Week 6 Normalisation

http://en.wikipedia.org/wiki/Ursus_Wehrli

Housekeeping

- **Assignment 1 (SQL)**
  - The mark and feedback will be released on 13 Sep 2022.
  - A tailored database will be designed to reveal common issues of incorrect queries and made available to you as part of the feedback.
  - More drop-in sessions will be available after 14 Sep if you need any clarification on the marking of Assignment 1.

Housekeeping

- **Assignment 1 (SQL)**
  - The mark and feedback will be released on 13 Sep 2022.
  - A tailored database will be designed to reveal common issues of incorrect queries and made available to you as part of the feedback.
  - More drop-in sessions will be available after 14 Sep if you need any clarification on the marking of Assignment 1.
- **Thanks for participating in the anonymous survey in Week 5** and more information will be available on Wattle during the teaching break.
- **An optional exercise website for our course**

  `https://cs.anu.edu.au/dab/bench/db-exercises/`

Housekeeping

- **Assignment 1 (SQL)**
  - The mark and feedback will be released on 13 Sep 2022.
  - A tailored database will be designed to reveal common issues of incorrect queries and made available to you as part of the feedback.
  - More drop-in sessions will be available after 14 Sep if you need any clarification on the marking of Assignment 1.
- **Thanks for participating in the anonymous survey in Week 5** and more information will be available on Wattle during the teaching break.
- **An optional exercise website for our course**

  `https://cs.anu.edu.au/dab/bench/db-exercises/`
- **Enjoy the semester break!**

Decomposition vs Normalisation

Decomposin[illegible]n schema can possibly create more proble[illegible]olves!

Decomposition vs Normalisation

> Decomposin[illegible]n schema can possibly create more proble[illegible]olves!

- Thus, we need to consider two important questions:

  1. **Do we need to decompose a relation?**

Decomposition vs Normalisation

Decomposin[illegible]n schema can possibly create more proble[illegible]olves!

- Thus, we need to consider two important questions:

1. **Do we need to decompose a relation?**
   - **Several normal forms**

     $\hookrightarrow$ help us to decide whether or not to decompose a relation

Decomposition vs Normalisation

> Decomposing a relation schema can possibly create more problems than it solves!

- Thus, we need to consider two important questions:

1. **Do we need to decompose a relation?**
   - **Several normal forms**

     ↪ help us to decide whether or not to decompose a relation

2. **What problem (if any) does a given decomposition cause?**

Decomposition vs Normalisation

> Decomposi[illegible]n schema can possibly create more proble[illegible]olves!

- Thus, we need to consider two important questions:

1. **Do we need to decompose a relation?**
   - **Several normal forms**

     $\hookrightarrow$ help us to decide whether or not to decompose a relation

2. **What problem (if any) does a given decomposition cause?**

   - **Two properties**

     $\hookrightarrow$ help us to decide how to decompose a relation

Two Properties

- In addition to **data re**[illegible]t we need to consider the following properties when decomposing a relation:

Two Properties

- In addition to **data re**[illegible] we need to consider the following properties when deco[illegible] a relation:

  1. **Lossless join** – " **capture the same data** "

     To disallow the possibility of generating spurious tuples when a NATURAL JOIN operation is applied to the relations after decomposition.

Two Properties

- In addition to **data re**[illegible]t we need to consider the following properties when decomposing a relation:

1. **Lossless join** – " **capture the same data** "

   To disallow the possibility of generating spurious tuples when a NATURAL JOIN operation is applied to the relations after decomposition.

2. **Dependency preservation** – " **capture the same meta-data** "

   To ensure that each functional dependency can be inferred from functional dependencies after decomposition.

Lossless Join – Example

- **Lossless join** – " **ca… ame data** "
  To disallow the possi[illegible]erating spurious tuples when a NATURAL JOIN operation is app[illegible]elations after decomposition.

Lossless Join – Example

- **Lossless join** – " **ca[illegible]ame data** "
  To disallow the possi[illegible]erating spurious tuples when a NATURAL JOIN operation is app[illegible]elations after decomposition.

R		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

R_1	
Name	StudentID
Mike	123456
Mike	123458

R_2	
StudentID	DoB
123456	20/09/1989
123458	25/01/1988

- **Example 1:** Does the decomposition of R into R_1 and R_2 has the lossless join property?

Lossless Join – Example

- **Lossless join** – " **ca**[illegible]**ame data** "
  To disallow the possi[illegible]erating spurious tuples when a NATURAL JOIN operation is app[illegible]elations after decomposition.

R		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

R_1	
Name	StudentID
Mike	123456
Mike	123458

R_2	
StudentID	DoB
123456	20/09/1989
123458	25/01/1988

- **Example 1:** Does the decomposition of R into R_1 and R_2 has the lossless join property?
  Yes, because the natural join of R_1 and R_2 yields R.

Lossless Join – Example

- **Lossless join** – " **ca[illegible]ame data** "
  To disallow the possi[illegible]erating spurious tuples when a NATURAL JOIN operation is app[illegible]relations after decomposition.

R		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

R_3	
Name	StudentID
Mike	123456
Mike	123458

R_4	
Name	DoB
Mike	20/09/1989
Mike	25/01/1988

- **Example 2:** Does the decomposition of R into R_3 and R_4 has the lossless join property?

Australian National University

Lossless Join – Example

- **Lossless join** – " **ca[illegible]ame data** "
  To disallow the possi[illegible]erating spurious tuples when a NATURAL JOIN operation is app[illegible]relations after decomposition.

R		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

R_3	
Name	StudentID
Mike	123456
Mike	123458

R_4	
Name	DoB
Mike	20/09/1989
Mike	25/01/1988

- **Example 2:** Does the decomposition of R into R_3 and R_4 has the lossless join property?
  No, because the natural join of R_3 and R_4 generates spurious tuples.

Lossless Join – Example

- **Example 2:** The follo[illegible]mposition from R into R_3 and R_4 doesn't have the lossless joi[illegible]**t generates spurious tuples.**

R		
Name	<u>StudentID</u>	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

SELECT * FROM R_3 NATURAL JOIN R_4		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123456	25/01/1988
Mike	123458	20/09/1989
Mike	123458	25/01/1988

R_3	
Name	StudentID
Mike	123456
Mike	123458

R_4	
Name	DoB
Mike	20/09/1989
Mike	25/01/1988

Lossless Join – Example

- **Lossless join** – " **ca** [illegible] **ame data** "

R		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123458	25/01/1988

R_1	
Name	StudentID
Mike	123456
Mike	123458

R_2	
StudentID	DoB
123456	20/09/1989
123458	25/01/1988

Lossless join

R_3	
Name	StudentID
Mike	123456
Mike	123458

R_4	
Name	DoB
Mike	20/09/1989
Mike	25/01/1988

Not lossless join

Dependency Preservation – Example

- **Dependency preser**... ensure that each functional dependency can be inferred from ... dependencies after decomposition.

