Chapter 7: Normalization

Database System Concepts, 7th Ed.

Overview of Normalization

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

-
تست کردن میشه این فصل
تیبل نهایی فرم خوبی داره تهش
هدف نرمال سازی این است که ناهنجاری داده ای رو از بین ببریم
جدول اولیه رو می کنیم جدول کوچکتر --> توی نرمال سازی این کارو میکنیم ینی تجزیه می کنیم
در نهایت ما جدول هایی داریم که افزونگی توی جدول کمتر میشه

Anomalies in DBMS

- There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.
- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *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

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

-
افزونگی داده داریم توی جدول
id کلید است
مثلا میخوایم یه رکوردی رو اضافه بکنیم که id فعلا نداره و نمی تونیم چون id کلید است پس باید یه چیزی اول داشته باشد که بتونیم رکورد اضافه بکنیم پس مشکل درج داریم
اگر بخوایم بودجه یه دانشکده رو اپدیت بکنیم باید بودجه اون دانشکده رو توی کل اون رکورد ها اپدیت بکنیم که این کارو سخت میکنه و اگر فقط بخوایم واسه همون یه رکورد اون کارو بکنیم این کار غلط میشه چون بودجه دانشکده فیزیک یه مقدار ثابته همیشه
دیلیت: نمی تونیم دانشکده ای داشته باشیم که استاد نداشته باشه ؟؟
نکته: با تجزیه این جدول به دو جدول کوچکتر مشکل هایی بالایی از بین می ره ینی طراحی ما درست داره انجام میشه
نکته: اگر اون دو جدول کوچک رو جوین کنیم میشه جدول بزرگه ینی هیچ اطلاعاتی از دست نمی ره و این دوتا معادل هم هستند

A Lossy Decomposition

اینجا تجزیه درست نیست

ID	name	street	city	salary
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000

employee

ID	name
57766	Kim
98776	Kim

name	street	city	salary
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000

natural join

ID	name	street	city	salary
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000

با جوین کردن ما تکرار داریم و دیگه اون دانش اولیه هم نداریم ینی چی؟ مثلا اینجا با ایدیه 57766 دو نفر رو داریم که غلطه ینی دانش اولیه رو از دست دادیم

Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$

  $R_1 = (A, B)$ $\quad$ $R_2 = (B, C)$

اینجا تجزیه داریم که دانش اولیه رو از دست نمیدیم
اینجا اطلاعاتی از دست ندادیم و اطلاعات تکراری هم نداریم

