to *line_id*
    - They are only related to *station_id*, which is only part of (a subset of) the primary key
  - 非主属性 *line_color*, *operator*, *english_name*, *chinese_name*, *district* 只依赖于主键中的部份属性

Second Normal Form (2NF)

- Example: Consider this table with the composite primary key (*station_id, line_id*)

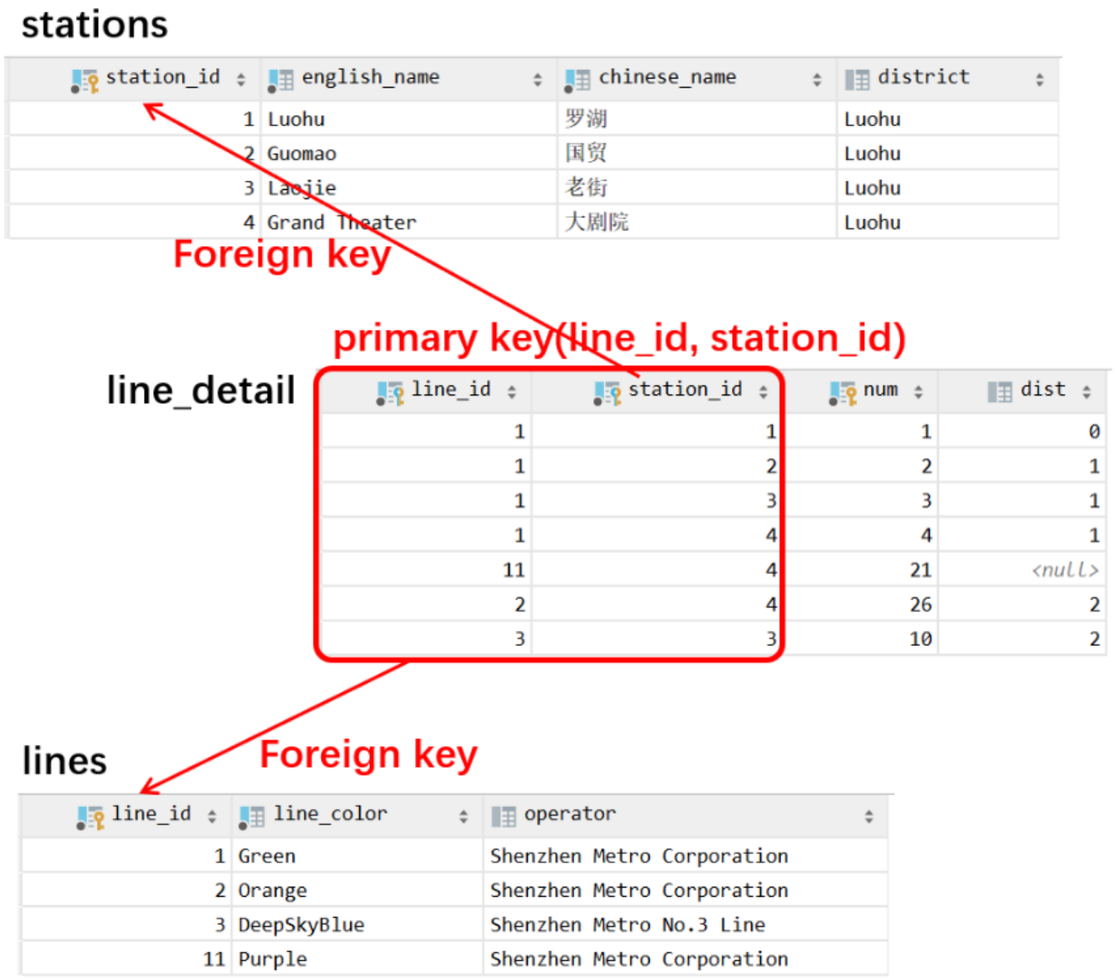
station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - Problem when not meeting 2NF: Insertion and deletion anomaly
    - We CANNOT insert a new station with no lines assigned yet (unless using NULLs)
    - If we delete a line, all stations associated with this line will be deleted as well