Dependency Preservation – Example

- **Dependency preser**[illegible] ensure that each functional dependency can be inferred from [illegible]ependencies after decomposition.
- **Example 1:** Given a [illegible]tID} → {Name} defined on R

R		
Name	<u>StudentID</u>	<u>CourseNo</u>
Mike	123456	COMP2400
Mike	123458	COMP2600

R_1	
Name	StudentID
Mike	123456
Mike	123458

R_2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

- Does the above decomposition preserves {StudentID} → {Name}?

Dependency Preservation – Example

- **Dependency preser**... ensure that each functional dependency can be inferred from ... dependencies after decomposition.
- **Example 1:** Given a ...ntID} $\rightarrow$ {Name} defined on R

R		
Name	StudentID	CourseNo
Mike	123456	COMP2400
Mike	123458	COMP2600

R_1	
Name	StudentID
Mike	123456
Mike	123458

R_2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

- Does the above decomposition preserves {StudentID} $\rightarrow$ {Name}? **Yes**, because {StudentID} and {Name} are both in R_1 after decomposition and thus {StudentID} $\rightarrow$ {Name} is preserved in R_1.

Dependency Preservation – Example

- **Dependency preser** ensure that each functional dependency can be inferred from lependencies after decomposition.
- **Example 2:** Given a ntID} → {Name} defined on R

R		
Name	StudentID	CourseNo
Mike	123456	COMP2400
Mike	123458	COMP2600

R_1	
Name	CourseNo
Mike	COMP2400
Mike	COMP2600

R_2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

- Does the above decomposition preserves {StudentID} → {Name}?

Dependency Preservation – Example

- **Dependency preser**... ensure that each functional dependency can be inferred from ... dependencies after decomposition.
- **Example 2:** Given a ...ntID} → {Name} defined on *R*

R		
Name	<u>StudentID</u>	<u>CourseNo</u>
Mike	123456	COMP2400
Mike	123458	COMP2600

R_1	
Name	CourseNo
Mike	COMP2400
Mike	COMP2600

R_2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

- Does the above decomposition preserves {StudentID} → {Name}? **No**, because {StudentID} and {Name} are not in a same relation after decomposition.

Dependency Preservation – Example

- **Dependency preser**[illegible] ensure that each functional dependency can be inferred from [illegible] dependencies after decomposition.
- **Example 3:** Given a [illegible] { {StudentID} → {Email}, {Email} → {Name}, {StudentID} → {Name} } defined on R

R		
Name	StudentID	Email
Mike	123456	123456@anu.edu.au
Tom	123123	123123@anu.edu.au

R_1	
Name	Email
Mike	123456@anu.edu.au
Tom	123123@anu.edu.au

R_2	
StudentID	Email
123456	123456@anu.edu.au
123123	123123@anu.edu.au

Dependency Preservation – Example

- **Dependency preser**[illegible] ensure that each functional dependency can be inferred from [illegible] ependencies after decomposition.
- **Example 3:** Given a [illegible] { {StudentID} → {Email}, {Email} → {Name}, {StudentID} → {Name} } defined on R

R		
Name	StudentID	Email
Mike	123456	123456@anu.edu.au
Tom	123123	123123@anu.edu.au

R_1	
Name	Email
Mike	123456@anu.edu.au
Tom	123123@anu.edu.au

R_2	
StudentID	Email
123456	123456@anu.edu.au
123123	123123@anu.edu.au

- Does the above decomposition preserves {StudentID} → {Name}?

Dependency Preservation – Example

- **Dependency preser**[illegible] ensure that each functional dependency can be inferred from [illegible] dependencies after decomposition.
- **Example 3:** Given a [illegible] { {StudentID} → {Email}, {Email} → {Name}, {StudentID} → {Name} } defined on R

R		
Name	<u>StudentID</u>	Email
Mike	123456	123456@anu.edu.au
Tom	123123	123123@anu.edu.au

R_1	
Name	Email
Mike	123456@anu.edu.au
Tom	123123@anu.edu.au

R_2	
StudentID	Email
123456	123456@anu.edu.au
123123	123123@anu.edu.au

- Does the above decomposition preserves {StudentID} → {Name}?
  Yes, because {StudentID} → {Name} can be inferred by {StudentID} → {Email} (preserved in R_2) and {Email} → {Name} (preserved in R_1).

Discussion

- If R with a set Σ of F… nposed into R_1 with Σ_1 and R_2 with Σ_2,

Discussion

- If R with a set Σ of F[illegible]mposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;

Discussion

- If R with a set Σ of F[illegible]nposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

Discussion

- If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?

Discussion

- If R with a set Σ of F[illegible]posed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \to B, B \to C, A \to C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \to B\}$ and $\Sigma_2 = \{A \to C\}$

Australian National University

Discussion

- If R with a set Σ of F[illegible]nposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \to B, B \to C, A \to C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \to B\}$ and $\Sigma_2 = \{A \to C\}$
  - **Lossless join?**

Discussion

- If R with a set Σ of F[illegible]nposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \to B, B \to C, A \to C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \to B\}$ and $\Sigma_2 = \{A \to C\}$
  - **Lossless join?** Yes because A is a superkey for R_1.

Discussion

- If R with a set Σ of F[illegible]mposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_2 = \{A \rightarrow C\}$
  - **Lossless join?** Yes because A is a superkey for R_1.
  - **Dependency preserving?**

Discussion

- If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_2 = \{A, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_2 = \{A \rightarrow C\}$
  - **Lossless join?** Yes because A is a superkey for R_1.
  - **Dependency preserving?** No because $(\Sigma_1 \cup \Sigma_2)^* \neq \Sigma^*$ from the fact that $\{A \rightarrow B, A \rightarrow C\} \not\models B \rightarrow C$.

Discussion

- If R with a set Σ of F[illegible]nposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into R[illegible] $= \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?

Discussion

- If R with a set Σ of F[illegible]mposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i[illegible]f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_3 = \{B \rightarrow C\}$

Discussion

- If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_3 = \{B \rightarrow C\}$
  - **Lossless join?**

Australian National University

Discussion

- If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \to B, B \to C, A \to C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \to B\}$ and $\Sigma_3 = \{B \to C\}$
  - **Lossless join?** Yes because B is a superkey for R_3.

