香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

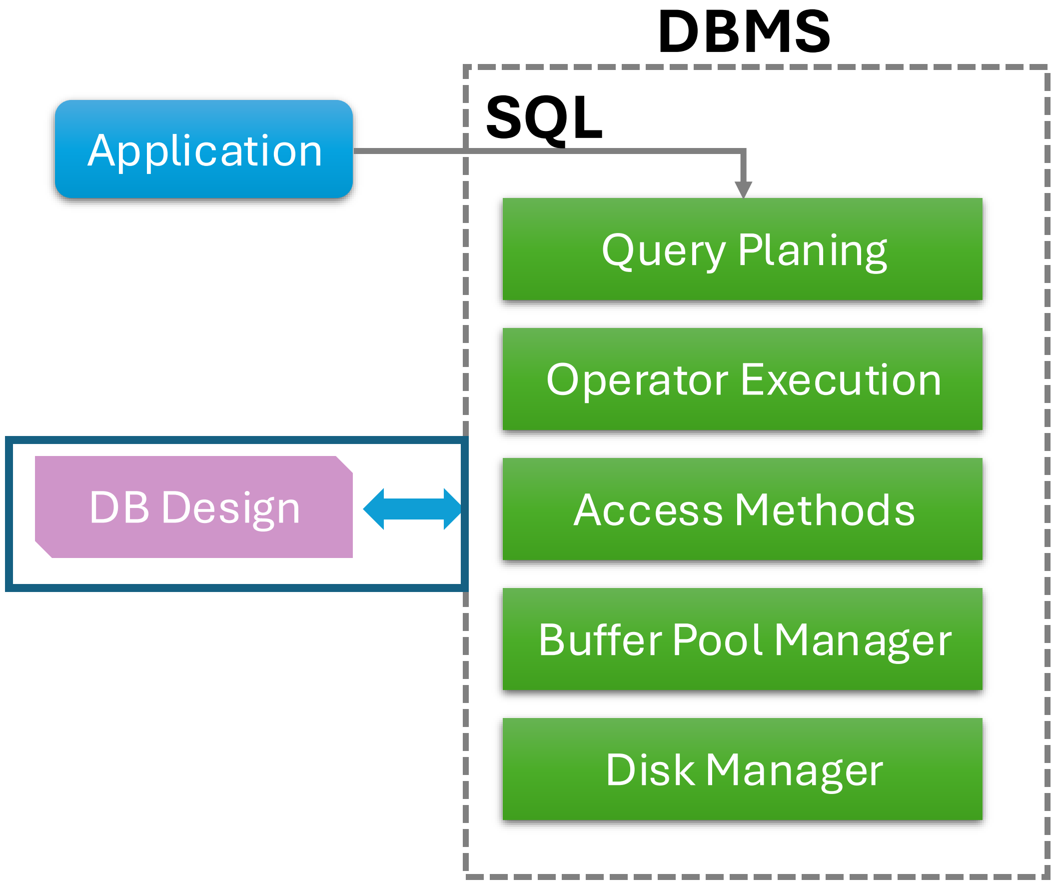
Ack: Dr. Chui Chun Kit @ HKU

CSC3170
15: DB Design *part b*

Chenhao Ma
School of Data Science
The Chinese University of Hong Kong, Shenzhen

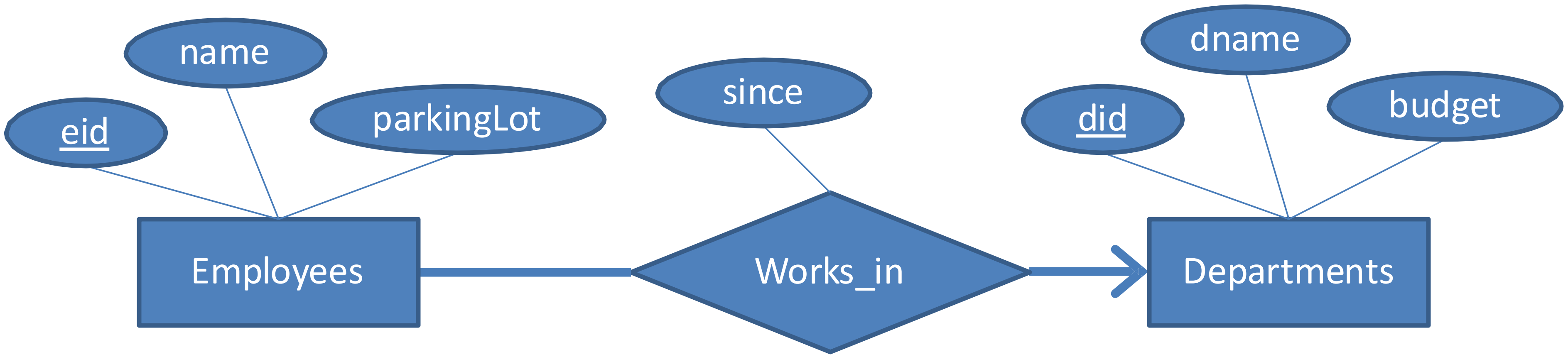
This Lecture

- Normalization



Motivating Example

- Let’s consider the following specifications
  - **Employees** have ***eid*** (key), ***name***, ***parkingLot***.
  - **Departments** have ***did*** (key), ***dname***, ***budget***.
  - An employee works in exactly one department, **since** some date.
  - Employees who work in the same department must park at the same ***parkingLot.***



