Database

Theory

Anomalies and Functional Dependencies

Case Study

Case

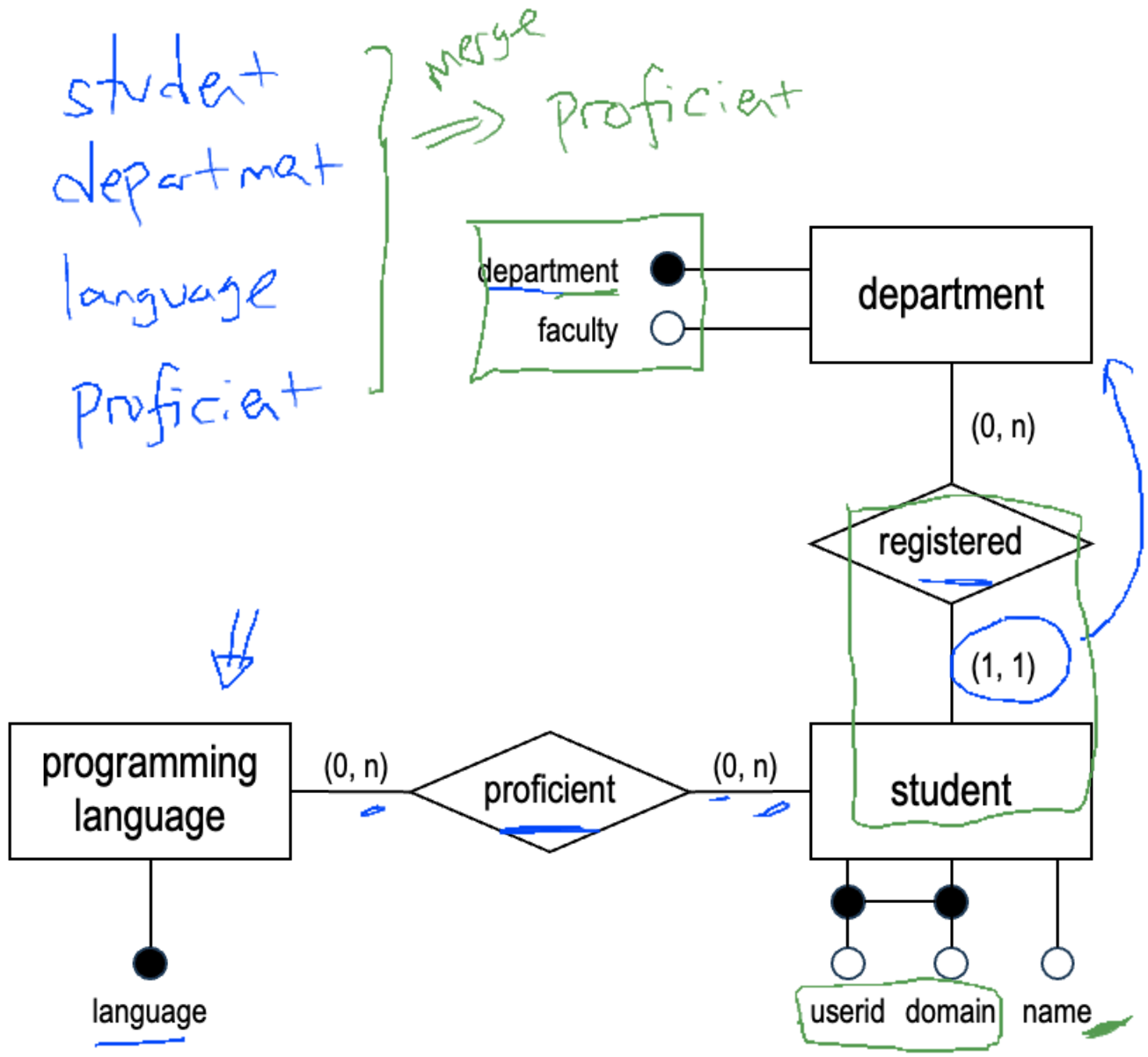
SUT

The Case

Sentosa University of Technology (SUT) records the the **programming language skills** of the students of its **different faculties** and **departments**.

The Schema

The schema should consist of 4 tables with `student` and `registered` merged.



Case Study

Case

Alternative

One Table

We store everything in one table (`proficiency`). What could go wrong?

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Case Study

Case

Alternative

One Table

We store everything in **one table** (`proficiency`). What could go wrong?

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Requirement

Each faculty is organized into several departments.

A department **belongs to one faculty** only and there is **no two departments with the same name** in SUT.

Case Study

Case

Alternative

One Table

We store everything in **one table** (`proficiency`). What could go wrong?

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Requirement

Students are **identified by their email**. The email of a student is composed of her **userid and domain**.

We record the primary department of the student.

Case Study

Case

Alternative

One Table

We store everything in **one table** (`proficiency`). What could go wrong?

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

student department faculty lang

Requirement

For each student, the database records *(only once)* **each programming language** in which the student is proficient.

Case Study

Anomalies

Redundant

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	**computer science**	**computing**	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	**computer science**	**computing**	Python
Tan Hee Wee	tanh	comp.sut.edu	**computer science**	**computing**	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Redundant Storage

The faculty of a department is repeated for every student of the department and every time the student is proficient in a programming language.

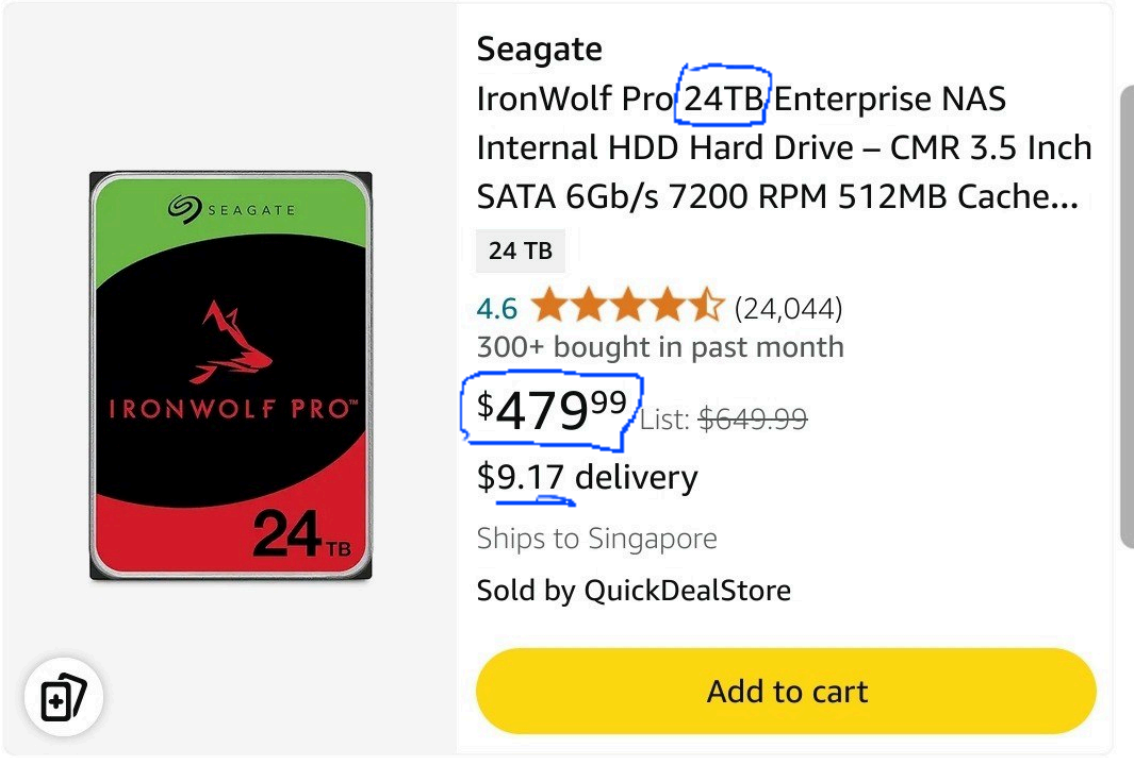
Case Study

Anomalies

Redundant

Issue?