Discussion

- If R with a set Σ of F… …nposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** i… …f the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_3 = \{B \rightarrow C\}$
  - **Lossless join?** Yes because B is a superkey for R_3.
  - **Dependency preserving?**

Australian National University

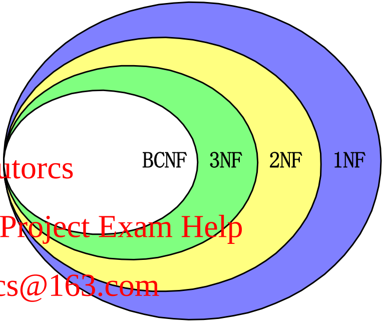
Discussion

- If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2,
  - **Lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2;
  - **Dependency preserving** if and only if $(\Sigma_1 \cup \Sigma_2)^* = \Sigma^*$ holds.

- Consider R={A, B, C} with the set of FDs $\Sigma = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Does the decompostion of R into $R_1 = \{A, B\}$ and $R_3 = \{B, C\}$ fullfill **lossless join** and **dependency preserving**?
  - $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_3 = \{B \rightarrow C\}$
  - **Lossless join?** Yes because B is a superkey for R_3.
  - **Dependency preserving?** Yes because $(\Sigma_1 \cup \Sigma_3)^* = \Sigma^*$ from the fact that $\{A \rightarrow B, B \rightarrow C\} \vDash A \rightarrow C$.

Normal Forms

Normal forms	Tes
1NF	
⇓	
2NF	
⇓	
3NF	⇓
⇓	
BCNF	strong
...	



- **Note that:**
  - 1NF is independent of keys and functional dependencies.
  - 2NF, 3NF and BCNF are based on keys and functional dependencies.
  - 4NF and 5NF are based on other dependencies (will not be covered in this course).

BCNF

Do not represent the same fact twice (within a relation)!

BCNF - Definition

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **super**

BCNF - Definition

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super
- When a relation schema is in **BCNF**, all data redundancy based on functional dependency are removed.

BCNF - Definition

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super

- When a relation schema is in **BCNF**, all data redundancy based on functional dependency are removed.
  - Here **data redundancy** is considered in terms of FDs, i.e., for a non-trivial FD $X \rightarrow Y$, there exists a relation R that contains two distinct tuples t_1 and t_2 with $t_1[XY] = t_2[XY]$.

{CourseName} $\rightarrow$ {Instructor}

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

Normalisation to BCNF

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super
- Consider the relation EACH with the following FD:
  - {CourseName} $\rightarrow$ {Instructor}.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Is TEACH in BCNF?

Normalisation to BCNF

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super
- Consider the relation ACH with the following FD:
  - {CourseName} $\rightarrow$ {Instructor}

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Is TEACH in BCNF?

Not in BCNF because {CourseName} is not a superkey.

Normalisation to BCNF

- A relation schema R [illegible] if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **super**[illegible]
- Consider the relation [illegible]EACH with the following FD:
  - {CourseName} $\rightarrow$ {Instructor}.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Is TEACH in BCNF?

  Not in BCNF because {CourseName} is not a superkey.

- Did we represent the same fact twice (or more times)?

Normalisation to BCNF

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then **X is a super**
- Consider the relation EACH with the following FD:
  - $\{$CourseName$\} \rightarrow \{$Instructor$\}$

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Is TEACH in BCNF?

  Not in BCNF because $\{$CourseName$\}$ is not a superkey.
- Did we represent the same fact twice (or more times)?

  Yes, the Instructor of Relational Databases is Yu.

Normalisation to BCNF

- **Algorithm** for a BCN[illegible]sition

  Input: a relation sch[illegible] a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

Normalisation to BCNF

- **Algorithm** for a BCNF decomposition

  Input: a relation schema R' and a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

  - Start with $\mathcal{S} = \{R'\}$;

Normalisation to BCNF

- **Algorithm** for a BCN[illegible]sition

  Input: a relation sch[illegible] a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

  - Start with $\mathcal{S} = \{R'\}$;
  - Do the following for each $R \in \mathcal{S}$ iteratively until no changes on $\mathcal{S}$:
    - Find a (non-trivial) FD $X \rightarrow Y$ on R that violates BCNF, if any;
    - Replace R in $\mathcal{S}$ by two relation schemas XY and $(R - Y)$ and project the FDs to these two relation schemas.

Normalisation to BCNF

- **Algorithm** for a BCNF decomposition

  Input: a relation schema R' and a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

  - Start with $\mathcal{S} = \{R'\}$;
  - Do the following for each $R \in \mathcal{S}$ iteratively until no changes on $\mathcal{S}$:
    - Find a (non-trivial) FD $X \rightarrow Y$ on R that violates BCNF, if any;
    - Replace R in $\mathcal{S}$ by two relation schemas XY and $(R - Y)$ and project the FDs to these two relation schemas.

- Does the above **Algorithm** always produce a lossless decomposition?

Normalisation to BCNF

- **Algorithm** for a BCNF decomposition

  Input: a relation schema R' and a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

  - Start with $\mathcal{S} = \{R'\}$;
  - Do the following for each $R \in \mathcal{S}$ iteratively until no changes on $\mathcal{S}$:
    - Find a (non-trivial) FD $X \rightarrow Y$ on R that violates BCNF, if any;
    - Replace R in $\mathcal{S}$ by two relation schemas XY and $(R - Y)$ and project the FDs to these two relation schemas.

- Does the above **Algorithm** always produce a lossless decomposition?

  If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2, this decomposition is **lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2.

Normalisation to BCNF

- **Algorithm** for a BCNF decomposition

  Input: a relation schema R' and a set Σ of FDs on R'.

  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

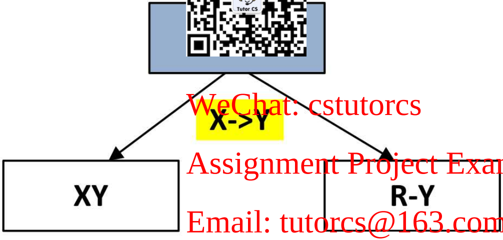
  - Start with $\mathcal{S} = \{R'\}$;
  - Do the following for each $R \in \mathcal{S}$ iteratively until no changes on $\mathcal{S}$:
    - Find a (non-trivial) FD $X \rightarrow Y$ on R that violates BCNF, if any;
    - Replace R in $\mathcal{S}$ by two relation schemas XY and $(R - Y)$ and project the FDs to these two relation schemas.

- Does the above **Algorithm** always produce a lossless decomposition?