SCHOOL OF DATA SCIENCE

Motivating Example

- **Reduce to relational tables**
  - **Employees(*<u>eid</u>*, *name*, *parkingLot*, *did*, *since*)**

    Foreign key: ***did*** references Departments(did)
  - **Departments(*<u>did</u>*, *dname*, *budget*)**

Observation: In **Employees** table, whenever ***did*** is **1**, **parkingLot** must be “**A**”!
Implication: The constraint “*Employees who work in the same department must park at the same **parkingLot***” is **NOT** utilized in the design!!!
There are some **redundancy** in the Employees table.

eid	name	parkingLot	did	since
1	Kit	A	1	1/9/2014
2	Ben	B	2	2/4/2010
3	Ernest	B	2	30/5/2011
4	Betty	A	1	22/3/2013
5	David	A	1	4/11/2004
6	Joe	B	2	12/3/2008
7	Mary	B	2	14/7/2009
8	Wandy	A	1	9/8/2008

did	dname	budget
1	Human Resource	4M
2	Accounting	3.5M

Yes! As *parkingLot* is “**functionally depend**” on ***did***, we should not put ***parkingLot*** in the **Employee** table.

So We Need…

- Database normalization
  - The process of organizing the columns and tables of a relational database to minimize redundancy and dependency.
- **To make sure that every relation R is in a "good" form.**
  - If R is not "good", decompose it into a set of relations $\{R1, R2, \ldots, Rn\}$.

Normalization Goals

- **We would like to meet the following goals when we decompose a relation schema R with a set of functional dependencies F into $R_1, R_2, \ldots, R_n$**
  - **Lossless-join** – Avoid the decomposition result in information loss.
  - **Reduce redundancy** – The decomposed relations R_i should not be redundant.
    - i.e., in **Boyce-Codd Normal Form** (**BCNF**).
  - **Dependency preserving** – Avoid the need to join the decomposed relations to check the functional dependencies when new tuples are inserted into the database.

Lossless-join Decomposition

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

$F = \{B \rightarrow C\}$

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

$\neq$

$R_1 \bowtie R_2 = \pi_{A,\ B}(R) \bowtie \pi_{A,\ C}(R)$

A	B	C
1	1	3
1	1	2
1	2	3
1	2	2
2	1	3
3	2	2
3	2	3
3	1	2
3	1	3
4	2	2
4	2	3
4	1	2
4	1	3

To check if the decomposition will cause information lost, let's try to join $\mathbf{R_1}$ and $\mathbf{R_2}$ and see if we can recover **R**.
As we see that $\mathbf{R_1} \bowtie \mathbf{R_2} \neq \mathbf{R}$, the decomposition has information lost.
This is <u>NOT a lossless-join decomposition</u>.

Think in this way:
Is this decomposition "**lossless join decomposition**"?
I.e., Is there any information lost if we decompose **R** in this way?

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{B,\ C}(R)$

B	C
1	3
2	2

=

$R_1 \bowtie R_2 = \pi_{A,\ B}(R) \bowtie \pi_{B,\ C}(R)$

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Well done! Since $\mathbf{R_1 \bowtie R_2 = R}$, breaking down **R** to $\mathbf{R_1}$ and $\mathbf{R_2}$ in this way has no information lost.
This decomposition is lossless-join decomposition.

How about decomposing the relation **R(A,B,C)** into $\mathbf{R_1(A,B)}$ and $\mathbf{R_2(B,C)}$?

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

$F = \{B \rightarrow C\}$

What is/are the condition(s) for a decomposition to be **lossless-join**?

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{B,\ C}(R)$

B	C
1	3
2	2

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	B
1	1

Let's consider the first tuple (**1**,**1**,**3**) in **R**.

Note that there is only **ONE** tuple in $\mathbf{R_1}$ with **A=1, B=1**.

NOT Lossless-join decomposition

$R_1 = \pi_{A,\,B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\,C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B →C}

1

A	B
1	1

Let's consider the first tuple (**1**,**1**,**3**) in **R**.

Note that there is only **ONE** tuple in **R_1** with **A=1, B=1**.

2

A	C
1	3
1	2

Since **A →AC** is **NOT** a functional dependency in F^+, there can be **more than one tuples** with **A=1** in **R_2** (e.g., (**1**,**3**), (**1**,**2**)) .

NOT Lossless-join decomposition

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

SCHOOL OF DATA SCIENCE

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	B
1	1

Let's consider the first tuple (**1**,**1**,**3**) in **R**.

Note that there is only **ONE** tuple in $\mathbf{R_1}$ with **A=1, B=1**.

⋈

2

A	C
1	3
1	2

Since **A → AC** is **NOT** a functional dependency in F^+, there can be **more than one tuples** with **A=1** in $\mathbf{R_2}$ (e.g., (**1**,**3**), (**1**,**2**)) .

=

3

A	B	C
1	1	3
1	1	2

✔

X

Therefore when we join $\mathbf{R_1}$ and $\mathbf{R_2}$, **more than one tuples will be generated** (i.e., (**1**,**1**) in $\mathbf{R_1}$ combine with (**1**,**3**) and (**1**,**2**) in $\mathbf{R_2}$)

NOT Lossless-join decomposition

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

Observation:

The decomposition of R(A,B,C) into R_1(**A**,B) and R_2(**A**,C) is NOT lossless-join because

- **A → AC**

is **NOT** in F^+ , **and … (to be explained in the next slide)**

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	C
1	3

Let's consider the first tuple (**1,1,3**) in **R**.