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

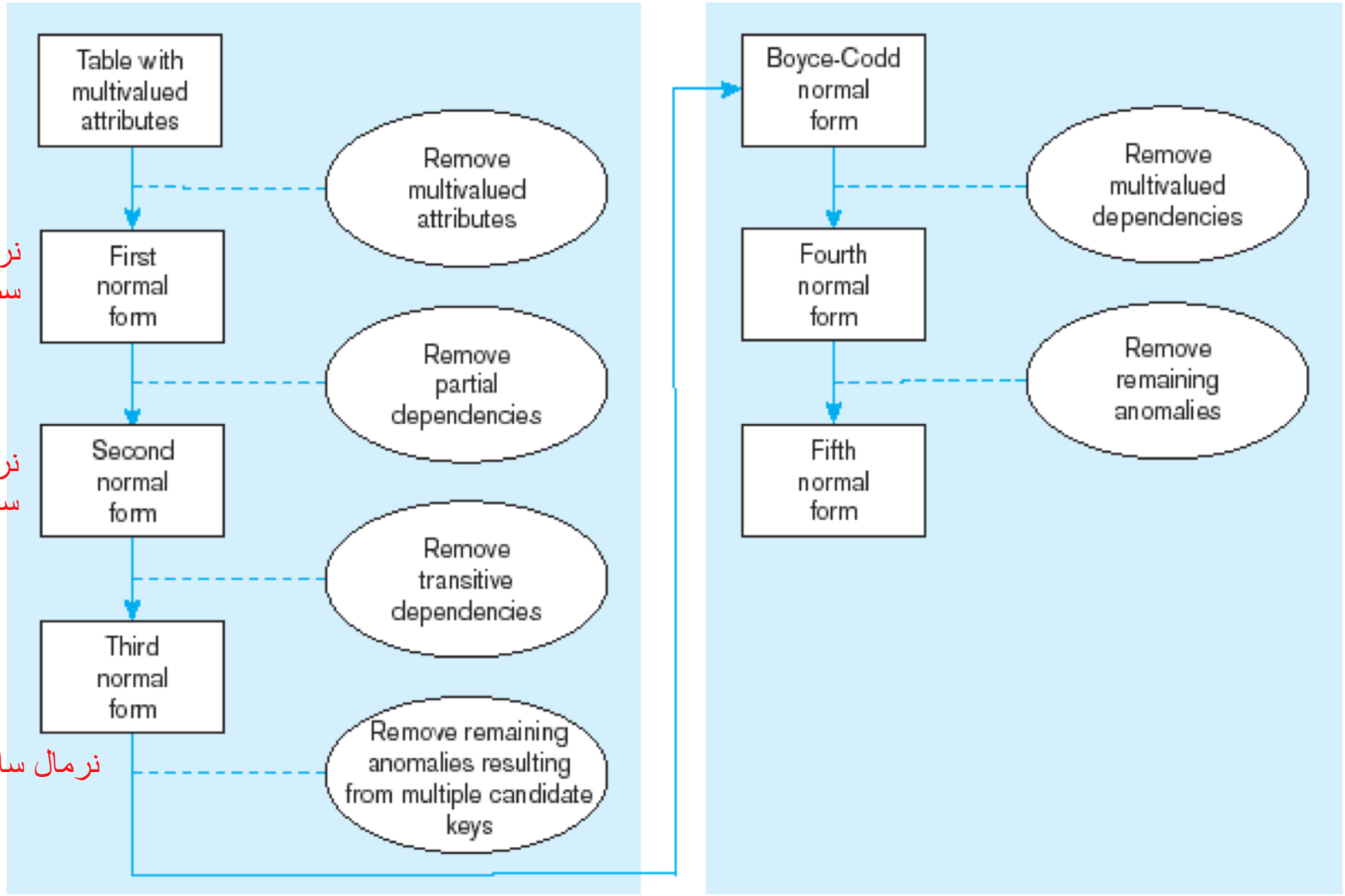
A	B	C
α	1	A
β	2	B

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 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

-
رابطه هایی رو توی جدول داشته باشیم که فرم خوبی داشته باشند
اگر رابطه ما خوش فرم نیستند اونارو تجزیه میکنیم به جدول های کوچیکی که خوش فرم هستند
این تجزیه هایی که داریم از جنس lossless است