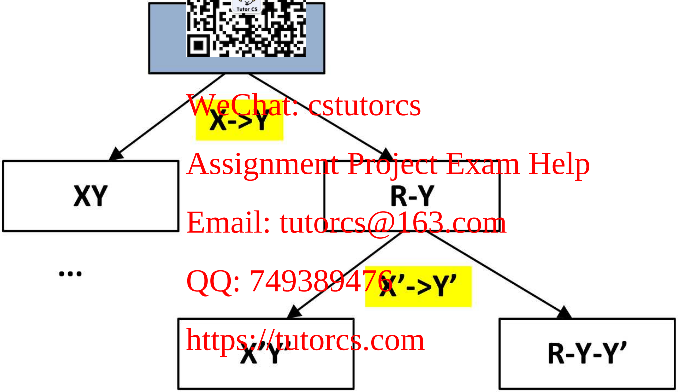
  If R with a set Σ of FDs is decomposed into R_1 with Σ_1 and R_2 with Σ_2, this decomposition is **lossless join** if and only if the common attributes of R_1 and R_2 are a superkey for R_1 or R_2.

- **Yes because X is a superkey for XY.**

Normalisation to BCNF



Normalisation to BCNF



Normalisation to BCNF

- A relation schema R [illegible] if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super[illegible]
- Consider the relation [illegible]EACH with the following FD:

$\{\text{CourseName}\} \rightarrow \{\text{Instructor}\}$

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

Normalisation to BCNF

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super
- Consider the relation TEACH with the following FD:

$\{\text{CourseName}\} \rightarrow \{\text{Instructor}\}$

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Can we normalise TEACH into BCNF?

Normalisation to BCNF

- A relation schema R [illegible] if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a super[illegible]
- Consider the relation [illegible]EACH with the following FD:

$\{\text{CourseName}\} \rightarrow \{\text{Instructor}\}$

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Can we normalise TEACH into BCNF? **Yes**.

R_1

CourseName	Instructor
Operating Systems	Hegel
Relational Databases	Yu

R_2

StudentID	CourseName
u123456	Operating Systems
u234566	Relational Databases
u234567	Relational Databases

Normalisation to BCNF

- A relation schema R if whenever a non-trivial FD $X \rightarrow A$ holds in R, then **X is a super**
- Consider the relation ACH with the following FD:

$\{\text{CourseName}\} \rightarrow \{\text{Instructor}\}$

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Hegel
u234566	Relational Databases	Yu
u234567	Relational Databases	Yu

- Can we normalise TEACH into BCNF? **Yes**.

R_1

CourseName	Instructor
Operating Systems	Hegel
Relational Databases	Yu

R_2

StudentID	CourseName
u123456	Operating Systems
u234566	Relational Databases
u234567	Relational Databases

- **Do not represent the same fact twice (within a relation)!**

BCNF - Exercise

- Consider INTERVIEW , CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

- Is INTERVIEW in BCNF? If not, normalize INTERVIEW into BCNF.

BCNF - Exercise

- Consider INTERVIEW, CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

- Is INTERVIEW in BCNF? If not, normalize INTERVIEW into BCNF.
  - {CustomerID, Date}, {OfficerID, Date, Time}, and {Date, Time, Room} are the keys.

BCNF - Exercise

- Consider INTERVIEW , CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}
- Is INTERVIEW in BCNF? If not, normalize INTERVIEW into BCNF.
  - {CustomerID, Date}, {OfficerID, Date, Time}, and {Date, Time, Room} are the keys.
  - Any superkey must contain one of these keys as a subset.

BCNF - Exercise

- Consider INTERVIEW [illegible], CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}
- Is INTERVIEW in BCNF? If not, normalize INTERVIEW into BCNF.
  - {CustomerID, Date}, {OfficerID, Date, Time}, and {Date, Time, Room} are the keys.
  - Any superkey must contain one of of these keys as a subset.
- **INTERVIEW is not in BCNF because {OfficerID, Date} → {Room} and {OfficerID, Date} is not a superkey.**

BCNF - Exercise

- We decompose INTE[illegible]g the FD: {OfficerID, Date} → {Room}:

[illegible]V				
OfficerID	CustomerID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
S1011	P105	12/11/2013	12:00	R15
S1024	P108	14/11/2013	14:00	R10
S1024	P107	14/11/2013	14:00	R10

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

Australian National University

BCNF - Exercise

- We decompose INTERVIEW using the FD: {OfficerID, Date} → {Room}:

INTERVIEW				
OfficerID	CustomerID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
S1011	P105	12/11/2013	12:00	R15
S1024	P108	14/11/2013	14:00	R10
S1024	P107	14/11/2013	14:00	R10

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- **Do not represent the same fact twice (within a relation)!**

BCNF - Exercise

- Consider INTERVIEW[illegible], CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date[illegible]m}
  - {CustomerID, D[illegible]fficerID, Time}
  - {OfficerID, Date, Time} $\rightarrow$ {CustomerID}
  - {Date, Time, Room} $\rightarrow$ {CustomerID}

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- Project FDs on two new relation schemas.

BCNF - Exercise

- Consider INTERVIEW[illegible], CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- Project FDs on two new relation schemas:

  INTERVIEW1: {OfficerID, Date} → {Room}

BCNF - Exercise

- Consider INTERVIEW={OfficerID, CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date} → {Room}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- Project FDs on two new relation schemas:

  INTERVIEW1: {OfficerID, Date} → {Room}

  INTERVIEW2: {CustomerID, Date} → {OfficerID, Time}, {OfficerID, Date, Time} → {CustomerID}.

BCNF - Exercise

- Consider INTERVIEW[illegible], CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date[illegible]n}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

INTERVIEW1

OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2

OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- Is this decomposition dependency-preservation?

BCNF - Exercise

- Consider INTERVIEW[illegible], CustomerID, Date, Time, Room} with the following FDs:
  - {OfficerID, Date[illegible]n}
  - {CustomerID, Date} → {OfficerID, Time}
  - {OfficerID, Date, Time} → {CustomerID}
  - {Date, Time, Room} → {CustomerID}

INTERVIEW1		
OfficerID	Date	Room
S1011	12/11/2013	R15
S1024	14/11/2013	R10

INTERVIEW2			
OfficerID	CustomerID	Date	Time
S1011	P100	12/11/2013	10:00
S1011	P105	12/11/2013	12:00
S1024	P108	14/11/2013	14:00
S1024	P107	14/11/2013	14:00

- Is this decomposition dependency-preservation?
  No because {Date, Time, Room} → {CustomerID} is lost (and cannot be recovered)!

BCNF - Order Does Matter

- When applying BCN[illegible]sition, the order in which the FDs are applied may lead to different results.

BCNF - Order Does Matter

- When applying BCNF decomposition, the order in which the FDs are applied may lead to different results.

- **Example:** Consider $R = \{A, B, C\}$ and $\{A \rightarrow B, C \rightarrow B, B \rightarrow C\}$.

BCNF - Order Does Matter

- When applying BCNF decomposition, the order in which the FDs are applied may lead to different results.

- **Example:** Consider $R = \{A, B, C\}$ and $\{A \rightarrow B, C \rightarrow B, B \rightarrow C\}$.
  - **Case 1**: (Using $C \rightarrow B$ first)

BCNF - Order Does Matter

- When applying BCNF decomposition, the order in which the FDs are applied may lead to different results.