Note that there is only **ONE** tuple in $\mathbf{R_2}$ with **A=1, C=3**.

NOT Lossless-join decomposition

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

SCHOOL OF DATA SCIENCE

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	C
1	3

Let's consider the first tuple (**1,1,3**) in **R**.

Note that there is only **ONE** tuple in $\mathbf{R_2}$ with **A=1, C=3**.

2

A	B
1	1
1	2

Since **A → AB** is **NOT** a functional dependency in F^+, there can be **more than one tuples** with **A=1** in $\mathbf{R_1}$ (i.e., (**1,1**), (**1,2**)) .

NOT Lossless-join decomposition

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\ C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	C
1	3

Let's consider the first tuple (**1,1,3**) in **R**.

Note that there is only **ONE** tuple in $\mathbf{R_2}$ with **A=1, C=3.**

⋈

2

A	B
1	1
1	2

Since **A → AB** is **NOT** a functional dependency in F^+, there can be **more than one tuples with A=1** in $\mathbf{R_1}$ (i.e., (**1,1**), (**1,2**)) .

=

3

A	B	C
1	1	3
1	2	3

✔ X

Therefore when we join $\mathbf{R_1}$ and $\mathbf{R_2}$, **more than one tuples will be generated** (i.e., (**1,3**) in $\mathbf{R_2}$ combine with (**1,1**) and (**1,2**) in $\mathbf{R_1}$)

NOT Lossless-join decomposition

$R_1 = \pi_{A,\,B}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

$R_2 = \pi_{A,\,C}(R)$

A	C
1	3
1	2
2	3
3	2
3	3
4	2
4	3

Observation:

The decomposition of R(A,B,C) into R_1(**A**,B) and R_2(**A**,C) is NOT lossless-join because

- A→ AC (explained in previous slide), **and**
- **A→ AB**

are **NOT** in F^+ .

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	B
1	1

Let's consider the first tuple (**1,1,3**) in **R**. Note that there is only **ONE** tuple in $\mathbf{R_1}$ with **A=1, B=1**.

Lossless-join decomposition

$R_1 = \pi_{A,\,B}(R)$ $R_2 = \pi_{B,\,C}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

B	C
1	3
2	2

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

$F = \{B \rightarrow C\}$

1

A	B
1	1

Let's consider the first tuple (**1,1,3**) in **R**. Note that there is only **ONE** tuple in **R_1** with **A=1, B=1**.

2

B	C
1	3

Since **B $\rightarrow$BC** is a functional dependency in F^+, there is only **one tuple** with **B=1** in **R_2**.

Lossless-join decomposition

$R_1 = \pi_{A,\,B}(R)$ $R_2 = \pi_{B,\,C}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

B	C
1	3
2	2

Lossless-join Decomposition

R

A	B	C
1	1	3
1	2	2
2	1	3
3	2	2
3	1	3
4	2	2
4	1	3

Functional dependencies

F = {B → C}

1

A	B
1	1

Let's consider the first tuple (**1,1,3**) in **R**. Note that there is only **ONE** tuple in $\mathbf{R_1}$ with **A=1, B=1**.

⋈

2

B	C
1	3

Since **B → BC** is a functional dependency in F^+, there is only **one tuple** with **B=1** in $\mathbf{R_2}$.

=

3

A	B	C
1	1	3

✔

Therefore when we join $\mathbf{R_1}$ and $\mathbf{R_2}$, there will be **ONLY ONE tuple generated**, and that must be the corresponding tuple (**1,1,3**) in **R**.

Lossless-join decomposition

$R_1 = \pi_{A,\ B}(R)$ $R_2 = \pi_{B,\ C}(R)$

A	B
1	1
1	2
2	1
3	2
3	1
4	2
4	1

B	C
1	3
2	2

Observation:

The decomposition of R(A,B,C) into R_1(A,**B**) and R_2(**B**,C) is lossless-join because

- **B→ BC**

is in F^+ .

Testing for Lossless-join Decomposition

- **Consider a decomposition of R into R_1 and R_2.**
  - Schema of R = schema of R_1 ∩ schema of R_2.
- **Let schema of R_1 ∩ schema of R_2 be R_1 and R_2's common attributes.**
  - A decomposition of R into R_1 and R_2 is lossless-join if and only if at least one of the following dependencies holds.

Schema of $R_1 \cap$ schema of $R_2 \rightarrow$ schema of R_1

OR

Schema of $R_1 \cap$ schema of $R_2 \rightarrow$ schema of R_2

Dependency preserving Decomposition

Dependency preserving

- **When decomposing a relation, we also want to keep the functional dependencies.**
  - A FD X → Y is preserved in a relation R if R contains all the attributes of X and Y.
- **If a dependency is lost when R is decomposed into R_1 and R_2:**
  - When we insert a new record in R_1 and R_2, we have to obtain $R_1 \bowtie R_2$ and check if the new record violates the lost dependency before insertion.
  - It could be very inefficient because joining is required in every insertion!