Second Normal Form (2NF)

- Fix it by
  - Splitting the two unrelated parts into two different tables of entities
  - And create a relationship set (if it is the many-to-many relationship between the two entities)

- By the way…
  - A relation with a single-attribute primary key is automatically in 2NF once it meets 1NF



Third Normal Form (3NF，第三范式)

- A relation satisfying 3NF must:
  - be in 2NF
  - all the attributes in a table are determined only by the candidate keys of that relation, not by any non-prime attributes
  - 所有属性 只依赖于主键，不依赖于任意非主属性

Third Normal Form (3NF)

- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan

    - *station_id* depends on the primary key (*bus_line*)
    - However, the columns *chinese_name*, *english_name*, and *district* depend on *station_id*, which is not the primary key.
      - They only have "indirect/transitive" dependence （非直接/传递依赖） on the primary key
    - Problem: Data redundancy

Third Normal Form (3NF)

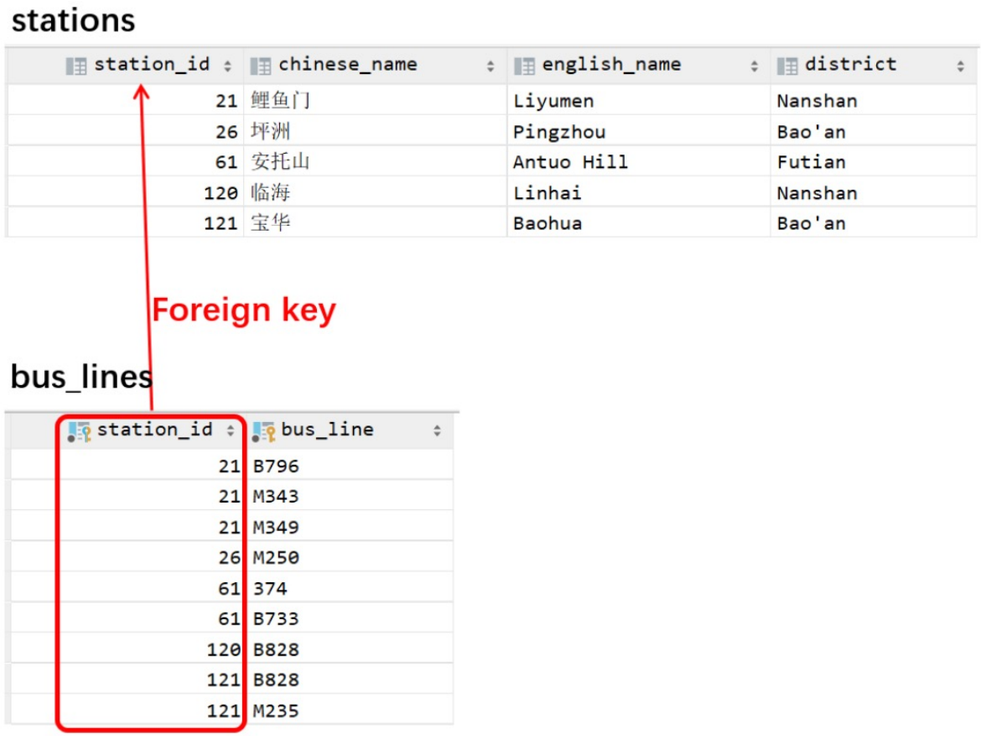
- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan

  - Problem when not meeting 3NF:
    - Data redundancy: as you can see in the table, the attributes for a station have been stored multiple times
    - Insertion and deletion anomaly: inserting a new bus line with no station becomes impossible without NULLs; deleting a station/bus line may also delete corresponding bus lines/stations.