An overall view of steps



نرمال سازی سطح 1

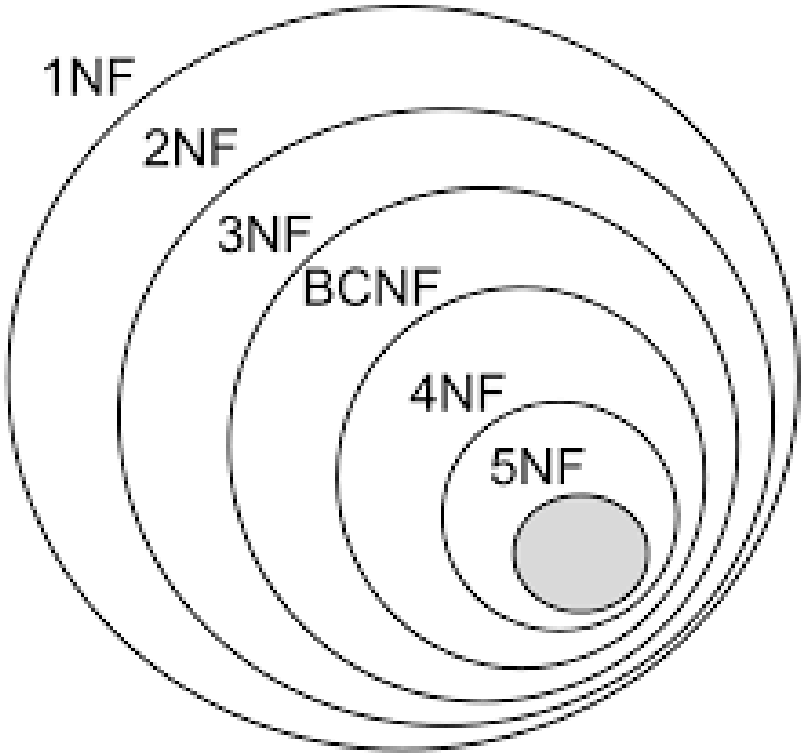
نرمال سازی سطح 2

نرمال سازی سطح 3

توی بحث نرمال سازی سطوح مختلفی داریم

The Data Normalization Process

- A group of tables is said to be in a particular normal form if every table in the group is in that normal form.
- The data normalization process is progressive.
  – For example, if a group of tables is in second normal form, it is also in first normal form.



-
جدول های پایگاه دادمون توی سطوح مختلف چک بشه و بررسی میکنیم اون جداول توی اون سطح هستند یا نه گروهی از جداول رو میگیم توی یه سطحی از نرمال سازی قرار گرفتند اگر تمام اون جداول اون سطح نرمال سازی رو شامل شده باشند
نکته: اگر جدول ما مثلا 4NF باشه حتما جدول های قبلیش رو مثل 1NF,..رو هم شامل میشه

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- The multivalued attributes have been eliminated

دامنه های ما هم اتمیک هستند

پس باید ویژگی های چند مقداره رو از جدول حذف کنیم

- Example: Records contain multivalued attributes

ID	FistName	LastName	Knowledge
1	John	Day	C++,Java,C
2	Thom	Luce	PHP, Java

ستون knowledge چند مقداره است که این اتمیک نیست یعنی این جدول نرمال سطح یک نیست

-
به جدولی میگیم نرمال سطح یک که داخل هیچ رکوردی ما نتونیم مقداری واسه یک ستون پیدا کنیم که چندتا مقدار داخلش باشه ینی مقادیر اتمیک هستند

First Normal Form: Example

ID	FistName	LastName
1	John	Day
2	Thom	Luce

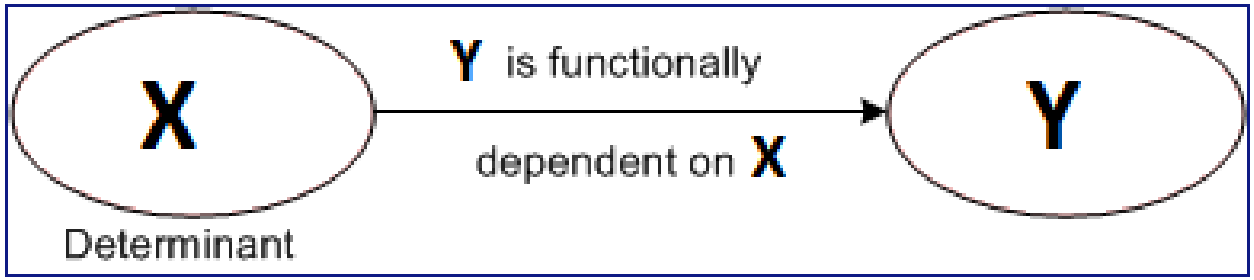
ID	Knowledge
1	C++
1	Java
1	C
2	PHP
2	Java

OR

ID	FistName	LastName	Knowledge
1	John	Day	C++
1	John	Day	Java
1	John	Day	C
2	Thom	Luce	PHP
2	Thom	Luce	Java

Functional Dependencies

y وابستگی تابعی داره به x ینی با داشتن یک مقدار از x به یک مقدار از y برسیم



Functional dependency between X and Y

student_name وابستگی تابعی دارد به student_id

Student_ID ⟶ Student_Name با یک id فقط به یک اسم می رسیم

- Student_ID is the determinant.
- The value of Student_ID determines the value of Student_Name.
- Student_Name is functionally dependent on Student_ID.