IBM 726 *(rental USD 850 per month in 1952)*



Case Study

Anomalies

Update

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	**computer science**	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	**computer science**	informatics	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Update Anomaly

```
UPDATE proficiency SET faculty='informatics' WHERE language='C++' AND domain='comp.sut.edu'
```

We may **forget** to update all faculties for the same department.

Case Study

Anomalies

Delete

name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	informatics	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
~~Ami Mokhtar~~	~~ami~~	~~med.sut.edu~~	~~pharmacy~~	~~medicine~~	~~Rust~~

Deletion Anomaly

```
DELETE FROM proficiency WHERE userid='ami' AND domain='med.sut.edu'
```

We may **forget** we even have faculty of medicine.

Case Study

Anomalies

Insert

name A	userid B	domain C	department D	faculty E	language F
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	informatics	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
→ NULL	NULL	NULL	**pharmacy**	**medicine**	NULL

Insertion Anomaly

We cannot record medicine unless there is a student because we cannot insert `NULL` due to some columns being `PRIMARY KEY`*.

*Can you guess what should be the primary key of the table?

Case Study

Anomalies

Source

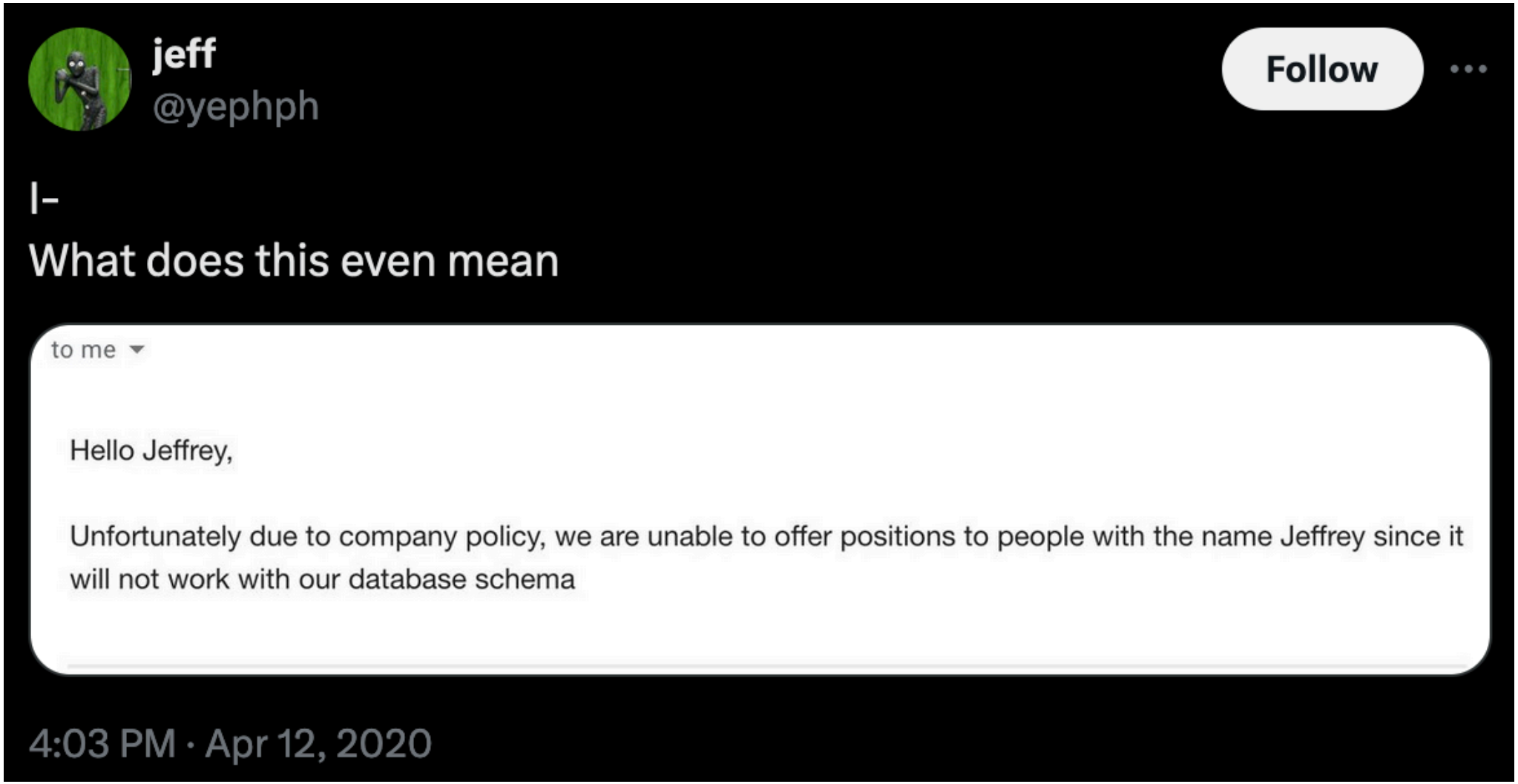
name	userid	domain	department	faculty	language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Source of Anomalies

The source of these anomalies is because some columns are **uniquely determined** by other columns. This can be translated into an integrity constraint called **functional dependencies**.

Case Study

Anomalies



Functional Dependencies

FD

Basic

Definition

An instance r of a relation schema R *satisfies* the functional dependencies σ of the form $X \rightarrow Y$ with $X \subseteq R$ and $Y \subseteq R^*$, if and only if two tuples of r *agree on their X-values*, then they *agree on their Y-values*.

Terminology

$X \rightarrow Y$ reads:

X *functionally determines* Y
Y is *functionally dependent* on X

X *determines* Y
X *implies* Y *(casually)*

*We abuse the notation a little, by $X \subseteq R$ we meant the set of columns from schema R.
So we can write it as either $R(A, B, C, D)$ or $R = \{A, B, C, D\}$.

Functional Dependencies

FD

Basic

Definition

An instance r of a relation schema R *satisfies* a set of functional dependencies Σ if and only if it *satisfies all the functional dependencies* σ in the set Σ.

Valid Instance

An instance r of a relation schema R is a *valid instance* of R with Σ if and only if it *satisfies* Σ.

In other words, there are no functional dependencies $\sigma \in \Sigma$ that are violated *(i.e., not satisfied)*. $X \rightarrow Y$ is violated if there are two tuples of r *agree on their X-values*, BUT *do not agree on their Y-values*.

Functional Dependencies

FD

Holds

Definition

A relation R with a **set of functional dependencies Σ** --denoted as ***R with Σ***-- refers to the set of valid instances of R ***with respect to*** the functional dependencies in Σ.

Holds

We say that a set of functional dependencies Σ ***holds*** on a relation R

Note that in this case, we ***only consider the valid instances*** of R with Σ. If Σ holds on R, then there cannot be violation and consequently, every instance r must be a valid instance of R.

Functional Dependencies

FD

Example

Let $R = \{A, B, C, D\}$ and $\sigma = \{A, B\} \rightarrow \{D\}$

Valid

A	B	C	D
1	**2**	a	**4**
1	**2**	b	**4**
1	3	c	4

Violate

A	B	C	D
1	**2**	a	**4**
1	**2**	b	**3**
1	3	c	4

Functional Dependencies

FD

Example

Let $R = \{A, B, C, D\}$ and $\sigma = \{A, B\} \rightarrow \{D\}$

Valid

A	B	C	D

Violate

A	B	C	D
1	2	a	4
1	2	b	3

Note

Empty instance is always a *(vacuously)* valid instance.

Note

Both examples are the smallest instance.

Functional Dependencies

Triviality

Trivial

Definition

A functional dependency $\sigma : X \rightarrow Y$ is ***trivial*** if and only if $Y \subseteq X$. In other words, a set X always ***uniquely determine*** its subset X.