Third Normal Form (3NF)

- Fix it by:
  - Create a new table with *station_id* as the primary key
    - i.e., the column which *chinese_name*, *english_name*, and *district* depend on
  - Move all columns which depend on the new primary key into the new table
    - … and, only leave the primary key of the new table (*station_id*) in the original table
  - (*In practice, if necessary) Add a foreign-key constraint
    - Not related to relational database modeling, only in implementations



Normalization

- In practice, we usually just satisfy 1NF, 2NF and 3NF

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells cannot have tables as values)[5]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[8]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

https://en.wikipedia.org/wiki/Database_normalization

Normalization

Every non key attribute must
provide a fact about the key,
the whole key,
and nothing but the key.

2NF

3NF

William Kent (1936 – 2005)

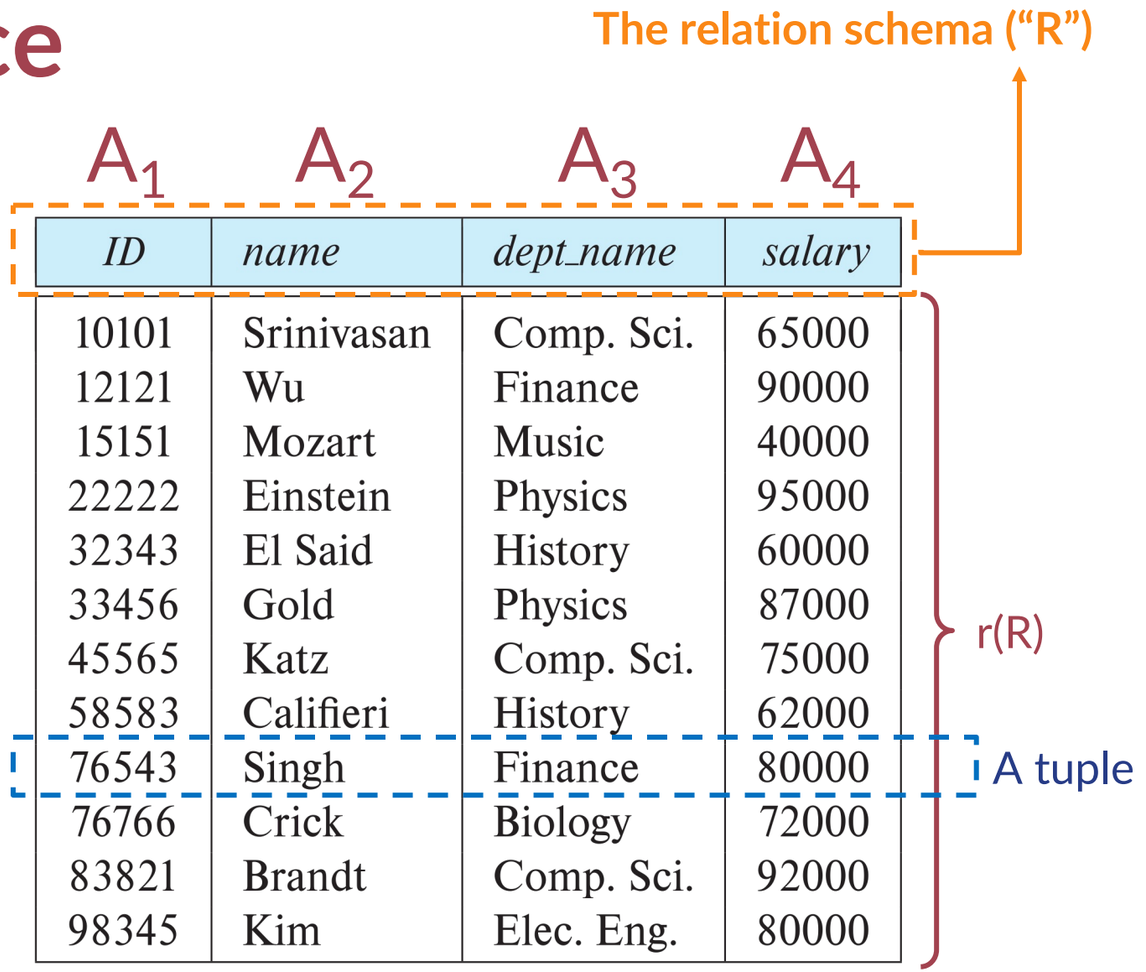
William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM 26 (2), Feb. 1983, pp. 120–125.