-
وابستگی تابعی که اساس کار ما است
یه عضو از دامنه که برمیداریم ما رو به یه عضو از برد فقط برسونه ینی اگر x1=x2 است حتما به یه مقدار از y برسیم

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation;
- 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.
  - Each department has only one value for its budget, and only one associated building.

Functional Dependencies Definition

الفا و بتا یه سری ویژگی ها هستند حالا الفا می تونه یه ویژگی باشه یا می تونه چندتا باشه

- Let *R* be a relation schema

$$\alpha \subseteq R \quad and \quad \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

بتا وابستگی تابعی داره به الفا در شرایطی که اگر و تنها اگر دوتا تاپل از r که گرفتیم واسه الفا با هم برابر باشند در این حالت بتاهاشون هم با هم برابر است

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,

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

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

1	4
1	5
3	7

- On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold,

از A نمی رسیم به B ینی نمی تونیم بگیم B وابستگی تابعی داره به A چون دوتا رکورد از A داریم که با هم برابرند ولی B هاشون یکی نیست

Closure of Attribute Sets

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

هر صفتی که وابستگی تابعی داره مثلا با id رو در نظر بگیریم و همه رو با هم احتماع بکنیم
یک مجموعه ای به دست میاد که بهش میگن id به توان + مثلا همون الفا بالا

وابستگی تابعیه اولیه رو توی یه مجموعه به اسم F قرارم میدن
پس F شامل وابستگی های اولیه است که توی اون بیزنس در اول کار به ما دادن

Closure of Attribute Sets

- Algorithm to compute α^+, the closure of α under F

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

الگوریتم الفا پلاس رو داره میگه

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$ برای A,G با هم دیگه

همه صفات R وابستگی تابعی دارند به AG

  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is *AG* a candidate key?

تا اینجای کار تمام صفت هایی که توی مجموعه R بوده به result اضافه کردیم

  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq$ R
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$?
     2. Does $G \rightarrow R$?

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of *R*.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful

ویژگی های بستا الفا:
الفا میتونه بررسی کنه که سوپر کلید هست یا نه --> اگر الفا تمام صفت های R رو شامل بشه می تونیم بگیم هست
برای اینکه ببینم بتا وابستگی تابعی داره به بستار الفا کافیه ببینیم که بتا زیرمجموعه الفا استار هست یا نه

Closure 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 $A \rightarrow C$
  - etc.
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by F^+.

تمام وابستگی های تابعی که میشه از اون مجموعه اولیه برداشت کرد و بهش رسید رو به دست میاریم و همشو اضافه میکنیم به مجموعه F پلاس

Closure of a Set of Functional Dependencies

- We can compute F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - **Reflexive rule:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ — قانون انعکاسی — گاما یک صفت است
  - **Augmentation rule:** if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ — رابطه افزایشی
  - **Transitivity rule:** if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ — رابطه تعدی

  با این سه تا قانون می تونیم به بستار F برسیم
- These rules are
  - **Sound** -- generate only functional dependencies that actually hold, and
  - **Complete** -- generate all functional dependencies that hold.

Example of F^+

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,
      and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
      and then transitivity

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - قانون اجتماع **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
  - قانون تجزیه **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
  - قانون شبه تعدی **Pseudotransitivity rule**:If $\alpha \rightarrow \beta$ holds and $\gamma\ \beta \rightarrow \delta$ holds, then $\alpha\ \gamma \rightarrow \delta$ holds.
- The above rules can be inferred from Armstrong's axioms.