Let $R = \{A, B, C\}$

$\{A\} \rightarrow \{A\}$	is trivial
$\{A,B\} \rightarrow \{A\}$	is trivial
$\{A,B\} \rightarrow \{\}$	is trivial *(also denoted as $\{A,B\} \rightarrow \varnothing$)*

Functional Dependencies

Triviality

Non-Trivial

Definition

A functional dependency $\sigma : X \rightarrow Y$ is ***non-trivial*** if and only if $Y \not\subseteq X$. Literally, the ***negation*** of the defintion of trivial.

Let $R = \{A, B, C\}$

$\{A\} \rightarrow \{B\}$	is non-trivial
$\{A,C\} \rightarrow \{B,C\}$	is non-trivial
$\{\} \rightarrow \{A,B\}$	is non-trivial *(also denoted as $\varnothing \rightarrow \{A,B\}$)*

Functional Dependencies

Triviality

Completely

Definition

A functional dependency $\sigma : X \rightarrow Y$ is ***completely non-trivial*** if and only if $Y \neq \varnothing$ and $Y \cap X = \varnothing$. In other words, ***they have no common attributes***.

Let $R = \{A, B, C\}$

$\{A\} \rightarrow \{B\}$	is completely non-trivial
$\{A,C\} \rightarrow \{B,C\}$	is **not** completely non-trivial but still non-trivial
$\{A,B\} \rightarrow \{\}$	is **not** completely non-trivial as it is trivial

Theorem #1

A non-trivial *(but not completely non-trivial)* functional dependency can be split into a trivial functional dependency and a completely non-trivial functional dependency.

Functional Dependencies

Key

Superkey

Definition

Let R be a relation. Let $S \subseteq R$ be a **set of attributes** of R. S is a ***superkey*** of R if and only if $S \rightarrow R$. A superkey is a ***superset of a key*** *(a key or candidate key is defined later)*.

In Other Words

A **superkey** is a set of attributes of a relation whose knowledge determines the value of the entire tuple.

Theorem #2

A relation R have *at least 1* superkey.

Functional Dependencies

Key

Candidate Key

Definition

Let R be a relation. Let $S \subseteq R$ be a **set of attributes** of R. S is a ***candidate key*** of R if and only if $S \rightarrow R$ and **for all $T \subset S$, T is NOT a superkey of R**.

In Other Words

A **candidate key** is a ***minimal**** superkey.

The **primary key** is the candidate key that the designer prefers.

*By **minimal** superkey, we meant that if any attribute of a candidate key is removed, the set is **no longer a superkey**.

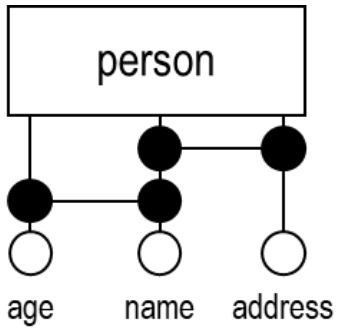
Functional Dependencies

Key

Candidate Key

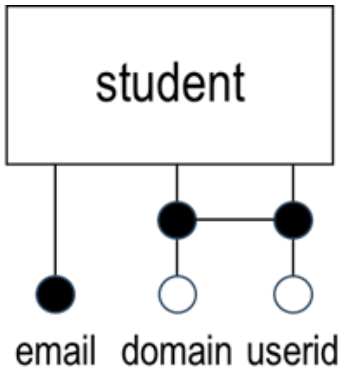
Definition

Let R be a relation. Let $S \subseteq R$ be a **set of attributes** of R. S is a ***candidate key*** of R if and only if $S \rightarrow R$ and **for all $T \subset S$, T is NOT a superkey of R.**



Candidate Keys

{ {age, name}
{name, address} }



Candidate Keys

{ {email}
{domain, userid} }

Functional Dependencies

Key

Prime Attribute

Definition

Let Σ be a **set of functional dependencies** on a relation schema R. A ***prime attribute*** is an attribute that ***appears in some candidate key*** of R **with** Σ.

Note

Only need to appear in at least **one** candidate key. In other words, if there are multiple candidate keys K_1, K_2, ..., K_n, the prime attribute is $K_1 \cup K_2 \cup \ldots \cup K_n$.

Otherwise

Attributes that are NOT prime attributes are called **non-prime attributes**.

Functional Dependencies

Key

Example

Let $R = \{A, B, C, D\}$ and $\Sigma = \{ \{A,B\} \rightarrow \{C,D\}, \{C\} \rightarrow \{A,B\} \}$

Superkeys

$\{A, B, C, D\}$
$\{A, B\}, \{C, D\}$
$\{C\}$ *etc...*

Candidate Keys

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

Prime Attributes

A
B
C *(i.e., {A, B, C})*

Functional Dependencies

Question

How do we know $\{B\} \rightarrow \{A\}$ does NOT hold?

Case

Instance

(A) name	(B) userid	(C) domain	(D) department	(E) faculty	(F) language
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	JavaScript
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	Python
Tan Hee Wee	tanh	comp.sut.edu	computer science	computing	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	C++
Tan Hee Wee	tanhw	eng.sut.edu	computer engineering	engineering	Fortran
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	C++
Bjorn Sale	bjorn	eng.sut.edu	computer engineering	engineering	Java
Ami Mokhtar	ami	med.sut.edu	pharmacy	medicine	Rust

Hold

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

Do Not Hold

$\{B\} \rightarrow \{A\}$
$\{F\} \rightarrow \{E\}$
$\{C,E\} \rightarrow \{B\}$ *(how do we know?)*

Functional Dependencies

Case

ERD

The Case

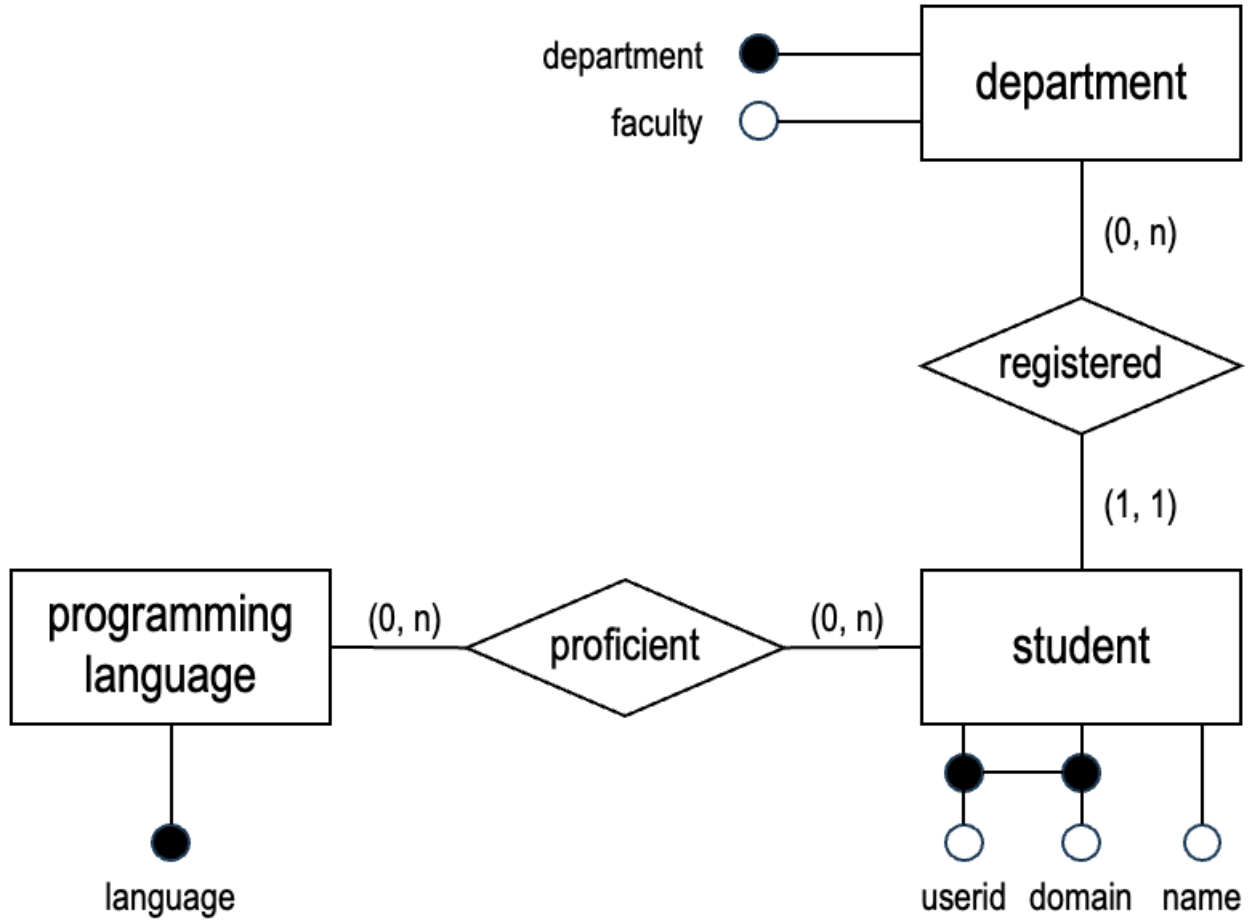
Sentosa University of Technology (SUT) records the the **programming language skills** of the students of its **different faculties** and **departments**.