(Recall) Prerequisites of Decomposition & Functional Dependency

Relation Schema and Instance

- A_1, A_2, …, A_n are attributes
- $R = (A_1, A_2, …, A_n)$ is a **relation schema**
  - Example on the right side:

    instructor = (*ID*, *name*, *dept_name*, *salary*)
- $r(R)$ denotes a relation instance r defined over schema R
  - Or to say, the entire table on the right side
- An element t of relation r is called a tuple
  - … and is represented by a row in a table



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Relation Schema and Instance

- An analogy to programming languages:
  - Relation - Variables
  - Relation schema – Variable types
  - Relation instance – Value(s) stored in the variable

Database Schema

- Database schema is the logical structure of the database
  - It contains a set of relation schemas
  - … and a set of integrity constraints
- Database instance is a snapshot of the data in the database at a given instant in time

Keys

- Let K $\subseteq$ *R*
  - K is a **superkey** of *R* if values for K are sufficient to identify a unique tuple of each possible relation *r*(*R*)
    - E.g., {*ID*} and {*ID*, *name*} are both superkeys of instructor
    - If K is a superkey, any superset K' of K where K' $\subseteq$ *R* is a superkey as well
  - Superkey K is a **candidate key** if K is minimal, i.e., no proper subset of K is a superkey
    - E.g., {ID} is a candidate key for *instructor*
  - One of the candidate keys is selected to be the **primary key**
    - We mark the primary key with an underline:

      instructor = (*ID*, *name*, *dept_name*, *salary*)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

Decomposition & Functional Dependency

Features of Good Relational Designs

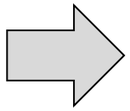
- Suppose we combine *instructor* and *department* into *in_dep*, which represents the natural join on the relations *instructor* and *department*
  - There is repetition of information (e.g., building and budget)
    - Could lead to inconsistency
  - Need to use nulls (if we add a new department with no instructors)
    - In many cases, null values are troublesome, as we saw in our study of SQL

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Physics	Watson	70000
Finance	Painter	120000
History	Painter	50000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Biology	Watson	90000
Comp. Sci.	Taylor	100000
History	Painter	50000
Comp. Sci.	Taylor	100000
Music	Packard	80000
Physics	Watson	70000
Finance	Painter	120000