Dependency preserving

Note that $A \rightarrow CD$ is in F^+ because of the **Transitivity axiom.**

- **Consider R(A,B,C,D), F = {$A \rightarrow B$, $B \rightarrow CD$}**
  - $F^+ = \{A \rightarrow B, B \rightarrow CD, A \rightarrow CD$, trivial FDs$\}$
- **If R is decomposed to $R_1(A,B)$, $R_2(B,C,D)$:**
  - $F_1 = \{A \rightarrow B$, trivials$\}$, the projection of F^+ on R_1
  - $F_2 = \{B \rightarrow CD$, trivials$\}$, the projection of F^+ on R_2

R

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

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
2	1
3	2
4	1

$R_2 = \pi_{B,\ C,\ D}(R)$

B	C	D
1	3	4
2	2	3

This is a **dependency preserving decomposition** as:

$$(F_1 \cup F_2)^+ = F^+$$

Let us illustrate the implication of dependency preserving in the next slide.

Dependency preserving

- **Consider R(A,B,C,D), F = {A → B, B → CD}**
  - F^+ = {A → B, B → CD, A → CD, trivial FDs}
- **Is this a lossless join decomposition?**
  - Yes! As B → R_2 (i.e., B → BCD) holds in F^+. That mean we can recover R by $R_1 \bowtie R_2$.
- **Why it is dependency preserving?**

Think about it…

If we insert a new record

A	B	C	D
5	1	4	4

into R_1 and R_2:

- R_1

A	B
5	1

- R_2

B	C	D
1	4	4

We need to check if the new record will make the database violate any FDs in F^+.

Is such decomposition allow us to do the validation on R_1 and R_2 **ONLY?** (But no need to join R_1 and R_2 to validate it?)

R

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

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
2	1
3	2
4	1

$R_2 = \pi_{B,\ C,\ D}(R)$

B	C	D
1	3	4
2	2	3

SCHOOL OF DATA SCIENCE

Dependency preserving

- $F^+ = \{ A \rightarrow B, B \rightarrow CD, A \rightarrow CD$, trivials}
  - Inserting tuple (5,1,4,4) violates $B \rightarrow CD$.
- **The decomposition is dependency preserving as we only need to check:**
  - Inserting

A	B
5	1

    violate any F_1 in R_1?

    This involves checking $F_1=\{A \rightarrow B\}$.
  - Inserting

B	C	D
1	4	4

    violate any F_2 in R_2?

    This involves checking $F_2=\{B \rightarrow CD\}$.

We can check F_1 on R_1 and F_2 on R_2 only because $(F_1 \cup F_2)^+ = F^+$

A	B	C	D
1	1	3	4
2	1	3	4
3	2	2	3
4	1	3	4
5	1	4	4

Decompose

$R_1 = \pi_{A, B}(R)$ $R_2=\pi_{B, C, D}(R)$

A	B
1	1
2	1
3	2
4	1
5	1

B	C	D
1	3	4
2	2	3
1	4	4

Although among the two validations we haven't checked **A→CD**, but since $A \rightarrow B$ is checked in F_1, and $B \rightarrow CD$ is checked in F_2, if we pass both F_1 and F_2, it implies $A \rightarrow CD$.

Dependency preserving

- **What about decompose R to $R_1(A,B)$, $R_2(A,C,D)$?**
- **R is decomposed to $R_1(A,B)$, $R_2(A,C,D)$**
  - $F^+ = \{A \rightarrow B, B \rightarrow CD, A \rightarrow CD$, trivial FDs$\}$
  - $F_1 = \{A \rightarrow B$, trivials$\}$, the projection of F^+ on R_1
  - $F_2 = \{A \rightarrow CD$, trivials$\}$, the projection of F^+ on R_2

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

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
2	1
3	2
4	1

$R_2 = \pi_{A,\ C,\ D}(R)$

A	C	D
1	3	4
2	3	4
3	2	3
4	3	4

This is **NOT a dependency preserving decomposition** as:

$$(F_1 \cup F_2)^+ \neq F^+$$

Let us illustrate the implication of NOT dependency preserving in the next slide.

Dependency preserving

- **What about decompose R to R_1(A,B), R_2(A,C,D) ?**
- **Is this a lossless join decomposition?**
  - Yes! As $A \rightarrow R_1$ (i.e., $A \rightarrow AB$) holds in F^+. That mean we can recover R by $R_1 \bowtie R_2$.
- **Is it dependency preserving?**

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

Decompose

$R_1 = \pi_{A,\ B}(R)$

A	B
1	1
2	1
3	2
4	1

$R_2 = \pi_{A,\ C,\ D}(R)$

A	C	D
1	3	4
2	3	4
3	2	3
4	3	4

Think about it…

If we insert a new record

A	B	C	D
5	1	4	4

into R_1 and R_2:

- R_1

A	B
5	1

- R_2

A	C	D
5	4	4

We need to check if the new record will make the database violate any FDs in F^+. Is such decomposition allow us to do the validation on R_1 and R_2 **only** (but no need to join R_1 and R_2)?

SCHOOL OF DATA SCIENCE

Dependency preserving

- $F^+ = \{ A \rightarrow B, B \rightarrow CD, A \rightarrow CD \}$
  - Inserting tuple (5,1,4,4) **violates** $B \rightarrow CD$.
- **The decomposition is NOT dependency preserving as if we only check:**
  - Inserting

A	B
5	1

violate any F_1 in R_1?

This involves checking $F_1=\{A \rightarrow B\}$.