The Schema

The schema should consist of 4 tables with `student` and `registered` merged.

Answer

`userid` does not uniquely identify `domain`.



Break

Closures

Closure

Closure of Σ

Note

This is the **closure** of the *set of functional dependencies*. We have a simpler closure later.

Definition

Let Σ be a set of functional dependencies of a relation schema *R*. The **closure of Σ** --denoted by Σ^+-- is the **set of all functional dependencies *logically entailed*** by the functional dependencies in Σ.

Example

Let $R = \{A, B, C, D\}$ and $\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$

$\Sigma^+ = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\}, \{A\} \rightarrow \{A\}, \{D\} \rightarrow \{D\}, \{A,B\} \rightarrow \{A\}, \{A,C\} \rightarrow \{B,C\}, \{A,D\} \rightarrow \{B\},$
$\{C\} \rightarrow \{B\}, \ldots \}$

Closures

Closure

Proof?

Example

Let $R = \{A, B, C, D\}$ and $\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$

$\{C\} \rightarrow \{B\} \in \Sigma^+$?

1. Assume $\{C\} \rightarrow \{B\}$ does not hold.
   Then there are two rows $(A_1, B_1, \mathbf{C}, D_1)$ and $(A_2, B_2, \mathbf{C}, D_2)$ such that $B_1 \neq B_2$.

2. Since $\{C\} \rightarrow \{A\}$ holds, then $A_1 = A_2$ because $C = C$.
   We rewrite both as $\mathbf{A}$ to get $(\mathbf{A}, B_1, \mathbf{C}, D_1)$ and $(\mathbf{A}, B_2, \mathbf{C}, D_2)$.

3. Since $\{A\} \rightarrow \{B\}$ holds, then $B_1 = B_2$ because $A = A$.
   We rewrite both as $\mathbf{B}$ to get $(\mathbf{A}, \mathbf{B}, \mathbf{C}, D_1)$ and $(\mathbf{A}, \mathbf{B}, \mathbf{C}, D_2)$.

4. But this contradict (1) that states $B_1 \neq B_2$.
5. So the assumption (1) must be wrong and we conclude $\{C\} \rightarrow \{B\}$ holds.

Closures

Closure

Closure of S

Definition

Let **Σ** be a set of functional dependencies of a relation schema R. The **closure of $S \subseteq R$** *(i.e., a set of attribute)* --denoted by S^+-- is the **set of all attributes *functionally dependent*** on S.

Example

Let $R = \{A, B, C, D\}$ and $\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$

$\{A\}^+ = \{A, B\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{A, B, C\}$

$\{C,D\}^+ = \{A, B, C, D\}$

Note

Because S^+ is *functionally dependent* on S, we have

$$S \rightarrow S^+$$

In fact, for all $Y \subseteq S^+$, we have

$$S \rightarrow Y$$

Closures

Armstrong

Rules

Armstrong Axioms

Let R be a set of attributes. The following **inference rules** are the **Armstrong Axiom**.

- **Reflexivity**

  $\forall X \subseteq R \; \forall Y \subseteq R \; ((Y \subseteq X) \Rightarrow (X \rightarrow Y))$

  Trivial functional dependencies are always valid

- **Augmentation**

  $\forall X \subseteq R \; \forall Y \subseteq R \; \forall Z \subseteq R \; ((X \rightarrow Y) \Rightarrow ((X \cup Z) \rightarrow (Y \cup Z)))$

  You can add the same attribute to both left hand and right hand side ***at the same time***

- **Transitivity**

  $\forall X \subseteq R \; \forall Y \subseteq R \; \forall Z \subseteq R \; ((X \rightarrow Y \wedge Y \rightarrow Z) \Rightarrow (X \rightarrow Z))$

  You can short-circuit the arrows

*Out of scope. We put this for completeness of explanation only.

Closures

Armstrong

Example

Question

Let $R = \{A, B, C, D\}$ and $\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$. Can we prove that $\{C,D\} \rightarrow \{B,A\} \in \Sigma^+$?

Proof

1. We know that $\{A\} \rightarrow \{B\}$. (*given in Σ*)
2. We know that $\{C\} \rightarrow \{A\}$. (*given in Σ*)
3. Therefore $\{A\} \rightarrow \{A,B\}$. (*Augmentation of (1) with $\{A\}$*)
4. Therefore $\{C\} \rightarrow \{A,B\}$. (*Transitivity of (2) and (3)*)
5. Therefore $\{C,D\} \rightarrow \{A,B,D\}$. (*Augmentation of (4) with $\{D\}$*)
6. Therefore $\{A,B,D\} \rightarrow \{A,B\}$. (*Reflexivity since $\{A,B\} \subseteq \{A,B,D\}$*)
7. Therefore $\{C,D\} \rightarrow \{A,B\}$. (*Transitivity of (5) and (6)*) □

*Out of scope. We put this for completeness of explanation only.

Closures

Armstrong

Theorems

Theorem #3: Soundness

The **Reflexivity** inference rule is *sound (i.e., correct, valid)*.
The **Augmentation** inference rule is *sound (i.e., correct, valid)*.
The **Transitivity** inference rule is *sound (i.e., correct, valid)*.

All functional dependencies obtained using Armstrong axioms are guaranteed to hold.

Theorem #4: Completeness

The **Armstrong axiom** inference rule is *complete*.

All functional dependencies that holds can be obtained using Armstrong axioms.

Note

What we are interested in is the **consequence** of the theorem. We can formalize this is as ***algorithm***.

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

$\Omega := \Sigma$; // Ω stands for "unused"

$\Gamma := S$; // Γ stands for "closure"

while $(X \rightarrow Y \in \Omega)$ ***and*** $(X \subseteq \Gamma)$ **do**

$\Omega := \Omega - \{ X \rightarrow Y \}$;

$\Gamma := \Gamma \cup Y$;

return Γ

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

$\Omega := \Sigma$; // Ω stands for "unused"

$\Gamma := S$; // Γ stands for "closure"

while $(X \rightarrow Y \in \Omega)$ ***and*** $(X \subseteq \Gamma)$ **do**

$\Omega := \Omega - \{ X \rightarrow Y \}$;

$\Gamma := \Gamma \cup Y$;

return Γ

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$
2. use $\{C\} \rightarrow \{A\}$ $(\{C\} \subseteq \Gamma)$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$
2. use $\{C\} \rightarrow \{A\}$ $(\{C\} \subseteq \Gamma)$
   $\Omega = \{ \{A\} \rightarrow \{B\} \}$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$
2. use $\{C\} \rightarrow \{A\}$ $(\{C\} \subseteq \Gamma)$
   $\Omega = \{ \{A\} \rightarrow \{B\} \}$
   $\Gamma = \{C\} \cup \{A\} = \{A, C\}$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$
2. use $\{C\} \rightarrow \{A\}$ $(\{C\} \subseteq \Gamma)$
   $\Omega = \{ \{A\} \rightarrow \{B\} \}$
   $\Gamma = \{C\} \cup \{A\} = \{A, C\}$
3. use $\{A\} \rightarrow \{B\}$ $(\{A\} \subseteq \Gamma)$
   $\Omega = \varnothing$
   $\Gamma = \{A, C\} \cup \{B\} = \{A, B, C\}$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

```
Ω := Σ;      // Ω stands for "unused"
Γ := S;      // Γ stands for "closure"
while (X → Y ∈ Ω) and (X ⊆ Γ) do
    Ω := Ω - { X → Y };
    Γ := Γ ∪ Y;
return Γ
```