department



ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

in_dep

Decomposition

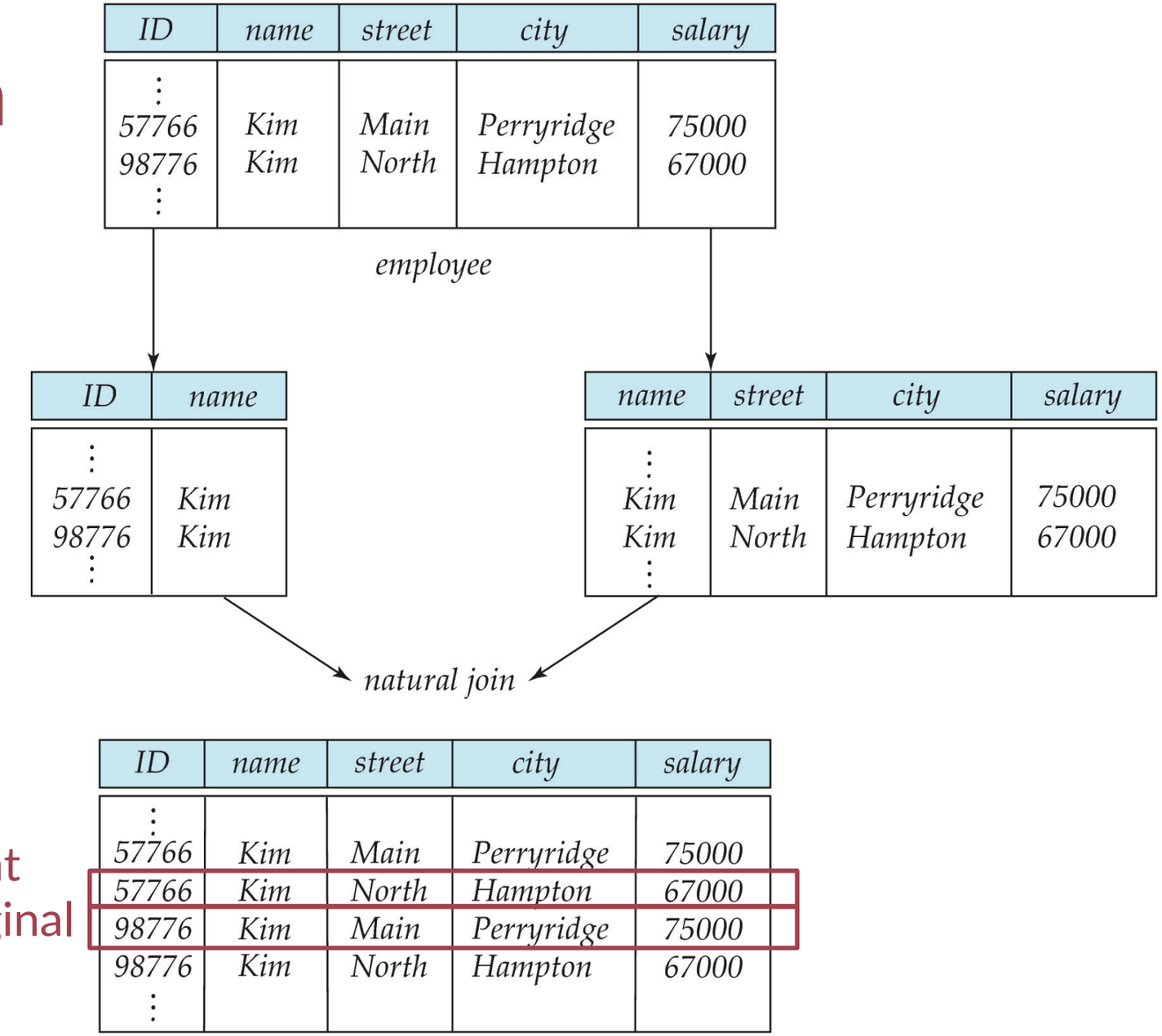
- Avoid the repetition-of-information problem
  - Decompose *in_dep* into two schemas: *instructor* and *department*

- However, not all decompositions are good
  - E.g., decompose *employee*(ID, name, street, city, salary) into:
    - employee1(ID, name)
    - employee2(name, street, city, salary)

  The problem arises when we have two employees with the same name

A Lossy Decomposition

- (Continue) we cannot reconstruct the original employee relation with the join operation
  - Unable to represent certain important facts about the university employee
  - We call it a **lossy decomposition**



Two “ghost” records that do NOT exist in the original table

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R
  - That is, $R = R_1 \cup R_2$
  - The decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R = R_1 \cup R_2$

- Formally, $\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$
  - … and a decomposition is lossy if $r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$

(!) proper subset

- Or to say, the two SQL queries on the right side generate identical results:

```
select * -- 1
from (select R1 from r)
    natural join
     (select R2 from r);

select * from R; -- 2
```

Normalization Theory

- Decide whether a particular relation *R* is in "good" form
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - * Multivalued dependencies (self study)
- Generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world

- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID
  - Each student and instructor has only one name
  - Each instructor and student is (primarily) associated with only one department
    - E.g., an instructor can have multiple departments, a student can have double major
    - But we assume that they can only have one
  - Each department has only one value for its budget, and only one associated building

Functional Dependencies

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation
  - A legal instance of a database is one where all the relation instances are legal instances
- Constraints on the set of legal relations
  - Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a key
- Functional dependencies allow us to express constraints that we cannot express with superkeys

Definition of Functional Dependencies

- Let R be a relation schema, and $\alpha \subseteq R$ *and* $\beta \subseteq R$,

  the functional dependency

  $$\alpha \rightarrow \beta$$

  holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β.

  That is,

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

- Example: Consider $r(A, B)$ with the following instance of r,
  - On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold

A	B
1	4
1	5
3	7

Closure (闭包) of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F:
  - 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 F^+
  - F^+ is a superset of F

Keys and Functional Dependencies

- Let's see how can we (re)define the concept of "**keys**" under the language of functional dependencies

Keys and Functional Dependencies

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for **no** $\alpha \subset K$, $\alpha \rightarrow R$

(!) proper subset, again

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys

- E.g. Consider the schema: *inst_dept*(*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)
  - We expect these functional dependencies to hold:

    dept_name*→ *building*, *ID* → *building

  … but would not expect the following to hold:

  dept_name* → *salary

Use of Functional Dependencies

- We use functional dependencies to
  - To test relations to see if they are legal under a given set of functional dependencies
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F*
    - For each functional dependency $\alpha \rightarrow \beta$, for all pairs of tuples t1 and t2 in the instance, $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
  - To specify constraints on the set of legal relations
    - If all legal relations on *R* satisfy the set of functional dependencies *F*, we say that *F* holds on *R*

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- Example: List some functional dependencies that the table satisfies
  - $A \rightarrow C$ (but C $\rightarrow$ A is not satisfied)
  - $D \rightarrow B$

Can you find more?

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Use of Functional Dependencies

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.

- Example: we see that *room_number* → *capacity* is satisfied.
  - However, in real world, two classrooms in different buildings can have the same room number but with different room capacity
  - We prefer {*building*, *room_number*} → *capacity*

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Trivial Functional Dependencies

- A functional dependency is **trivial** if it is satisfied by all relations

- Example:
  - *ID*, *name* → *ID*
  - *name* → *name*

- In general,
  - $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless
  - For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

  - A decomposition of R into R_1 and R_2 is a lossless decomposition if at least one of the following dependencies is in F^+:
    - $R_1 \cap R_2 \rightarrow R_1$
    - $R_1 \cap R_2 \rightarrow R_2$

In other words, if $R_1 \cap R_2$ forms a superkey for either R_1 or R_2, the decomposition of R is a lossless decomposition

Lossless Decomposition

- Example:
  - *in_dep (ID, name, salary, dept_name, building, budget)*
  - … and the decomposed schemas, *instructor* and *department*:
    - *instructor(ID, name, dept_name, salary)*
    - *department(dept_name, building, budget)*

$instructor \cap department = dept\_name$

$dept\_name \rightarrow dept\_name, building, budget$

(… which means the decomposition is lossless)

Lossless Decomposition

- Another example:

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
- *Note:*
  - $B \rightarrow BC$

    is a shorthand notation for
  - $B \rightarrow \{B, C\}$

Lossless Decomposition

- Note: the above functional dependencies are a sufficient condition for lossless join decomposition
  - The dependencies are a necessary condition only if all constraints are functional dependencies

*(There are other types of constraints, e.g., **multivalued dependencies**, that can ensure that a decomposition is lossless even if no functional dependencies are present)*