Extraneous Attributes

مفهوم صفت اضافی

- An attribute of a functional dependency in F is **extraneous** if we can remove it without changing F^+
- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - **Remove from the left side**: Attribute A is **extraneous** in α if
    - $A \in \alpha$ and می خوایم ببینیم ویژگی A اضافی هست یا ن و این A زیرمجموعه از الفا است
    - F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$. به این صورت میشه A رو حذف کرد
  - **Remove from the right side**: Attribute A is **extraneous** in β if
    - $A \in \beta$ and
    - The set of functional dependencies

      $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one

یه ویژگی a توی این F اضافیه و می تونه حذف بشه اگر ک توی این مجموعه جدید که F پرین هست و a داخلش نیس و ما بتونیم از طریق این F پرین برسیم به یک F پرین پلاسی که این F پرین پلاس با همون F پلاس یکی باشه

Extraneous Attributes, Example

- Example: Given F = {AB → C, A → D, D → C}
  - Then we can show that F logically implies A → C, making B extraneous in AB → C.

B بخوایم از سمت چپ حذف کنیم پس باید از اون مجموعه اولیه B رو حذف کنیم میگیم از رابطه تعدی داریم
A->D و از D->C می تونیم برسیم به A->C پس B توی اون رابطه بالا اضافس و میشه حذفش کرد

- Example: Given F = {A → C, AB → CD}
  - C is extraneous in AB → CD

C را از سمت راست میخوایم حذف کنیم

  - We can show that even after replacing *AB → CD* by *AB → D*, we can still infer *AB → C* and thus *AB → CD*.

Testing if an Attribute is Extraneous

- Let R be a relation schema and let F be a set of functional dependencies that hold on R . Consider an attribute in the functional dependency $\alpha \rightarrow \beta$.
- To test if attribute $A \in \beta$ is extraneous in β
  - Consider the set:
    $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  - check that α^+ contains A; if it does, A is extraneous in β
- To test if attribute $A \in \alpha$ is extraneous in α
  - Let $\gamma = \alpha - \{A\}$. Check if $\gamma \rightarrow \beta$ can be inferred from F.
    - Compute γ^+ using the dependencies in F
    - If γ^+ includes all attributes in β then , A is extraneous in α

Examples of Extraneous Attributes

- Let $F = \{AB \rightarrow CD, A \rightarrow E, E \rightarrow C\}$
- To check if C is extraneous in $AB \rightarrow CD$, we: C صفت اضافی است
  - Compute the attribute closure of AB under $F' = \{AB \rightarrow D, A \rightarrow E, E \rightarrow C\}$
  - The closure is *ABCDE*, which includes *CD*
  - This implies that *C* is extraneous

اول F پرین به دست میاریم
بستار سمت چپ رابطه رو به دست میاریم و اگر در اخر CD توی بستار بود می تونیم بگیم C اضافی است

Canonical Cover

Intuitively, a **canonical cover** of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

A **canonical cover** for *F* is a set of dependencies F_c such that

اگر سمت چپشون یکی بود ینی داشتیم A->B و A->C مرج می کنیم و می گیم میشه A->BC

- *F* logically implies all dependencies in F_c , and
- F_c logically implies all dependencies in *F,* and
- No functional dependency in F_c contains an extraneous attribute, and
- Each left side of functional dependency in F_c is unique. That is, there are no two dependencies in F_c
  - $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that
  - $\alpha_1 = \alpha_2$

مجموعه F اولیه رو تا حد ممکن کمش کنیم ینی F رو بکنیم Fc ینی Fc یک مجموعه مینیمالی است از همون مجموعه اولیه با این شرطی که یک نماینده ای است از همون مجموعه F و هیچ وابستگی تابعی رو miss نکنیم

پس F می خوایم بکنیم Fc ینی کوجیکش بکنیم بدون اینکه هیچ دانشی این وسط miss بشه یا دانش اضافیه داخلش باشه

این مجموعه Fc چهارتا قید داره:

1- ینی F اولیه باید بتونه به شکل منطقی تمام وابستگی که داخل Fc بوده رو به دست بیاره