  - Inserting

A	C	D
5	4	4

violate any F_2 in R_2?

This involves checking $F_2=\{A \rightarrow CD\}$.

We CANNOT check F_1 on R_1 and F_2 on R_2 only because
$(F_1 \cup F_2)^+ \neq F^+$
Decomposition in this way requires joining tables to validate $B \rightarrow CD$ for **EVERY INSERTION**!

A	B	C	D
1	1	3	4
2	1	3	4
3	2	2	3
4	1	3	4
5	1	4	4

Decompose

$R_1 = \pi_{A,\,B}(R)$

A	B
1	1
2	1
3	2
4	1
5	1

$R_2=\pi_{A,\,C,\,D}(R)$

A	C	D
1	3	4
2	3	4
3	2	3
4	3	4
5	4	4

Although we passed F_1 and F_2, it doesn't mean that we passed all FDs in F!
It is because **we lost the FD $B \rightarrow CD$ in the decomposition**.

SCHOOL OF DATA SCIENCE

Dependency preserving

What is the condition(s) for a decomposition to be **dependency preserving**?

- **Let F be a set of functional dependencies on R.**
  - $R_1, R_2, \ldots, R_n$ be a decomposition of R.
  - F_i be the set of FDs in F^+ that include only attributes in R_i.
- **A decomposition is dependency preserving if and only if**

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

  - Where F_i is the set of FDs in F^+ that include only attributes in R_i.

Exercise

- **Given R(A, B, C) , F = {A → B , B → C}**
  - Is **R_1(A, B), R_2(B, C)** a dependency preserving decomposition?
- **First we need to find F^+ , F_1 and F_2.**
  - $F^+ = \{A \rightarrow B , B \rightarrow C, A \rightarrow C,$ some trivial FDs$\}$
  - $F_1 = \{A \rightarrow B$ and trivial FDs $\}$
  - $F_2 = \{B \rightarrow C$ and trivial FDs $\}$

Note that A→C is in F^+ because of the **Transitivity axiom.**

- **Then we check if $(F_1 \cup F_2)^+ = F^+$ is true.**
  - Since $F_1 \cup F_2 = F$,this implies $(F_1 \cup F_2)^+ = F^+$.
- **This decomposition is dependency preserving.**

Boyce-Codd Normal Form

SCHOOL OF DATA SCIENCE

FD and redundancy

- **Consider the following relation:**
  - **Customer(** ***<u>id</u>*, *name*, *dptID***)
  - F = { {***id***} → {***name***, ***dptID***} }

Customer

id	name	dptID
1	Kit	1
2	David	1
3	Betty	2
4	Helen	2

- **{*id*} is a key in Customer.**
  - Because the attribute closure of {***id***} (i.e., $\{id\}^+$ = {***id, name, dptID***}), which covers all attributes of **Customer**.

Observation: All non-trivial FDs in F form a key in the relation Customer.

- This implies that there are no other FD that is just involve a subset of columns in the relation.
- This implies that **Customer** has **no redundancy**.

FD and redundancy

- **As another example:**
  - **Customer(** ***<u>id</u>*, *name*, *dptID*, *building*****)**
  - F = { {***id***} → {***name***, ***dptID*** , ***building***}
    {***dptID***} → {***building***} }

Customer

id	name	dptID	building
1	Kit	1	CYC
2	David	1	CYC
3	Betty	2	HW
4	Helen	2	HW

- **{*dptID*} → {*building*} brings redundancy. Why?**
  - Tuples have the same ***dptID*** must have the same ***building*** (e.g., ***dptID***=**1**, ***building***=“**CYC**”).
  - But those tuples can have different values in ***id*** and ***name***. For each different ***id*** values with the same ***dptID***, ***building*** will be repeated (**redundancy**).

For example, for tuples with (***id***=**1**, ***dptID***=**1**) and (***id***=**2**, ***dptID***=1) , ***building*** must equal “**CYC**” (redundancy).

FD and redundancy

- **As another example:**
  - **Customer(** ***<u>id</u>*, *name*, *dptID*, *building*** **)**
  - F = { {***id***} → {***name***, ***dptID*** , ***building***}
    {***dptID***} → {***building***} }

Customer

id	name	dptID	building
1	Kit	1	CYC
2	David	1	CYC
3	Betty	2	HW
4	Helen	2	HW

- **How to check?**
  - Check if the attribute set closure of {***dptID***} covers all attributes in **Customer**. ({***dptID***}$^+$ = {***dptID, building***} ≠ **Customer**)

Redundancy is related to FDs. If there is an FD $\alpha \rightarrow \beta$, where $\{\alpha\}^+$ does not cover all attributes in R, then we will have redundancy in R!

Boyce-Codd Normal Form

- **Summarizing the observations, a relation R has no redundancy, or in Boyce-Codd Normal Form (BCNF), if the following is satisfied:**
  - For all FDs 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., $\beta \subseteq \alpha$)

We won't border with trivial FDs such as A→A, AB→A …etc

α is a key (superkey) for R

i.e., The attribute set closure of α, represented as $\{\alpha\}^+$, covers all attributes in **R**.

In another word, in BCNF, every non-trivial FD forms a key.

SCHOOL OF DATA SCIENCE

How to test for BCNF?