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

1. $\Omega = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$
   $\Gamma = \{C\}$
2. use $\{C\} \rightarrow \{A\}$ $(\{C\} \subseteq \Gamma)$
   $\Omega = \{ \{A\} \rightarrow \{B\} \}$
   $\Gamma = \{C\} \cup \{A\} = \{A, C\}$
3. use $\{A\} \rightarrow \{B\}$ $(\{A\} \subseteq \Gamma)$
   $\Omega = \varnothing$
   $\Gamma = \{A, C\} \cup \{B\} = \{A, B, C\}$
4. return $\Gamma = \{A, B, C\}$

Closures

Algorithm

Attribute Closure

Algorithm #1: Attribute Closure

input S, Σ

output S^+

begin

$\Omega := \Sigma$; // Ω stands for "unused"

$\Gamma := S$; // Γ stands for "closure"

while $(X \rightarrow Y \in \Omega)$ ***and*** $(X \subseteq \Gamma)$ **do**

$\Omega := \Omega - \{ X \rightarrow Y \}$;

$\Gamma := \Gamma \cup Y$;

return Γ

Question

Let $R = \{A, B, C, D\}$
$\Sigma = \{ \{A\} \rightarrow \{B\}, \{C\} \rightarrow \{A\} \}$.

Compute $\{C\}^+$ using **Algorithm #1**.

Equivalence

Equivalent

Equivalent Σ

Definition

Two sets of functional dependencies Σ_1 and Σ_2 are ***equivalent*** if and only if they have the **same closure**.

More formally, $\Sigma_1 \equiv \Sigma_2 \Leftrightarrow \Sigma^+_1 = \Sigma^+_2$.

Cover

Σ_1 is a ***cover*** of Σ_2 *(and Σ_2 is a cover of Σ_1)* if and only if $\Sigma_1 \equiv \Sigma_2$.

Equivalence

Equivalent

Example

Question

Let $R = \{A, B, C\}$. Are the following set of functional dependencies equivalent?

$\Sigma_1 = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\} \}$

$\Sigma_2 = \{ \{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\} \}$

Equivalence

Equivalent

Example

Question

Let $R = \{A, B, C\}$. Are the following set of functional dependencies equivalent?

$\Sigma_1 = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\} \}$

$\Sigma_2 = \{ \{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\} \}$

- $\Sigma^+_1 = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\},$
  $\{A\} \rightarrow \{B,C\}, \{B\} \rightarrow \{A,C\}, \{C\} \rightarrow \{A,B\},$
  $\{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\}$, ... $\}$
- $\Sigma^+_2 = \{ \{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\},$
  $\{A\} \rightarrow \{B,C\}, \{B\} \rightarrow \{A,C\}, \{C\} \rightarrow \{A,B\},$
  $\{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\}$, ... $\}$

Equivalence

Equivalent

Example

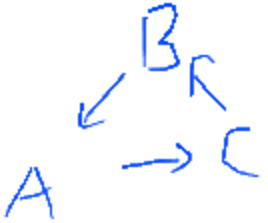
Question

Let $R = \{A, B, C\}$. Are the following set of functional dependencies equivalent?

$\Sigma_1 = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\} \}$

$\Sigma_2 = \{ \{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\} \}$

- $\Sigma^+_1 = \{ \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\},$
  $\{A\} \rightarrow \{B,C\}, \{B\} \rightarrow \{A,C\}, \{C\} \rightarrow \{A,B\},$
  $\{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\}, \ldots \}$
- $\Sigma^+_2 = \{ \{A\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{A\},$
  $\{A\} \rightarrow \{B,C\}, \{B\} \rightarrow \{A,C\}, \{C\} \rightarrow \{A,B\},$
  $\{A\} \rightarrow \{B\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{A\}, \ldots \}$



Equivalence

Equivalent

Theorem

Theorem #5

Two sets of functional dependencies Σ_1 and Σ_2 are ***equivalent*** if and only if all of the following condition is satisfied.

- All functional dependency σ in Σ_1 is a ***logical consequence*** of the functional dependencies in Σ_2.
- All functional dependency σ in Σ_2 is a ***logical consequence*** of the functional dependencies in Σ_1.

Equivalence

Cover

Minimal

Definition

A set Σ of functional dependencies is ***minimal*** if and only if

- The **right hand side** of every functional dependency Σ is **minimal**.
  i.e., every functional dependency is of the form $X \rightarrow \{A\}$.
- The **left hand side** of every functional dependency Σ is **minimal**.
  i.e., for every functional dependency in Σ of the form $X \rightarrow \{A\}$
  there is no functional dependency $Y \rightarrow \{A\}$ in Σ^+ such that $Y \subset X$.
- The **set itself** is **minimal**.
  i.e., none of the functional dependency in Σ can be derived from other functional dependencies in Σ.

Σ is a ***minimal cover*** of Σ' if and only if Σ is both **minimal** and a **cover** for Σ'.

Equivalence

Cover

Minimal

In Other Words

A set Σ of functional dependencies is ***minimal*** if and only if

- The **right hand side** of every functional dependency Σ only has one attribute.
- Cannot remove any attribute from the **left hand side**.
- Cannot remove any functional dependencies from Σ.

Equivalence

Cover

Canonical

Definition

A set Σ is a ***canonical cover*** of Σ' if there is a Σ_1 that is a **minimal cover** of Σ' and we produce Σ by **regrouping** all the functional dependencies with the same left hand side in Σ_1.

Note

In other words, we find **minimal cover** and then we *merge* the right hand side if the left hand side are equal.

- $\Sigma = \{ \{A\} \rightarrow \{B\}, \{A\} \rightarrow \{C\} \}$ is **minimal** but **not** canonical.
- $\Sigma = \{ \{A\} \rightarrow \{B,C\} \}$ is **canonical** *(but is it minimal?)*.

Equivalence

Cover

Theorem

Theorem #6

Every set of functional dependencies has at least one **minimal cover** *(also called as minimal basis)*. The minimal cover **may not be unique**.

Equivalence

Cover

Algorithm

Algorithm #2: Minimal Cover

input Σ
output Σ_C

1. Simplify *(minimize)* the **right hand side** of every functional dependency in Σ
   produce Σ_A
2. Simplify *(minimize)* the **left hand side** of every functional dependency in Σ_A
   produce Σ_B
3. Simplify *(minimize)* the **set** Σ_B to produce Σ_C
4. Return Σ_C

Equivalence

Cover

Algorithm

Algorithm #3: Canonical Cover

input Σ
output Σ_D

1. Simplify *(minimize)* the **right hand side** of every functional dependency in Σ
   produce Σ_A
2. Simplify *(minimize)* the **left hand side** of every functional dependency in Σ_A
   produce Σ_B
3. Simplify *(minimize)* the **set** Σ_B to produce Σ_C
4. **Regroup** all functional dependencies ***with the same left hand side*** in Σ_C
   produce Σ_D
5. Return Σ_D

Equivalence

Cover

Simplification

{A, C, D} → {A}

{C, D}⁺ = X

{C, D} → {A} ✓

{A} ⊆ X

Step #2: Simplifying Left Hand Side

Let $X \rightarrow \{A\}$ be a functional dependency in Σ'. Attribute $B \in X$ **can be removed** from X if

$$(X - \{B\}) \rightarrow \{A\} \text{ is } \underline{\textit{logically entailed}} \text{ by } \Sigma'$$

Then we can **replace** $X \rightarrow \{A\}$ by $(X - \{B\}) \rightarrow \{A\}$ in Σ'

Step #3: Simplifying the Set

Let $X \rightarrow Y$ be a functional dependency in Σ'. It **can be removed** from **Σ'** if

$$X \rightarrow Y \text{ is } \underline{\textit{logically entailed}} \text{ by } (\Sigma' - \{X \rightarrow Y\})$$

Then we can **replace Σ'** by $(\Sigma' - \{X \rightarrow Y\})$.