2- Fc باید بتونه به شکل منطقی تمام وابستگی هایی که داخل F بوده رو بهش برسه

3- داخل Fc ما یک صفت اضافیه نداریم

4- سمت چپ مجموعه Fc باید یوینک باشه ینی الفا یک و الفا دو نباید نداشته باشیم که با هم یکی باشند

Canonical Cover

- To compute a canonical cover for *F*:

  repeat

  Use the union rule to replace any dependencies in *F* of the form

  $$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

  Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

  /* Note: test for extraneous attributes done using F_c, not F*/

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until (F_c not change

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

توی یه حلقه اولی میایم از قانون اجتماع می ریم
بعد صفت های اضافی رو از Fc حذف میکنیم و باز دوباره بررسی می کنیم چون با حذف شدن یک صفت از سمت چپ و راست ممکنه دوباره بتونیم از قانون اجتماع بریم و این کارو اینقدر انجام می دیم که دیگه نتونیم توی Fc تغییری ایجاد کنیم

Example: Computing a Canonical Cover

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

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

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

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

اگر بتونیم به یکی از این دو رابطه برسیم اون وقت میتونیم بگیم که R1,R2 یک تجزیه مناسبی هستند و دانشی رو این وسط از دست نمیدن

- 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

ی خوایم R رو تحزیه کنیم به R1,R2 بعد میخوایم با وابستگی تابعی مشخص کنیم در چه صورت این تجزیه دانشی رو از دست نمیده و یک lossless است؟
1- در صورتی که کل صفت های R1 وابستگی تابعی داشته باشند به اشتراک R1,R2
2- و همین طور برای R2 هم همینه

Example

اگر جوین بخوایم انجام بدیم روی صفت مشترک این کارو میکنیم

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

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

    این R2 است
    یا میتونیم بگیم AB ک این میشه R1
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless decomposition:

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

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

وابستگی تابعی و کلید

Keys and Functional Dependencies

- Given $r(R)$, a subset K of R is a **superkey** of $r(R)$ if, in any legal instance of $r(R)$, for all pairs $t1$ and $t2$ of tuples in the instance of r if $t1 \neq t2$, then $t1[K] \neq t2[K]$. That is, no two tuples in any legal instance of relation $r(R)$ may have the same value on attribute set K.

اگر تمام صفت های R وابستگی تابعی داشت به K می تونیم بگیم که k یک سوپر کلید است برای R

- 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 $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  in_dep (*ID, name, salary, dept_name, building, budget*).

  We expect these functional dependencies to hold:

  $$dept\_name \rightarrow building$$

  $$ID \rightarrow building$$

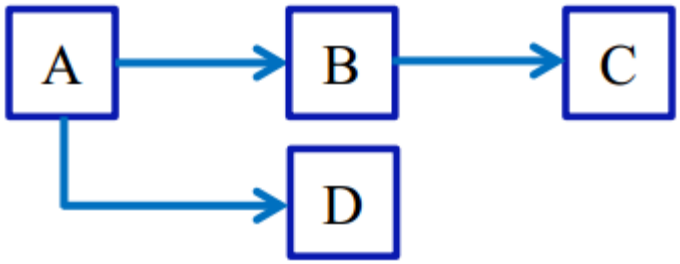
  but would not expect the following to hold:

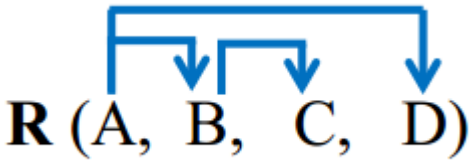
  $$dept\_name \rightarrow salary$$

Functional Dependencies

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$ وابستگی تابعی بدیهی:

$$\mathbf{F}=\{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow D\}$$





Dependency Preservation

- Let F be a set of functional dependencies on a schema R, and let $R1$, $R2$, … , Rn be a decomposition of R.
- The **restriction of** F **to** Ri is the set Fi of all functional dependencies in F^+ that include *only* attributes of Ri.
- Note that the definition of restriction uses all dependencies in F^+, not just those in F. For instance, suppose $F = \{A \rightarrow B, B \rightarrow C\}$, and we have a decomposition into AC and AB. The restriction of F to AC includes $A \rightarrow C$, since $A \rightarrow C$ is in F^+, even though it is not in F.
- Let $F' = F1 \cup F2 \cup \cdots \cup Fn$. F' is a set of functional dependencies on schema R, but, in general, $F' \neq F$. However, even if $F' \neq F$, it may be that $F'^+ = F^+$
- We say that a decomposition having the property $F'^+ = F^+$ is a **dependency-preserving decomposition**.