- **Example:** Consider $R = \{A, B, C\}$ and $\{A \rightarrow B, C \rightarrow B, B \rightarrow C\}$.
  - **Case 1**: (Using $C \rightarrow B$ first)

$$R_1 = \{B, C\}, \Sigma_1 = \{B \rightarrow C, C \rightarrow B\}, R_2 = \{A, C\}, \Sigma_2 = \{A \rightarrow C\}$$

BCNF - Order Does Matter

- When applying BCNF decomposition, the order in which the FDs are applied may lead to different results.

- **Example:** Consider $R = \{A, B, C\}$ and $\{A \rightarrow B, C \rightarrow B, B \rightarrow C\}$.
  - **Case 1**: (Using $C \rightarrow B$ first)

$$R_1 = \{B, C\}, \Sigma_1 = \{B \rightarrow C, C \rightarrow B\}, R_2 = \{A, C\}, \Sigma_2 = \{A \rightarrow C\}$$

  - **Case 2:** (Using $B \rightarrow C$ first)

BCNF - Order Does Matter

- When applying BCNF decomposition, the order in which the FDs are applied may lead to different results.

- **Example:** Consider $R = \{A, B, C\}$ and $\{A \rightarrow B, C \rightarrow B, B \rightarrow C\}$.
  - **Case 1**: (Using $C \rightarrow B$ first)

$$R_1 = \{B, C\}, \Sigma_1 = \{B \rightarrow C, C \rightarrow B\}; R_2 = \{A, C\}, \Sigma_2 = \{A \rightarrow C\}$$

  - **Case 2:** (Using $B \rightarrow C$ first)

$$R_1' = \{B, C\}, \Sigma_1' = \{B \rightarrow C, C \rightarrow B\}; R_2' = \{A, B\}, \Sigma_2' = \{A \rightarrow B\};$$

Lossless Join & Dependency Preservation

- So far, we know how to find a lossless BCNF-decomposition, but it may not be dependency-preserving.

- Is there **a less restrictive normal form** such that a lossless and dependency-preserving decomposition can always be found?

Lossless Join & Dependency Preservation

- So far, we know how to find a lossless BCNF-decomposition, but it may not be dependency-preserving.
- Is there **a less restrictive normal form** such that a lossless and dependency-preserving decomposition can always be found?

  Yes, refer to 3NF.

3NF - Definition

- A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.

- Question: If R is in **BCNF**, then R is in **3NF**?

3NF - Definition

- A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then *X* is a **superkey** or *A* is a **prime attribute**.

- Question: If *R* is in **BCNF**, then *R* is in **3NF**?

  Yes

3NF - Definition

- A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then **X is a superkey** or **A is a prime attribute**.

- Question: If R is in **BCNF**, then R is in **3NF**?

  Yes

- 3NF **preserves all the functional dependencies** at the cost of **allowing some data redundancy**.

Normalisation to 3NF

- Consider the followin[illegible]NROL:
  - {StudentID, Co[illegible]mester} → {ConfirmedBy_ID, StaffName};
  - {ConfirmedBy_[illegible]ffName}.

ENROL

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
123456	COMP2400	2010 S2	u12	Jane
123458	COMP2400	2008 S2	u13	Linda
123458	COMP2600	2008 S2	u13	Linda

Normalisation to 3NF

- Consider the followin[illegible]NROL:
  - {StudentID, Co[illegible]mester} → {ConfirmedBy_ID, StaffName};
  - {ConfirmedBy_[illegible]ffName}.

ENROL

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
123456	COMP2400	2010 S2	u12	Jane
123458	COMP2400	2008 S2	u13	Linda
123458	COMP2600	2008 S2	u13	Linda

- **Is ENROL in 3NF?**

Australian National University

Normalisation to 3NF

- Consider the followin NROL:
  - {StudentID, Co mester} → {ConfirmedBy_ID, StaffName};
  - {ConfirmedBy_ ffName}.

ENROL

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
123456	COMP2400	2010 S2	u12	Jane
123458	COMP2400	2008 S2	u13	Linda
123458	COMP2600	2008 S2	u13	Linda

- **Is ENROL in 3NF?**
  - {StudentID, CourseNo, Semester} is the only key.

Normalisation to 3NF

- Consider the followin NROL:
  - {StudentID, Co mester} $\rightarrow$ {ConfirmedBy_ID, StaffName};
  - {ConfirmedBy_ ffName}.

ENROL

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
123456	COMP2400	2010 S2	u12	Jane
123458	COMP2400	2008 S2	u13	Linda
123458	COMP2600	2008 S2	u13	Linda

- **Is ENROL in 3NF?**
  - {StudentID, CourseNo, Semester} is the only key.

A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.

Normalisation to 3NF

- Consider the following [illegible] ENROL:
  - {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName};
  - {ConfirmedBy_ID} → {StaffName}.

ENROL

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
123456	COMP2400	2010 S2	u12	Jane
123458	COMP2400	2008 S2	u13	Linda
123458	COMP2600	2008 S2	u13	Linda

- **Is ENROL in 3NF?**
  - {StudentID, CourseNo, Semester} is the only key.

    A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.
  - **Not in 3NF,** because of {ConfirmedBy_ID} → {StaffName}: {ConfirmedBy_ID} is NOT a **superkey** and {StaffName} is NOT a **prime attribute**.

Normalisation to 3NF

- **Algorithm** for a dependency-preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

Normalisation to 3NF

- **Algorithm** for a dependency-preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$

Normalisation to 3NF

- **Algorithm** for a dependency preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$
  - Group FDs in Σ' by their left-hand-side attribue sets

Normalisation to 3NF

- **Algorithm** for a dependency-preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$
  - Group FDs in Σ' by their left-hand-side attribue sets
  - For each distinct left-hand-side X_i of FDs in Σ' that includes $X_i \rightarrow A_1, X_i \rightarrow A_2, \ldots, X_i \rightarrow A_k$:
    - Add $R_i = X_i \cup \{A_1\} \cup \{A_2\} \cdots \cup \{A_k\}$ to $\mathcal{S}$

Normalisation to 3NF

- **Algorithm** for a dependency-preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$
  - Group FDs in Σ' by their left-hand-side attribue sets
  - For each distinct left-hand-side X_i of FDs in Σ' that includes $X_i \to A_1, X_i \to A_2, \ldots, X_i \to A_k$:
    - Add $R_i = X_i \cup \{A_1\} \cup \{A_2\} \cdots \cup \{A_k\}$ to $\mathcal{S}$
  - Remove all redundant ones from $\mathcal{S}$ (i.e., remove R_i if $R_i \subseteq R_j$)

Normalisation to 3NF

- **Algorithm** for a depe[illegible]eserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$
  - Group FDs in Σ' by their left-hand-side attribue sets
  - For each distinct left-hand-side X_i of FDs in Σ' that includes $X_i \to A_1, X_i \to A_2, \ldots, X_i \to A_k$:
    - Add $R_i = X_i \cup \{A_1\} \cup \{A_2\} \cdots \cup \{A_k\}$ to $\mathcal{S}$
  - Remove all redundant ones from $\mathcal{S}$ (i.e., remove R_i if $R_i \subseteq R_j$)
  - if $\mathcal{S}$ does not contain a superkey of R, add a key of R as R_0 into $\mathcal{S}$.