- **Formally, for verifying if R is in BCNF**
  - For each non-trivial dependency $\alpha \rightarrow \beta$ in **F^+ (the functional dependency closure)**, check if α^+ covers the whole relation (i.e., whether α is a superkey).
  - If any α^+ does not cover the whole relation, **R** is not in BCNF.
- **Simplified test:**
  - It is suffices to check only the dependencies in the given **F** for violation of BCNF, rather than check all dependencies in **F^+**

For example, given R(A,B,C); F = {A→B, B→C},
we only need to check if both $\{A\}^+$ and $\{B\}^+$ cover {A,B,C}.
We do not need to derive F^+ = {A→B, B→C, A→C, ...etc} and check each FD because A→C already considered when computing $\{A\}^+$.

How to test for BCNF?

- **However, if we decompose R into R_1 and R_2, we cannot use only F to check if the "decomposed" relations (i.e., R_1 and R_2) is BCNF, we have to use F^+ instead.**
- **Illustration**
  - R(A, B, C, D), F = {A → B, B → C}

To test if **R** is in BCNF, it is suffices to check only the dependencies in **F** (but not **F^+**)

R

A	B	C	D
1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	1	5

An example R that satisfies F

  - **$\{A\}^+$ covers all {A,B,C,D}?**
    Since $\{A\}^+$ = {A,B,C} ≠ {A,B,C,**D**}, **R** is not in BCNF.

As illustrated through this instance, since $\{A\}^+$ = {A,B,C} ≠ {A,B,C,**D**}, this implies that it will cause redundancy when we have tuples with the same value across {ABC} but different values in **D**.

How to test for BCNF?

To illustrate why we cannot use only **F** to test decomposed relations for BCNF, let's try to **decompose R into R_1(A, B) and R_2(A, C, D)**

- **Illustration**
  - R(A, B, C, D), F = {A → B, B → C}
- **Is R_2(A, C, D) in BCNF?**

When we check **R_2**, none of FDs in **F** is contained in **R_2**. Does this mean no non-trivial FDs are in **R_2**, and **R_2** is in BCNF?

R

A	B	C	D
1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	1	5

R_1(A, B)

A	B
1	1

R_2(A, C, D)

A	C	D
1	1	1
1	1	2
1	1	3
1	1	4
1	1	5

No! We need to use **F^+** to verify if **R_2** is BCNF

How to test for BCNF?

- **In R_2(A, C, D), $A \rightarrow C$ is in F^+, because:**
  - **$A \rightarrow C$** can be obtained by **transitivity rule on** **$A \rightarrow B$ and $B \rightarrow C$**
  - There is a non trivial FD **$A \rightarrow C$** in **R_2** that we have missed!
- **Therefore in R_2 we check $\{A\}^+ = \{A,C\} \neq \{A,C,D\}$**
  - Thus, **A** is not a key in R_2
  - **R_2** is NOT in BCNF.

R

A	B	C	D
1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	1	5

Conclusion: *When we test whether a **decomposed relation** is in BCNF, we must project **F+** onto the relation (e.g., **R_2**), not **F**!*

R_1(A, B)

A	B
1	1

R_2(A, C, D)

A	C	D
1	1	1
1	1	2
1	1	3
1	1	4
1	1	5

Normlization

Normalization goal

- **When we decompose a relation R with a set of functional dependencies F into R_1, R_2, …, R_n, we try to meet the following goals:**

1. Lossless-join – Avoid the decomposition result in information loss.

2. No Redundancy – The decomposed relations R_i should be in Boyce-Codd Normal Form (BCNF). (There are also other normal forms.)

3. Dependency preserving – Avoid the need to join the decomposed relations to check the functional dependencies.

Illustration

- **Consider R(A, B, C), F = {A→B , B→C}, is R in BCNF? If not, decompose R into relations that are in BCNF.**

R

A	B	C
1	1	2
2	1	2
3	1	2
4	1	2

- **Is R in BCNF?**
  - Because $\{B\}^+=\{B,C\} \neq \{A,B,C\}$
  - Since $\{B\}^+$ does not cover all attributes in R, R is **NOT** in BCNF.

Think in this way: How should we decompose **R** such that the decomposed relations are always lossless join?
Note: A decomposition is lossless join if **at least one of the following dependencies is in F^+**

Schema of $R_1 \cap$ schema of $R_2 \rightarrow$ schema of R_1

OR

Schema of $R_1 \cap$ schema of $R_2 \rightarrow$ schema of R_2

Illustration

Idea: To make the decomposition always lossless join, we can pick the FD **A→B** and make the decomposed relation as:

- R_1(**A,B**) – the attributes in the L.H.S. and R.H.S. of the FD.
- R_2(**A,C**) – the attribute(s) in the L.H.S. of the FD, and the remaining attributes that does not appear in R_1.

- If we decompose the relation R in this way the following must be true:

Schema of $R_1 \cap$ schema of $R_2 \rightarrow$ schema of R_1

- Schema of $R_1 \cap$ schema of R_2 is **A**.
- **A→R_1= A→AB must be true** because **R_1** must consists of the L.H.S. and R.H.S. of the FD **A→B** in **F**.

Illustration

$F = \{A \rightarrow B, B \rightarrow C\}$ $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, \text{trivial FDs}\}$

	$R_1(A, B)$	$R_2(A, C)$
F_x	$A \rightarrow B$	$A \rightarrow C$

- **Is $R_1(A, B)$ in BCNF?**
  - $F_1 = \{A \rightarrow B, \text{trivial FDs}\}$, it is a projection of F^+ on R_1.
  - Since $\{A\}^+ = \{A,B\} = R_1$, $\{A\}$ is a key in R_1.
  - Since all FDs in F_1 forms a key, R_1 is in BCNF.