توی نرمال سازی باید بیایم تجزیه بکنیم جدول ها رو
توی تجزیه کردن باید حواسمون باشه که وابستگی تابعی هایی که داریم از بین نره موقع تجزیه

Dependency Preservation Example

- R = (A, B, C)
  F = {A → B
  B → C}
  Key = {A}

- Decomposition R1 = (A, B), R2 = (B, C)
  – Lossless-join decomposition
  – Dependency preserving

Good decomposition

Rissanen's Theorem: Let relvar *R*, with heading *H*, have projections *R1* and *R2*, with headings *H1* and *H2*, respectively; further, let *H1* and *H2* both be proper subsets of *H*, let their union be equal to *H*, and let their intersection not be empty.[15] Then projections *R1* and *R2* are **independent** if and only if (a) their common attributes constitute a superkey for at least one of them and (b) every FD that holds in *R* is implied by those that hold in at least one of them.

Second Normal Form

- A relation is in the second normal form if it fulfills the following two requirements:
  1. It is in first normal form.
  2. No Partial Functional Dependencies
     - Every non-key attribute must be fully functionally dependent on the entire key of that table.
     - A non-key attribute cannot depend on only part of the key.

- It does not have any non-prime attribute that is functionally 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.
- Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF.

Second Normal Form Example

STID	COID	FIELD	DEPT	GR
777	CO1	Phys	D11	19
777	CO2	Phys	D11	16
777	CO3	Phys	D11	11
888	CO1	Math	D12	16
888	CO2	Math	D12	18
444	CO1	Math	D12	13
555	CO1	Phys	D11	14
555	CO2	Phys	D11	12

- The attributes STID and COID are the identification keys.
- STID→FIELD
- STID→DEPT
- FIELD→DEPT

Second Normal Form Example

STID	COID	FIELD	DEPT	GR
777	CO1	Phys	D11	19
777	CO2	Phys	D11	16
777	CO3	Phys	D11	11
888	CO1	Math	D12	16
888	CO2	Math	D12	18
444	CO1	Math	D12	13
555	CO1	Phys	D11	14
555	CO2	Phys	D11	12

STID	COID	GR
777	CO1	19
777	CO2	16
777	CO3	11
888	CO1	16
888	CO2	18
444	CO1	13
555	CO1	14
555	CO2	12

STID	FIELD	DEPT
777	Phys	D11
888	Math	D12
444	Math	D12
555	Phys	D11

Third Normal Form(3NF)

- Requiring existence of "the key" ensures that the table is in 1NF
  • Requiring that non-key attributes be dependent on "the whole key" ensures 2NF
  • Further requiring that non-key attributes be dependent on "nothing but the key" ensures 3NF
- A relation is in the 3NF if it fulfills the following two requirements:
  1. It is in 2NF
  2. Does not allow transitive dependencies in which one non-key attribute is functionally dependent on another.
     - Non-key attributes are not allowed to define other non-key attributes.

3NF Example

STID	COID	GR
777	CO1	19
777	CO2	16
777	CO3	11
888	CO1	16
888	CO2	18
444	CO1	13
555	CO1	14
555	CO2	12

STID	FIELD	DEPT
777	Phys	D11
888	Math	D12
444	Math	D12
555	Phys	D11

STID→FIELD
STID→DEPT
FIELD→DEPT

STID	FIELD
777	Phys
888	Math
444	Math
555	Phys

FIELD	DEPT
Phys	D11
Math	D12

Boyce-Codd Normal Form(BCNF)

- 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

Boyce-Codd Normal Form (Cont.)