Normalisation to 3NF

- **Algorithm** for a dependency preserving and lossless 3NF-decomposition

  Input: a relation schema R and a set Σ of FDs on R.

  Output: a set $\mathcal{S}$ of relation schemas in 3NF, each having a set of FDs

  - Compute a **minimal cover** Σ' for Σ and start with $\mathcal{S} = \phi$
  - Group FDs in Σ' by their left-hand-side attribue sets
  - For each distinct left-hand-side X_i of FDs in Σ' that includes $X_i \rightarrow A_1, X_i \rightarrow A_2, \ldots, X_i \rightarrow A_k$:
    - Add $R_i = X_i \cup \{A_1\} \cup \{A_2\} \cdots \cup \{A_k\}$ to $\mathcal{S}$
  - Remove all redundant ones from $\mathcal{S}$ (i.e., remove R_i if $R_i \subseteq R_j$)
  - if $\mathcal{S}$ does not contain a superkey of R, add a key of R as R_0 into $\mathcal{S}$.
  - Project the FDs in Σ' onto each relation schema in $\mathcal{S}$

Normalisation to 3NF

R

$R_1 = X_1 A_1 \ldots A_K$ … $R_n = X_n A$

$X_1 \rightarrow A_1$

…

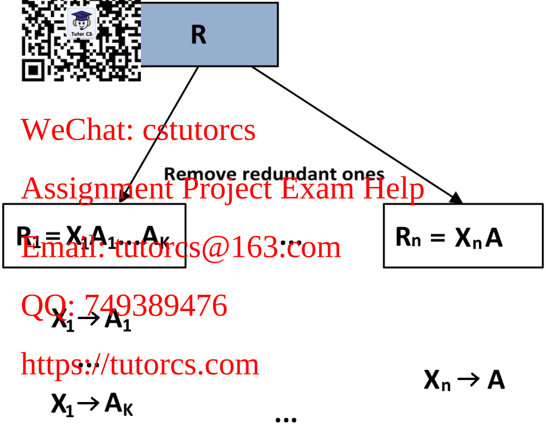
$X_1 \rightarrow A_K$

A minimal cover

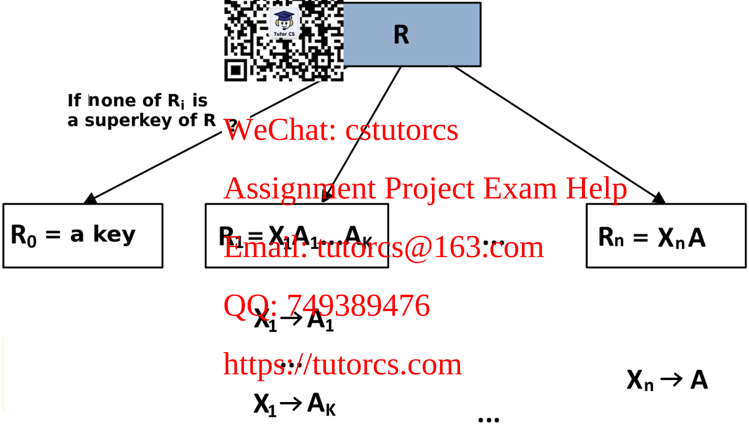
…

$X_n \rightarrow A$

Normalisation to 3NF

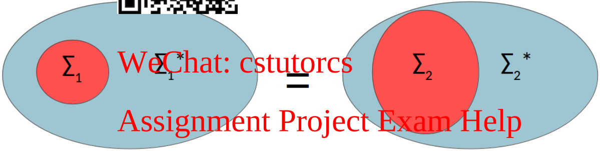


Normalisation to 3NF



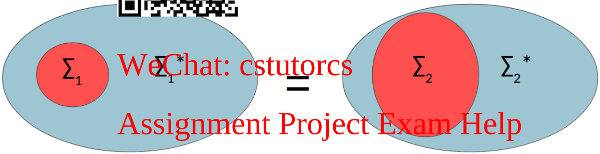
Equivalence of Functional Dependencies

- Σ_1 and Σ_2 are **equiv** Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \models \Sigma_2$ an



Equivalence of Functional Dependencies

- Σ_1 and Σ_2 are **equiv** Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \vDash \Sigma_2$ an



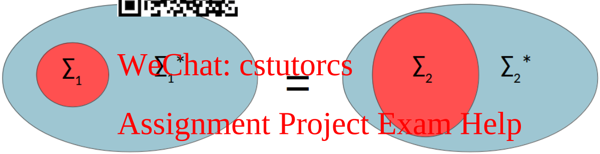
- **Example 1:**

  $\Sigma_1 = \{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, Y \rightarrow Z\}$

  If $\Sigma_1^* = \Sigma_2^*$,

Equivalence of Functional Dependencies

- Σ_1 and Σ_2 are **equiv** ... Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \vDash \Sigma_2$ an ...



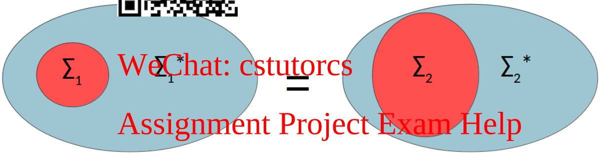
- **Example 1:**

  $\Sigma_1 = \{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, Y \rightarrow Z\}$

  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**

Equivalence of Functional Dependencies

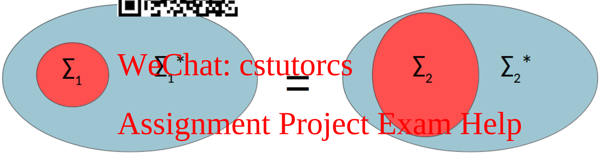
- Σ_1 and Σ_2 are **equiv** ... Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \models \Sigma_2$ an ...