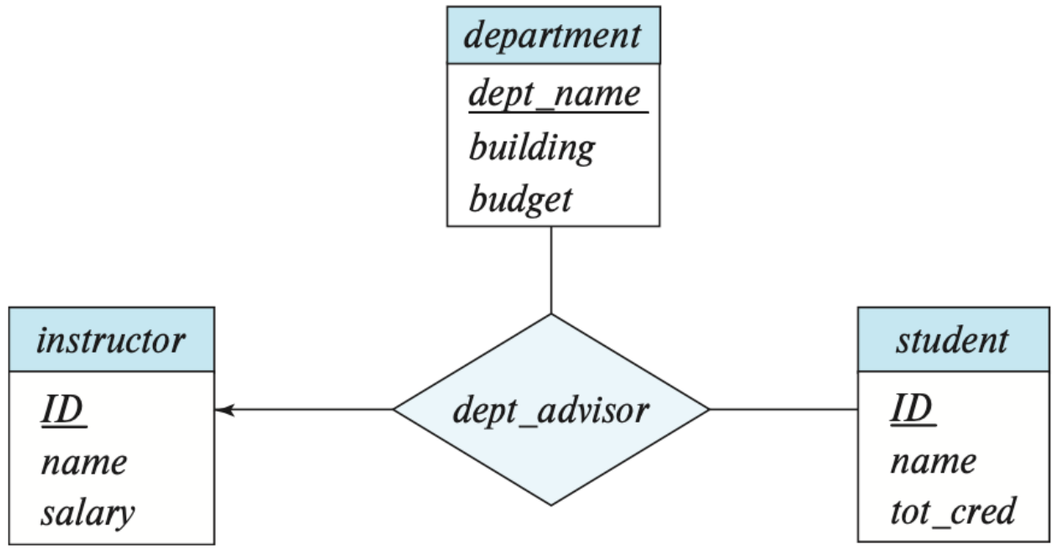
Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
  - It is useful to design the database in a way that constraints can be tested efficiently.

- If a functional dependency in the original relation *R* does not exist in any of the decomposed relations, we say it is not dependency-preserving
  - In the dependency preservation, at least one decomposed table must satisfy every dependency

Dependency Preservation

- Consider a new E-R design for relationships between students, instructors, and departments
  - An instructor can only be associated with one department
  - A student can have multiple advisors but not more than one from a given department
    - Think about double-major students



Dependency Preservation

- Consider a schema
  - *dept_advisor(s_ID, i_ID, dept_name)*
  - … with function dependencies: (1) *i_ID* → *dept_name* (2) *s_ID*, *dept_name* → *i_ID*

In the above design, we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.

Dependency Preservation

- Consider a schema
  - *dept_advisor(s_ID, i_ID, dept_name)*
  - … with function dependencies: (1) *i_ID* → *dept_name* (2) *s_ID*, *dept_name* → *i_ID*

In the above design, we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.

- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition will not include all the attributes in

    s_ID, *dept_name* → *i_ID*
  - Thus, the decomposition will NOT be dependency-preserving

Dependency Preservation

- Problem when not meeting dependency preservation
  - Every time the database wants to check the integrity of the functional dependency

    $$s\_ID, dept\_name \rightarrow i\_ID,$$

    the decomposed tables must be joined

  - … where the computational cost could be very high with join operations

BCNF and 3NF

Normal Forms: Revisited

- Boyce-Codd Normal Form (BCNF)
- 3NF
- Higher-order normal forms

Boyce-Codd Normal Form

- A relation schema *R* is in **BCNF** with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R

- * A database design is in BCNF if each member of the set of relation schemas that constitutes the design is in BCNF

Boyce-Codd Normal Form

- Example schema that is **not** in BCNF:
  - *in_dep (ID, name, salary, dept_name, building, budget)*

  Because,

  $$dept\_name \rightarrow building, budget$$

  - holds in *in_dep*, however, *dept_name* is not a superkey
    - … where {ID, dept_name} is

  - When decompose *in_dept* into *instructor* and *department*
    - *instructor* is in BCNF
    - *department* is in BCNF