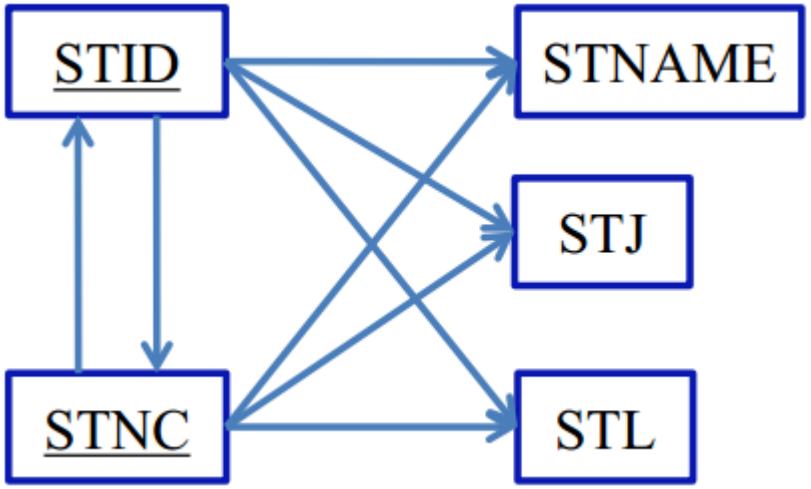
- Example schema that is ***not*** in BCNF:

  in_dep (*ID, name, salary, dept_name, building, budget*)

  because :

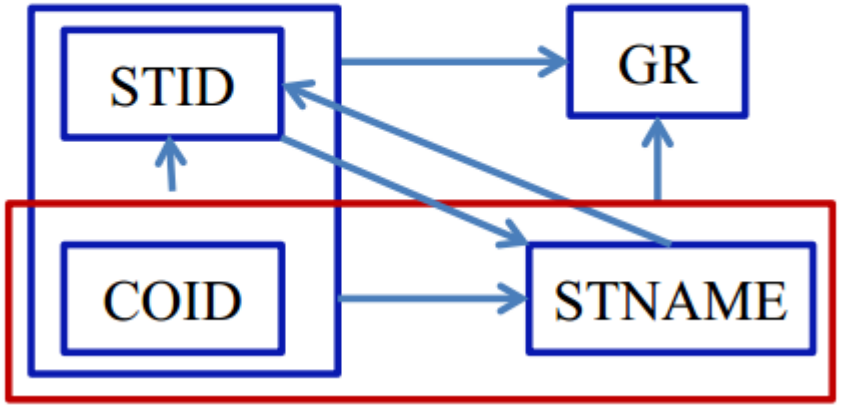
  - *dept_name* $\rightarrow$ *building, budget*
    - holds on *in_dep*
    - but
  - *dept_name* is not a superkey
- When decompose *in_dept* into *instructor* and *department*
  - *instructor* is in BCNF
  - *department* is in BCNF

Boyce-Codd Normal Form (Cont.)



SCNG (STID, COID, STNAME, GR)

C.K. C.K.



Decomposing a Schema into BCNF

- Let R be a schema *R* that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF.
- We decompose *R* into:
  - (α U β)
  - (*R* - (β - α))
- In our example : *in_dep* (*ID, name, salary, dept_name, building, budget*),
  - α = *dept_name*
  - β = *building, budget*

  and *in_dep* is replaced by
  - (α U β) = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, dept_name, salary*)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- *dept_advisor* is not in BCNF
  - *i_ID* is not a superkey.
- Any decomposition of *dept_advisor* will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$
- Thus, the composition is NOT be dependency preserving

Third Normal Form

- 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., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

    (**NOTE**: each attribute 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).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

3NF Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF
- *R,* however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$ *and* *i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* } and
    - *dept_name* is contained in a candidate key

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

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - The decomposition is lossless
  - The dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - The decomposition is lossless
  - It may not be possible to preserve dependencies.

End of Chapter 7