R

A	B	C
1	1	2
2	1	2
3	1	2
4	1	2

- **Is $R_2(A, C)$ in BCNF?**
  - $F_2 = \{A \rightarrow C, \text{trivial FDs}\}$, it is a projection of F^+ on R_2.
  - Since $\{A\}^+ = \{A,C\} = R_2$, $\{A\}$ is a key in R_2.
  - Since all FDs in F_2 forms a key, R_2 is in BCNF.

R_1

A	B
1	1
2	1
3	1
4	1

R_2

A	C
1	2
2	2
3	2
4	2

Therefore, decomposing $R(A, B, C)$ with $F = \{A \rightarrow B, B \rightarrow C\}$ to $R_1(A, B)$ and $R_2(A, C)$ result in a lossless join decomposition (**no information lost**), and BCNF relations (**no redundancy**)

Illustration

- **Is the decomposition dependency preserving ?**
  - $F = \{A \rightarrow B, B \rightarrow C\}$
  - $(F_1 \cup F_2) = (A \rightarrow B, A \rightarrow C)$
- Since $B \rightarrow C$ disappears in R_1 and R_2, $(F_1 \cup F_2)^+ \neq F^+$.
- The decomposition is **NOT dependency preserving**.

Note: Although the decomposition is not dependency preserving, but it is lossless join, so we can join $\mathbf{R_1}$ and $\mathbf{R_2}$ to test $B \rightarrow C$.

BCNF decomposition algorithm

```
result = {R};
done = false;
compute F+;
while (done == false) {
    if (there is a schema Ri in result and Ri is not in BCNF)
        let α → β be a non-trivial FD that holds on Ri s.t. {α}+ ≠ Ri
        result = (result – Ri) ∪ (α β) ∪ (Ri – β)
    else
        done = true;
}
```

α is not a key;
$\alpha \rightarrow \beta$ causes R_i to violate BCNF

1. Delete R_i

2. Create a relation with only α and β

3. Create a relation containing R_i but with β removed.

Each R_i is in BCNF, and the decomposition must be lossless-join

Example 1

	$R_1(B, C)$	$R_2(A, B)$
F_x	$B \rightarrow C$	$A \rightarrow B$

- **Consider R(A, B, C), F = {A→B , B→C}, decompose R into relations that are in BCNF.**

Alternative decomposition: To make the decomposition always lossless join, we can pick the FD **B→C** and make the decomposed relation as:

- R_1(**B,C**) – the attributes in the L.H.S. and R.H.S. of the FD.
- R_2(**A,B**) – the attribute(s) in the L.H.S. of the FD, and the remaining attributes that does not appear in R_1

R

A	B	C
1	1	2
2	1	2
3	1	2
4	1	2

R_1

B	C
1	2

R_2

A	B
1	1
2	1
3	1
4	1

Example 1

$F = \{A \rightarrow B, B \rightarrow C\}$ $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, \text{trivial FDs}\}$

	$R_1(B, C)$	$R_2(A, B)$
F_x	$B \rightarrow C$	$A \rightarrow B$

- **Decomposition: $R_1(B, C)$, $R_2(A, B)$**
- **Is $R_1(B, C)$ in BCNF?**
  - $F_1 = \{B \rightarrow C, \text{trivial FDs}\}$, it is a projection of F^+ on R_1.
  - Since $\{B\}^+ = \{B,C\} = R_1$, $\{B\}$ is a key in R_1.
  - Since all FDs in F_1 forms a key, R_1 is in BCNF.
- **Is $R_2(A, B)$ in BCNF?**
  - $F_2 = \{A \rightarrow B, \text{trivial FDs}\}$, it is a projection of F^+ on R_2.
  - Since $\{A\}^+ = \{A,B\} = R_2$, $\{A\}$ is a key in R_2.
  - Since all FDs in F_2 forms a key, R_2 is in BCNF.

R

A	B	C
1	1	2
2	1	2
3	1	2
4	1	2

R_1

B	C
1	2

R_2

A	B
1	1
2	1
3	1
4	1

Example 1

	$R_1(B, C)$	$R_2(A, B)$
F_x	$B \rightarrow C$	$A \rightarrow B$

- **Is the decomposition lossless join?**
  - From the illustration in example 1, the decomposition must be lossless join.
- **Is the decomposition dependency preserving ?**
  - $F = \{A \rightarrow B , B \rightarrow C\}$
  - $(F_1 \cup F_2) = (B \rightarrow C , A \rightarrow B)$
- Since $F = (F_1 \cup F_2)$, this implies $(F_1 \cup F_2)^+ = F^+$.
- **The decomposition is dependency preserving.**
  - That means if we insert a new tuple, if the new tuple does not violate F_1 in R_1, and F_2 in R_2, it won't violate F^+ in R.