Σ = { X → {A}, Y → {B}, Z → {C} }

Σ' =

Hitung X^+ w.r.t. Σ'

{A} ⊆ X^+

Equivalence

Cover

Simplification

Step #2: Simplifying Left Hand Side

Consider $X \rightarrow \{A\}$ in Σ'. Consider an attribute $B \in X$ and construct $(X - \{B\}) \rightarrow \{A\}$.

Compute $(X - \{B\})^+$ *with respect to* Σ'

If $A \in (X - \{B\})^+$, then we can **replace** $X \rightarrow \{A\}$ by $(X - \{B\}) \rightarrow \{A\}$ in Σ'

Step #3: Simplifying the Set

Consider $X \rightarrow Y$ in Σ'. Consider $\Sigma'' = \Sigma' - \{X \rightarrow Y\}$.

Compute X^+ *with respect to* by Σ''

Then we can **replace** Σ' by $\Sigma'' = (\Sigma' - \{X \rightarrow Y\})$.

Work

Worked Example

Question

$R = \{A, B, C, D, E\}$

$\Sigma = \{ \{A,B\} \rightarrow \{C,D,E\}, \{A,C\} \rightarrow \{B,D,E\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{B\}, \{C\} \rightarrow \{D\}, \{B\} \rightarrow \{E\}, \{C\} \rightarrow \{E\} \}$

1. Find all the candidate keys.
2. Find a minimal cover.
3. Find a canonical cover.

Work

Candidate Key

Compute Closures of Singletons

$\{A\}^+ = \{A\}$

$\{B\}^+ = \{B,C,D,E\}$

$\{C\}^+ = \{B,C,D,E\}$

$\{D\}^+ = \{D\}$

$\{E\}^+ = \{E\}$

Tips #1

Always start from the smallest cardinality *(i.e., size)*.

Tips #2

If any attribute does not appear on the right hand side, it must be part of all candidate keys.

Work

Candidate Key

Compute Closures of Pairs

$\{A,B\}^+ = \mathbf{\{A,B,C,D,E\}}$ *(a candidate key)*

$\{A,C\}^+ = \mathbf{\{A,B,C,D,E\}}$ *(a candidate key)*

$\{A,D\}^+ = \{A,D,E\}$

$\{A,E\}^+ = \{A,E\}$

~~$\{B,C\}^+ = \{B,C,D,E\}$~~

~~$\{B,D\}^+ = \{B,C,D,E\}$~~

~~$\{B,E\}^+ = \{B,C,D,E\}$~~

~~$\{C,D\}^+ = \{B,C,D,E\}$~~

~~$\{C,E\}^+ = \{B,C,D,E\}$~~

~~$\{D,E\}^+ = \{D,E\}$~~

Note

Any set of attributes containing {A,B} or {A,C} is a superkey but not a key.

Work

Candidate Key

Compute Closures of Triples

$\{A,D,E\}^+ = \{A,D,E\}$

~~$\{B,C,D\}^+ = \{B,C,D,E\}$~~

~~$\{B,C,E\}^+ = \{B,C,D,E\}$~~

~~$\{B,D,E\}^+ = \{B,C,D,E\}$~~

~~$\{C,D,E\}^+ = \{B,C,D,E\}$~~

Answer

There are no more set of attributes containing *A* but not superset of {A,B} or {A,C}.

Answer: {A,B} and {A,C} are the only candidate keys

Work

Minimal Cover

Original

$$
\begin{aligned}
\Sigma \; = \{ & \\
& \{A,B\} \rightarrow \{C,D,E\}, \\
& \{A,C\} \rightarrow \{B,D,E\}, \\
& \{B\} \rightarrow \{C\}, \\
& \{C\} \rightarrow \{B\}, \\
& \{C\} \rightarrow \{D\}, \\
& \{B\} \rightarrow \{E\}, \\
& \{C\} \rightarrow \{E\} \\
\} &
\end{aligned}
$$

Work

Minimal Cover

Step #1: Simplify Right Hand Side (easy)

Σ_1 = {

{A,B} → {C},
{A,B} → {D},
{A,B} → {E},
{A,C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

from {A,B} → {C,D,E}
from {A,B} → {C,D,E}
from {A,B} → {C,D,E}
from {A,C} → {B,D,E}
from {A,C} → {B,D,E}
from {A,C} → {B,D,E}

Work

Minimal Cover

Step #2: Simplify Left Hand Side (hard)

Σ_2 = {
{A,B} → {C},
{A,B} → {D},
{A,B} → {E},
{A,C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Work

Minimal Cover

Step #2: Simplify Left Hand Side (hard)

Σ_2 = {

~~{A,B} → {C},~~	*replaced with {B} → {C}*
~~{A,B} → {D},~~	*replaced with {B} → {D}*
~~{A,B} → {E},~~	*replaced with {B} → {E}*
~~{A,C} → {B},~~	*replaced with {C} → {B}*
~~{A,C} → {D},~~	*replaced with {C} → {D}*
~~{A,C} → {E},~~	*replaced with {C} → {E}*

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Work

Minimal Cover

Step #3: Simplify Set (medium)

$\Sigma_3 = \{$

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

$\}$

Work

Minimal Cover

Step #3: Simplify Set (medium)

Σ_3 = {

~~{B} → {C},~~	*duplicate*
{B} → {D},	
~~{B} → {E},~~	*duplicate*
~~{C} → {B},~~	*duplicate*
~~{C} → {D},~~	*duplicate*
~~{C} → {E},~~	*duplicate*
{B} → {C},	
{C} → {B},	
{C} → {D},	
{B} → {E},	
{C} → {E}	

}

Work

Minimal Cover

Step #3: Simplify Set (medium)

Σ_3 = {

{B} → {D},

{B} → {C},

{C} → {B},

{C} → {D},

{B} → {E},

{C} → {E}

}

Work

Minimal Cover

Step #3: Simplify Set (medium)

Σ_3 = {

~~{B} → {D}~~, *{B} → {C} and {C} → {D}*

{B} → {C},

{C} → {B},

{C} → {D},

~~{B} → {E}~~, *{B} → {C} and {C} → {E}*

{C} → {E}

}

Work

Minimal Cover

One Possible Answer

$\Sigma_3 = \{$

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

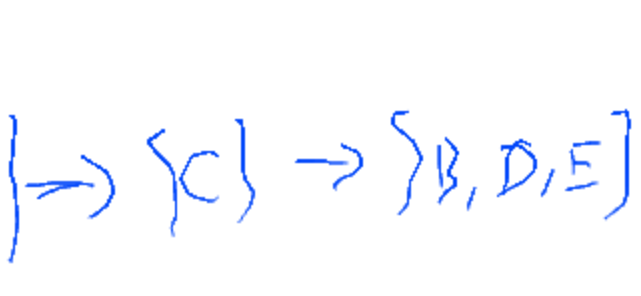
$\}$

Work

Minimal Cover

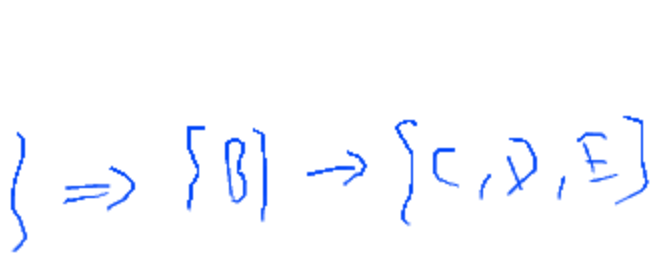
One Possible Answer

Σ_3 = {
  {B} → {C},
  {C} → {B},
  {C} → {D},
  {C} → {E}
}



Other Possible Answers

Σ_3 = {
  {C} → {B},
  {B} → {C},
  {B} → {D},
  {B} → {E}
}



Σ_3 = {
  {C} → {B},
  {B} → {C},
  {B} → {D},
  {C} → {E}
}

Note

We could reach different minimal covers by considering the constraints in a different order.