- **Example 1:**
  $\Sigma_1 = \{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, Y \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**
- **Example 2:**
  $\Sigma_1 = \{X \rightarrow Y, XY \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, X \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$,

Equivalence of Functional Dependencies

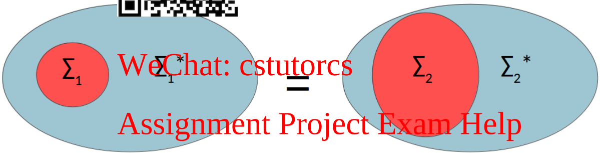
- Σ_1 and Σ_2 are **equiv** Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \vDash \Sigma_2$ an



- **Example 1:**
  $\Sigma_1 = \{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, Y \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**
- **Example 2:**
  $\Sigma_1 = \{X \rightarrow Y, XY \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, X \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**

Equivalence of Functional Dependencies

- Σ_1 and Σ_2 are **equiv**[illegible] Σ_2^*.
- $\Sigma_1^* = \Sigma_2^*$ if $\Sigma_1 \models \Sigma_2$ an[illegible]



- **Example 1:**
  $\Sigma_1 = \{X \rightarrow Y, Y \rightarrow Z, X \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, Y \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**
- **Example 2:**
  $\Sigma_1 = \{X \rightarrow Y, XY \rightarrow Z\}$ and $\Sigma_2 = \{X \rightarrow Y, X \rightarrow Z\}$
  If $\Sigma_1^* = \Sigma_2^*$, then Σ_1 is not **minimal**
- **Questions:** Can we find the **minimal** one among equivalent sets of FDs?

Minimal Cover – The Hard Part!

Minimal Cover – The Hard Part!

- Let Σ be a set of FDs. A **minimal cover** Σ_m of Σ is a set of FDs such that
  1. Σ_m is equivalent to Σ, i.e., start with $\Sigma_m = \Sigma$;

Minimal Cover – The Hard Part!

- Let Σ be a set of FDs. A **minimal cover** Σ_m of Σ is a set of FDs such that

  1. Σ_m is equivalent to Σ, i.e., start with $\Sigma_m = \Sigma$;
  2. **Dependent:** each FD in Σ_m has only a single attribute on its right hand side, i.e., replace each FD $X \rightarrow \{A_1, \ldots, A_k\}$ in Σ_m with $X \rightarrow A_1, \ldots, X \rightarrow A_k$;

Minimal Cover – The Hard Part!

- Let Σ be a set of FDs **al cover** Σ_m of Σ is a set of FDs such that

  1. Σ_m is equivalent to Σ, i.e., start with $\Sigma_m = \Sigma$;

  2. **Dependent:** each FD in Σ_m has only a single attribute on its right hand side, i.e., replace each FD $X \rightarrow \{A_1, \ldots, A_k\}$ in Σ_m with $X \rightarrow A_1, \ldots, X \rightarrow A_k$;

  3. **Determinant:** each FD has as few attributes on the left hand side as possible, i.e., for each FD $X \rightarrow A$ in Σ_m, check each attribute B of X to see if we can replace $X \rightarrow A$ with $(X - B) \rightarrow A$ in Σ_m;

Minimal Cover – The Hard Part!

- Let Σ be a set of FDs. A **minimal cover** Σ_m of Σ is a set of FDs such that

  1. Σ_m is equivalent to Σ, i.e., start with $\Sigma_m = \Sigma$;
  2. **Dependent:** each FD in Σ_m has only a single attribute on its right hand side, i.e., replace each FD $X \rightarrow \{A_1, \ldots, A_k\}$ in Σ_m with $X \rightarrow A_1, \ldots, X \rightarrow A_k$;
  3. **Determinant:** each FD has as few attributes on the left hand side as possible, i.e., for each FD $X \rightarrow A$ in Σ_m, check each attribute B of X to see if we can replace $X \rightarrow A$ with $(X - B) \rightarrow A$ in Σ_m;
  4. Remove a FD from Σ_m if it is redundant.

Minimal Cover - Examples

- Given the set of FDs
  {StudentID, CourseNumber} → {ConfirmedBy_ID, StaffName}
  {ConfirmedBy_ID} → {StaffName}
- we can compute the minimal cover of Σ as follows:

Minimal Cover - Examples

- Given the set of FDs Σ:
  $\{StudentID, CourseNumber\} \rightarrow \{ConfirmedBy\_ID, StaffName\}$
  $\{ConfirmedBy\_ID\} \rightarrow \{StaffName\}$
- we can compute the minimal cover of Σ as follows:
  1. start from Σ;

Minimal Cover - Examples

- Given the set of FDs [illegible]
  {StudentID, CourseN[illegible]er} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  {ConfirmedBy_ID} $\rightarrow$ [illegible]e}
- we can compute the [illegible]ver of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;

Minimal Cover - Examples

- Given the set of FDs Σ:
  {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  {ConfirmedBy_ID} $\rightarrow$ {StaffName}
- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
     {StudentID, CourseNo, Semester} $\rightarrow$ { ConfirmedBy_ID, StaffName }

Australian National University

Minimal Cover - Examples

- Given the set of FDs Σ:
  {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  {ConfirmedBy_ID} $\rightarrow$ {StaffName}
- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
     {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
     can be replaced by
     {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}
     {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}

Minimal Cover - Examples

- Given the set of FDs [illegible]
  - {StudentID, CourseN[illegible]er} → {ConfirmedBy_ID}
  - {StudentID, CourseN[illegible]er} → {StaffName}
  - {ConfirmedBy_ID} → {StaffName}
- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;

Minimal Cover - Examples

- Given the set of FDs

  {StudentID, CourseNumber} $\rightarrow$ {ConfirmedBy_ID}

  {StudentID, CourseNumber} $\rightarrow$ {StaffName}

  {ConfirmedBy_ID} $\rightarrow$ {StaffName}

- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
  3. check whether all the FDs in Σ have redundant attribute on the left hand side;

Minimal Cover - Examples

- Given the set of FDs Σ:

  {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}

  {StudentID, CourseNo, Semester} → {StaffName}

  {ConfirmedBy_ID} → {StaffName}

- we can compute the minimal cover of Σ as follows:

  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
  3. check whether all the FDs in Σ have redundant attribute on the left hand side;

     check if { StudentID, CourseNo, Semester } → {ConfirmedBy_ID} is minimal with respect to the left hand side

     check if { StudentID, CourseNo, Semester } → {StaffName} is minimal with respect to the left hand side

Minimal Cover - Examples

- Given the set of FDs Σ:
  $\{$StudentID, CourseNo, Semester$\} \rightarrow \{$ConfirmedBy_ID$\}$
  $\{$StudentID, CourseNo, Semester$\} \rightarrow \{$StaffName$\}$
  $\{$ConfirmedBy_ID$\} \rightarrow \{$StaffName$\}$
- we can compute the minimal cover of Σ as follows:

  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;

  3. check whether all the FDs in Σ have redundant attribute on the left hand side;

     check if $\{$StudentID, CourseNo, Semester$\} \rightarrow \{$ConfirmedBy_ID$\}$ is minimal with respect to the left hand side

     check if $\{$StudentID, CourseNo, Semester$\} \rightarrow \{$StaffName$\}$ is minimal with respect to the left hand side

     All look good!