R

A	B	C
1	1	2
2	1	2
3	1	2
4	1	2

R_1

B	C
1	2

R_2

A	B
1	1
2	1
3	1
4	1

Example 2

- **Is R_1(b_name, assets, b_city) in BCNF?**
  - F_1 = { {***b_name***} → {***assets***, ***b_city***}, trivial FDs} ← Projection of F^+ on F_1.
  - {***b_name***}$^+$ = {***b_name***, ***assets***, ***b_city***} = R_1, so {***b_name***} is a key in R_1.
  - Since all FD in F_1 forms a key in R_1, R_1 is in BCNF.
- **Is R_2(b_name, c_name, l_num, amount) in BCNF?**
  - F_2 = { {***l_num***} → {***amount***, ***b_name***} , {***l_num***, ***c_name***} → {all attributes} } ← Projection of F^+ on F_2.
  - {***l_num***}$^+$ = {***l_num***, ***amount***, ***b_name***} ≠ R_2, so {***b_name***} is NOT a key in R_2.
  - Since NOT all FD in F_2 forms a key in R_2, R_2 is NOT in BCNF.

Example 2

- **Picking {*l_num*} → {*amount, b_name*}, R_2 is further decomposed into:**
  - R_3(***l_num*, *amount*, *b_name***)
  - R_4(***c_name*, *l_num***)
- **Is R_3(l_num, amount, b_name) in BCNF?**
  - F_3 = {{***l_num***} → {***amount***, ***b_name***}, trivial FDs}
  - {***l_num***}$^+$ = {***l_num***, ***amount***, ***b_name***} = R_3, so {***l_num***} is a key in R_3.
  - Since all FD in F_3 forms a key in R_3, R_3 is in BCNF.

Example 2

- Is R_4(c_name, l_num) in BCNF?
  - F_4 = {trivial FDs}
  - Since all FD in F_4 forms a key in R_4, R_4 is in BCNF.
- Now, R_1, R_3 and R_4 are in BCNF;
- The decomposition is also lossless-join.

Example 2

- **The decomposition is also dependency preserving.**
  - F_1 = { {***b_name***} → {***assets***, ***b_city***}, trivial FDs}
  - F_3 = {{***l_num***} → {***amount***, ***b_name***}, trivial FDs}

{***l_num***} → {***b_name***} … (i)
by **Decomposition of** {***l_num***} → {***amount***, ***b_name***}

{***l_num***} → {***assets***, ***b_city***} … (ii)
by **Transitivity of** (i) and {***b_name***} → {***assets***, ***b_city***}

{***l_num***} → {***b_name*** ,***assets***, ***b_city, amount***} by **Union** of F_3 and (ii)

{***l_num***, ***c_name***} → {***l_num*** ,***c_name***, ***b_name*** ,***assets***, ***b_city, amount***} by **Augmentation**

  - Therefore $F_1 \cup F_3 \cup F_4 = F$, which implies $(F_1 \cup F_3 \cup F_4)^+ = F^+$.
  - The decomposition is **dependency preserving.**

BCNF doesn't imply dependency preserving

- **It is not always possible to get a BCNF decomposition that is dependency preserving.**
- Consider R(A, B, C); F = { AB→C, C→B }
- There are two candidate keys: {AB}, and {AC}.
  - $\{AB\}^+ = \{A,B,C\} = R$
  - $\{AC\}^+ = \{A,B,C\} = R$
- R is not in BCNF, since **C** is not a key.
- Decomposition of **R** must fail to preserve AB→C.

R

A	B	C
1	1	2
2	1	2
1	2	3

R_1

A	B
1	1
2	1
1	2

R_2

B	C
1	2
2	3

Not lossless decomposition

R_1

A	B
1	1
2	1
1	1

R_2

A	C
1	2
2	2
1	3

Not lossless decomposition

R_1

A	C
1	2
2	2
1	3

R_2

B	C
1	2
2	3

lossless

$F_1 = \{\emptyset\}$

$F_2 = \{C \rightarrow B\}$

Not dependency preserving

Motivating example

- **Back to our motivating example, we have:**
  - **Employees(** ***eid*****,** ***name*****,** ***parkingLot*****,** ***did*****,** ***since*****)**
  - **Departments(** ***did*****,** ***dname*****,** ***budget*****)**
- "Employees who work in the same department must park at the same ***parkingLot***." implies the following FD:

FD: did → parkingLot

- **Is Employees in BCNF?**
  - {***did***}$^+$ = {***parkingLot***} ≠ {***eid*****,** ***name*****,** ***parkingLot*****,** ***did*****,** ***since***}
  - Since ***did*** is not a key, **Employees** is NOT in BCNF.

Normalization

- **Employees**(***eid***, ***name***, ***parkingLot***, ***did***, ***since***) is decomposed to
  - **Employees2**(***eid***, ***name***, ***did***, ***since***)
  - **Dept_Lots**(***did***, ***parkingLot***)
- With **Departments**(***did***, ***dname***, ***budget***), the above two decomposed relations are further refined to
  - **Employees2**(***eid***, ***name***, ***did***, ***since***)
  - **Departments**(***did***, ***dname***, ***parkingLot***, ***budget***)

Good design: parking lots for all employees can be updated by changing their department-specific ***parkingLot***.

SCHOOL OF DATA SCIENCE

Conclusion

- **Relational database design goals**
  - Lossless-join
  - No redundancy (BCNF)
  - Dependency preservation
- **It is not always possible to satisfy the three goals.**
  - A lossless join, dependency preserving decomposition into BCNF may not always be possible.
- **SQL does not provide a direct way of specifying FDs other than superkeys.**
  - Can use assertions to check FD, but it is quite expensive.

Next Lecture

- Transactions