Work

Canonical Cover

One Possible Answer

$\Sigma_4 = \{ \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{B,D,E\} \}$

Other Possible Answers

$\Sigma_4 = \{ \{C\} \rightarrow \{B\}, \{B\} \rightarrow \{C,D,E\} \}$

$\Sigma_4 = \{ \{B\} \rightarrow \{C,D\}, \{C\} \rightarrow \{B,E\} \}$

$\Sigma_4 = \{ \{B\} \rightarrow \{C,E\}, \{C\} \rightarrow \{B,D\} \}$

```
postgres=# exit
Press any key to continue . . .
```

Proof Sketch

Only for Reading ; Not Tested

Proof Sketch

Theorem #1

Proof

Let $X \rightarrow Y$ be a non-trivial but not completely non-trivial functional dependency. Then

- $X \rightarrow (Y - X)$ is completely non-trivial
  $\Rightarrow (Y - X)$ is non-empty because $X \rightarrow Y$ is not completely non-trivial
  $\Rightarrow X \cap (Y - X) = \varnothing$ by definition

- $X \rightarrow (Y \cap X)$ is trivial
  $\Rightarrow (Y \cap X) \subseteq X$ by definition

So we can split $X \rightarrow Y$ into $X \rightarrow (Y - X)$ and $X \rightarrow (Y \cap X)$.

Proof Sketch

Theorem #1

Addendum: Why Can We Split?

The proof utilizes Armstrong axioms. Re-read the proof once you have covered Armstrong axiom. We want to show that if $X \rightarrow Y$ holds, then for all $Z \subseteq Y$, $X \rightarrow Z$ holds.

1. We know that $X \rightarrow Y$. (*given in Σ*)
2. Therefore $Y \rightarrow Z$. (*Reflexivity since $Z \subseteq Y$*)
3. Therefore $X \rightarrow Z$. (*Transitivity of (1) and (2)*) □

Proof Sketch

Theorem #2

Proof

It is sufficient to show that at least one set $X \subseteq R$ exists such that $X \rightarrow R$. We omit the full proof but it is quite easy to see that $R \rightarrow R$ holds.

Note that relation is defined using set. So there are no duplicate rows.

Proof Sketch

Theorem #3

Proof

We will sketch a proof for the transitivity property.

1. Let Σ be a set of functional dependencies on a relation schema R.
   Let $X \rightarrow Y$ and $Y \rightarrow Z$ be in Σ.
2. We know that for all valid instance r of R with Σ
   $$\forall t_1 \in r \; \forall t_2 \in r \; ((t_1[X] = t_2[X]) \Rightarrow (t_1[Y] = t_2[Y]))$$
   by definition of functional dependency.
3. We know that for all valid instance r of R with Σ
   $$\forall t_1 \in r \; \forall t_2 \in r \; ((t_1[Y] = t_2[Y]) \Rightarrow (t_1[Z] = t_2[Z]))$$
   by definition of functional dependency.
4. Therefore, for all valid instance r of R with Σ
   $$\forall t_1 \in r \; \forall t_2 \in r \; ((t_1[X] = t_2[X]) \Rightarrow (t_1[Z] = t_2[Z]))$$
   by definition of functional dependency.
5. Therefore, $X \rightarrow Z \in \Sigma^+$.

Proof Sketch

Theorem #4

Proof Sketch #1

We will prove that for any set of attributes $S \in R$, then $S \rightarrow S^+$ can be derived by Armstrong axioms.

1. This is *recursively* true because every step of the attribute closure algorith is of the form $S \rightarrow S^i$ and $X \rightarrow Y$ with $X \subseteq S^i$.
2. Therefore $S^i \rightarrow X$ by Reflexivity.
3. Therefore $S^i \rightarrow (X \cup S^i)$ by Augmentation of (2) with S^i.
4. Therefore $S \rightarrow (X \cup S^i)$ by Transitivity of (1) and (3).
5. Therefore $(X \cup S^i) \rightarrow (Y \cup S^i)$ by Augmentation of $X \rightarrow Y$ with S^i.
6. Therefore $S \rightarrow S^{i+1}$ where $S^{i+1} = (Y \cup S^i)$ by transitivity of (4) and (5).

Proof Sketch

Theorem #4

Proof Sketch #2

We have shown that $S \rightarrow S^+$ can be derived. Now we need to show that if $(X \rightarrow Y) \in \Sigma^+$ then it can be derived from $X \rightarrow X^+$.

1. We know that $Y \subseteq X^+$ by property of attribute closure.
2. Therefore $X^+ \rightarrow Y$ by Reflexivity.
3. Therefore $X \rightarrow Y$ by transitivity of $X \rightarrow X^+$ and $X^+ \rightarrow Y$.

Proof Sketch

Theorem #5

Proof

This is practically from the definition of Σ^+ and equivalence.

Proof Sketch

Theorem #6

Proof

If Σ is minimal, then Σ is a minimal cover for Σ.

Proof Sketch

Equivalence #2

Proof

- Let $\Sigma'' = (\Sigma' \cup (X - \{B\}) \rightarrow \{A\})$ and $\Sigma''' = \Sigma'' - \{X \rightarrow \{A\}\}$.
- Since $(X - \{B\}) \rightarrow \{A\}$ is logically entailed by Σ', Σ' logically entails Σ''.
  As required by the step.
- Since Σ'' has all functional dependencies Σ' has, Σ'' logically entails Σ'.
  Since $\Sigma'' \supseteq \Sigma'$. ***Therefore $\Sigma' \equiv \Sigma''$.***
- Since Σ'' has all functional dependencies Σ''' has, Σ'' logically entails Σ'''.
  Since $\Sigma'' \supseteq \Sigma'''$.
- Show Σ''' can logically entail Σ'' by showing that Σ''' can logically entail $X \rightarrow \{A\}$.
  1) We know $(X - \{B\}) \rightarrow \{A\}$ is in Σ'''.
  2) By augmentation of (1) with $\{B\}$, we obtain $(X - \{B\}) \cup \{B\} \rightarrow \{A\} \cup \{B\} = X \rightarrow \{A, B\}$.
  3) By reflexivity, since $\{A\} \subseteq \{A, B\}$, we have $\{A, B\} \rightarrow \{A\}$.
  4) By transitivity of (2) and (3), we have $X \rightarrow \{A\}$.

Hence Σ' and Σ''' are equivalent by transitivity of equivalence.

Proof Sketch

Equivalence #3

Proof

- Let $\Sigma'' = (\Sigma' - \{X \rightarrow Y\})$.
- Since $X \rightarrow Y$ is logically entailed by Σ'', Σ'' logically entails $\Sigma'' \cup \{X \rightarrow Y\}$.
- But $\Sigma'' \cup \{X \rightarrow Y\} = \Sigma'$, so Σ'' logically entails Σ'.
- Since Σ' has all functional dependencies Σ'' has, Σ' logically entails Σ''.

  Since $\Sigma' \supseteq \Sigma''$.

Hence Σ' and Σ'' are equivalent.

Proof Sketch

Equivalence #4

Proof

Although not mentioned as a theorem, we can show that given $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow (Y \cup Z)$ holds.

1. We know that $X \rightarrow Y$. (*given in Σ*)
2. We know that $X \rightarrow Z$. (*given in Σ*)
3. Therefore $X \rightarrow (X \cup Y)$. (*Augmentation of (1) with X*)
4. Therefore $(X \cup Y) \rightarrow (Y \cup Z)$. (*Augmentation of (2) with Y*)
5. Therefore $X \rightarrow (Y \cup Z)$. (*Transitivity of (3) and (4)*) □

This together with addendum on the proof of Theorem #1 implies that we can merge and split the right hand side of any functional dependencies.

Proof Sketch

Step 2

Steps

Consider {A,B} → {C}.