Decomposing a Schema into BCNF

- Let *R* be a schema *R* that is not in BCNF
- Let $\alpha \rightarrow \beta$ be the functional dependency that causes a violation of BCNF
  - We decompose *R* into:
    - $(\alpha \cup \beta)$
    - $(R - (\beta - \alpha))$

- Example: *in_dep (<u>ID</u>, name, salary, <u>dept_name</u>, building, budget)*
  - α = *dept_name*, β = *building*, *budget*
  - Thus, *in_dep* is replaced by:
    - $(\alpha \cup \beta)$ = (*dept_name*, *building*, *budget*)
    - $(R - (\beta - \alpha))$ = (*ID*, *name*, *dept_name*, *salary*)

Decomposing a Schema into BCNF

- Another example:
  - $R = (A, B, C)$
    $F = \{A \rightarrow B, B \rightarrow C\}$
  - $R_1 = (A, B)$, $R_2 = (B, C)$
    - Lossless-join decomposition:

      $$R_1 \cap R_2 = \{B\} \quad \text{and } B \rightarrow BC$$

    - Dependency preserving
  - $R_1 = (A, B)$, $R_2 = (A, C)$
    - Lossless-join decomposition:

      $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

    - Not dependency preserving
      (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider the schema (that we have visited before)
  - *dept_advisor(s_ID, i_ID, dept_name)*
  - … with function dependencies: (1) *i_ID* → *dept_name* (2) *s_ID*, *dept_name* → *i_ID*

  - *dept_advisor* is not in BCNF since for *i_ID* → *dept_name*, i_ID is not a superkey
    - (where {*s_ID*, *i_ID*, *dept_name*} is)

- To fix this problem, we need to decompose *dept_advisor*
  - However, any decomposition will **not** include all the attributes in
    s_ID, *dept_name* → *i_ID*
  - Thus, the decomposition will NOT be dependency-preserving

Third Normal Form (3NF)

- A relation schema *R* is in **third normal form (3NF)** if for all

$$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*

- Notes
  - Each attribute *A* may be in a different candidate key
  - If a relation is in BCNF, it is in 3NF (… since in BCNF, one of the first two conditions above must hold)
  - The third condition above is a minimal relaxation of BCNF to ensure dependency preservation

3NF Example

- Consider the schema (that we have visited before)
  - *dept_advisor(s_ID, i_ID, department_name)*
  - … with function dependencies: (1) *i_ID* $\rightarrow$ *dept_name* (2) *s_ID*, *dept_name* $\rightarrow$ *i_ID*

  - We have two candidate keys: {*s_ID*, *dept_name*} and {*s_ID*, *i_ID*}

- *dept_advisor* is not in BCNF, but it can be in 3NF
  - {s_ID, dept_name} is a superkey
  - *i_ID* $\rightarrow$ *dept_name* and *i_ID* is NOT a superkey (which violates BCNF), but:
    - α is *i_ID*, β is *dept_name*
    - {*dept_name*} – {*i_ID*} = {*dept_name*}
    - *dept_name* is contained in a candidate key (-> {*s_ID*, *dept_name*})

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$, $F = \{JK \rightarrow L, L \rightarrow K\}$, and an instance table:

J	*L*	*K*
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- Problems in this table:
  - Repetition of information
    - Row 1-3: L and K
  - Need to use nulls
    - Row 4: Represent the relationship l_2, k_2 with no corresponding value for J

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF
  - It is always possible to obtain a 3NF design without sacrificing *losslessness* or *dependency preservation*
- Disadvantages to 3NF
  - We may have to use nulls to represent some of the possible meaningful relationships among data items
  - There is a problem of potential repetition of information

Goals of Normalization

- Let *R* be a relation scheme with a set *F* of functional dependencies
  - Decide whether a relation scheme *R* is in “good” form.
  - In the case that a relation scheme R is not in “good” form, need to decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that:
    - Each relation scheme is in good form
    - The decomposition is a lossless decomposition
    - *Preferably*, the decomposition should be dependency preserving