1. Check if we can remove **B** from {A,B} → {C}
   A. Compute **{A}⁺** with respect to Σ_2. $\{A\}^+$ = **{A}**
   B. Since **{C}** ⊈ **{A}⁺**, we cannot remove **B** from {A,B} → {C}

2. Check if we can remove **A** from {A,B} → {C}
   A. Compute **{B}⁺** with respect to Σ_2. $\{B\}^+$ = **{B, C, D, E}**
   B. Since **{C}** ⊈ **{B}⁺**, we can remove **A** from {A,B} → {C}
   C. Replace {A,B} → {C} with **{B} → {C}**

Current Σ

Σ_2 = {
~~{A,B} → {C}~~ ⇒ {B} → {C},
{A,B} → {D},
{A,B} → {E},
{A,C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

Consider {A,B} → {D}.

1. Check if we can remove **B** from {A,B} → {D}
   A. Compute $\{A\}^+$ with respect to Σ_2. $\{A\}^+$ = **{A}**
   B. Since **{D}** $\not\subseteq$ $\{A\}^+$, we cannot remove **B** from {A,B} → {D}

2. Check if we can remove **A** from {A,B} → {D}
   A. Compute $\{B\}^+$ with respect to Σ_2. $\{B\}^+$ = **{B, C, D, E}**
   B. Since **{D}** $\not\subseteq$ $\{B\}^+$, we can remove **A** from {A,B} → {D}
   C. Replace {A,B} → {D} with **{B} → {D}**

Current Σ

Σ_2 = {
{B} → {C},
~~{A,B} → {D}~~ ⇒ {B} → {D},
{A,B} → {E},
{A,C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

Consider {A,B} → {E}.

1. Check if we can remove **B** from {A,B} → {E}
   A. Compute **{A}⁺** with respect to Σ_2. $\{A\}^+$ = **{A}**
   B. Since **{E}** ⊈ **{A}⁺**, we cannot remove **B** from {A,B} → {E}

2. Check if we can remove **A** from {A,B} → {E}
   A. Compute **{B}⁺** with respect to Σ_2. $\{B\}^+$ = **{B, C, D, E}**
   B. Since **{E}** ⊈ **{B}⁺**, we can remove **A** from {A,B} → {E}
   C. Replace {A,B} → {E} with **{B} → {E}**

Current Σ

Σ_2 = {
{B} → {C},
{B} → {D},
~~{A,B} → {E}~~ ⇒ {B} → {E},
{A,C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

Consider {A,C} → {B}.

1. Check if we can remove **C** from {A,C} → {B}
   A. Compute **{A}⁺** with respect to **Σ_2**. $\{A\}^+$ = **{A}**
   B. Since **{B} ⊈ {A}⁺**, we cannot remove **C** from {A,C} → {B}

2. Check if we can remove **A** from {A,C} → {B}
   A. Compute **{C}⁺** with respect to **Σ_2**. $\{C\}^+$ = **{B, C, D, E}**
   B. Since **{B} ⊈ {C}⁺**, we can remove **A** from {A,C} → {B}
   C. Replace {A,C} → {B} with **{C} → {B}**

Current Σ

Σ_2 = {
{B} → {C},
{B} → {D},
{B} → {E},
~~{A,C} → {B}~~ ⇒ {C} → {B},
{A,C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

Consider {A,C} → {D}.

1. Check if we can remove **C** from {A,C} → {D}
   A. Compute $\{A\}^+$ with respect to Σ_2. $\{A\}^+$ = **{A}**
   B. Since **{D}** $\not\subseteq$ $\{A\}^+$, we cannot remove **C** from {A,C} → {D}
2. Check if we can remove **A** from {A,C} → {D}
   A. Compute $\{C\}^+$ with respect to Σ_2. $\{C\}^+$ = **{B, C, D, E}**
   B. Since **{D}** $\not\subseteq$ $\{C\}^+$, we can remove **A** from {A,C} → {D}
   C. Replace {A,C} → {D} with **{C} → {D}**

Current Σ

Σ_2 = {
{B} → {C},
{B} → {D},
{B} → {E},
{C} → {B},
~~{A,C} → {D}~~ ⇒ {C} → {D},
{A,C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

Consider {A,C} → {E}.

1. Check if we can remove **C** from {A,C} → {E}
   A. Compute $\{A\}^+$ with respect to Σ_2. $\{A\}^+$ = **{A}**
   B. Since **{E}** $\not\subseteq$ **{A}$^+$**, we cannot remove **C** from {A,C} → {E}

2. Check if we can remove **A** from {A,C} → {E}
   A. Compute $\{C\}^+$ with respect to Σ_2. $\{C\}^+$ = **{B, C, D, E}**
   B. Since **{E}** $\not\subseteq$ **{C}$^+$**, we can remove **A** from {A,C} → {E}
   C. Replace {A,C} → {E} with **{C} → {E}**

Current Σ

Σ_2 = {
{B} → {C},
{B} → {D},
{B} → {E},
{C} → {B},
{C} → {D},
~~{A,C} → {E}~~ ⇒ {C} → {E},
{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}
}

Proof Sketch

Step 2

Steps

At this point, it is safe to stop because the rest of the functional dependencies only have a single attribute on the left-hand side.

While it is possible in general to have an empty set as the left-hand side *(e.g., {} → {A})*, our initial Σ must already have at least one functional dependency with an empty set as the left-hand side.

Since there is no such functional dependencies, it will not be suddenly introduced.

Current Σ

Σ_2 = {

{B} → {C},
{B} → {D},
{B} → {E},
{C} → {B},
{C} → {D},
{C} → {E},

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

~~{B} → {D},~~

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Steps

Consider {B} → {D}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing **{B} → {D}** from Σ_2.
2. Compute $\{B\}^+$ with respect to Σ'_2

   $\{B\}^+$ = **{B, C, D, E}**
3. Since **{D} ⊆ {B}⁺**, we can remove {B} → {D}

Current Σ

Σ_2 = {

~~{B} → {D},~~

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Steps

Consider {B} → {C}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing **{B} → {C}** from Σ_2.
2. Compute $\{B\}^+$ with respect to Σ'_2

   $\{B\}^+$ = **{B, E}**
3. Since **{C} ⊈ {B}$^+$**, we cannot remove {B} → {C}

Current Σ

Σ_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Steps

Consider {C} → {B}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing {C} → {B} from Σ_2.
2. Compute $\{C\}^+$ with respect to Σ'_2

   $\{C\}^+$ = {C, D, E}
3. Since $\{B\} \notin \{C\}^+$, we cannot remove {C} → {B}

Current Σ

Σ_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Steps

Consider {C} → {D}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing {C} → {D} from Σ_2.
2. Compute $\{C\}^+$ with respect to Σ'_2

   $\{C\}^+$ = {B, C, E}
3. Since $\{D\} \notin \{C\}^+$, we cannot remove {C} → {D}

Current Σ

Σ_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
{B} → {E},
{C} → {E}

}

Steps

Consider {B} → {E}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing **{B} → {E}** from Σ_2.
2. Compute $\{B\}^+$ with respect to Σ'_2

   $\{B\}^+$ = **{B, C, D, E}**
3. Since **{E} ⊆ {B}⁺**, we can remove {B} → {E}

Current Σ

Σ_2 = {

{B} → {C},
{C} → {B},
{C} → {D},
~~{B} → {E},~~
{C} → {E}

}

Proof Sketch

Step 3

Working Σ

Σ'_2 = {

{B} → {C},
{C} → {B},
{C} → {D},

{C} → {E}

}

Steps

Consider {C} → {E}, check if we can remove this functional dependency.

1. Compute Σ'_2 *(see left)* by removing **{C} → {E}** from Σ_2.
2. Compute $\{C\}^+$ with respect to Σ'_2

   $\{C\}^+$ = **{B, C, D}**
3. Since **{E}** $\notin$ $\{C\}^+$, we can remove {C} → {E}

Current Σ

Σ_2 = {

{B} → {C},
{C} → {B},
{C} → {D},

{C} → {E}

}

```
postgres=# \q
Press any key to continue . . .
```