Minimal Cover - Examples

- Given the set of FDs Σ:

  {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}

  {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}

  {ConfirmedBy_ID} $\rightarrow$ {StaffName}

- we can compute the minimal cover of Σ as follows:

  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
  3. check whether all the FDs in Σ have redundant attribute on the left hand side;

Minimal Cover - Examples

- Given the set of FDs Σ:

  {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}
  {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}
  {ConfirmedBy_ID} $\rightarrow$ {StaffName}

- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
  3. check whether all the FDs in Σ have redundant attribute on the left hand side;
  4. look for a redundant FD in { {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}, {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}, {ConfirmedBy_ID} $\rightarrow$ {StaffName} }

Minimal Cover - Examples

- Given the set of FDs Σ:

  {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}
  {StudentID, CourseNo, Semester} → {StaffName}
  {ConfirmedBy_ID} → {StaffName}

- we can compute the minimal cover of Σ as follows:
  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;
  3. check whether all the FDs in Σ have redundant attribute on the left hand side;
  4. look for a redundant FD in { {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}, {StudentID, CourseNo, Semester} → {StaffName}, {ConfirmedBy_ID} → {StaffName} }
     - {StudentID, CourseNo, Semester} → {StaffName} is redundant and thus is removed

Minimal Cover - Examples

- Given the set of FDs Σ:

  {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}

  {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}

  {ConfirmedBy_ID} $\rightarrow$ {StaffName}

- we can compute the minimal cover of Σ as follows:

  1. start from Σ;
  2. check whether all the FDs in Σ have only one attribute on the right hand side;

  3. check whether all the FDs in Σ have redundant attribute on the left hand side;

  4. look for a redundant FD in { {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}, {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName}, {ConfirmedBy_ID} $\rightarrow$ {StaffName} }

     - {StudentID, CourseNo, Semester} $\rightarrow$ {StaffName} is redundant and thus is removed

  5. Therefore, the minial cover of Σ is { {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}, {ConfirmedBy_ID} $\rightarrow$ {StaffName}}

Normalisation to 3NF – Example

- Consider ENROL again
  - {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} → {StaffName}

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
...	...	...	...	...

- **Can we normalise ENROL into 3NF by a lossless and dependency preserving decomposition?**

Normalisation to 3NF – Example

- Consider ENROL aga
  - {StudentID, Co mester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ ffName}

<u>StudentID</u>	<u>CourseNo</u>	<u>Semester</u>	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}, {ConfirmedBy_ID} $\rightarrow$ {StaffName}}.
- Hence, we have:

Normalisation to 3NF – Example

- Consider ENROL aga
  - {StudentID, Co mester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ ffName}

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}, {ConfirmedBy_ID} $\rightarrow$ {StaffName}}.
- Hence, we have:
  - R_1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID}

Normalisation to 3NF – Example

- Consider ENROL again:
  - {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} → {StaffName}

<u>StudentID</u>	<u>CourseNo</u>	<u>Semester</u>	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} → {ConfirmedBy_ID}, {ConfirmedBy_ID} → {StaffName}}.

- Hence, we have:
  - R_1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}
  - R_2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} → {StaffName}

Normalisation to 3NF – Example

- Consider ENROL again:
  - {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} → {StaffName}

<u>StudentID</u>	<u>CourseNo</u>	<u>Semester</u>	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} → {ConfirmedBy_ID}, {ConfirmedBy_ID} → {StaffName}}.

- Hence, we have:
  - R_1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}
  - R_2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} → {StaffName}
  - Omit R_0 because R_1 is a superkey of ENROL.

Normalisation to 3NF – Example

- Consider ENROL aga
  - {StudentID, Co mester} → {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ ffName}

<u>StudentID</u>	<u>CourseNo</u>	<u>Semester</u>	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} → {ConfirmedBy_ID}, {ConfirmedBy_ID} → {StaffName}}.

- Hence, we have:

  - R_1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}

  - R_2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} → {StaffName}

  - Omit R_0 because R_1 is a superkey of ENROL.
- Is {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName} preserved?

Normalisation to 3NF – Example

- Consider ENROL again:
  - {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} → {StaffName}

StudentID	CourseNo	Semester	ConfirmedBy_ID	StaffName
...	...	...	...	...

- A **minimal cover** is {{StudentID, CourseNo, Semester} → {ConfirmedBy_ID}, {ConfirmedBy_ID} → {StaffName}}.

- Hence, we have:
  - R_1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}
  - R_2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} → {StaffName}
  - Omit R_0 because R_1 is a superkey of ENROL.
- Is {StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName} preserved? **Yes**.

Normalisation to 3NF – Example

- Consider INTERVIEW

INTERVIEW				
OfficerID	CustomerID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
...	...	...	...	...

1. $\{\text{OfficerID, Date}\} \rightarrow \{\text{Room}\}$
2. $\{\text{CustomerID, Date}\} \rightarrow \{\text{OfficerID, Time}\}$
3. $\{\text{OfficerID, Date, Time}\} \rightarrow \{\text{CustomerID}\}$
4. $\{\text{Date, Time, Room}\} \rightarrow \{\text{CustomerID}\}$

- **Is INTERVIEW in 3NF? If not, normalise INTERVIEW into lossless and dependency preserving 3NF.**

Normalisation to 3NF – Example

- Consider INTERVIEW:

INTERVIEW				
OfficerID	[illegible]ID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
...	...	...	...	...

1. {OfficerID, Date} → {Room}
2. {CustomerID, Date} → {OfficerID, Time}
3. {OfficerID, Date, Time} → {CustomerID}
4. {Date, Time, Room} → {CustomerID}

- **Is INTERVIEW in 3NF? If not, normalise INTERVIEW into lossless and dependency preserving 3NF.**
  - A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.

Normalisation to 3NF – Example

- Consider INTERVIEW:

INTERVIEW				
OfficerID	CustomerID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
...	...	...	...	...

1. {OfficerID, Date} $\rightarrow$ {Room}
2. {CustomerID, Date} $\rightarrow$ {OfficerID, Time}
3. {OfficerID, Date, Time} $\rightarrow$ {CustomerID}
4. {Date, Time, Room} $\rightarrow$ {CustomerID}

- **Is INTERVIEW in 3NF? If not, normalise INTERVIEW into lossless and dependency preserving 3NF.**
  - A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.
  - We know that {CustomerID, Date}, {OfficerID, Date, Time}, and {Date, Time, Room} are the keys.

Normalisation to 3NF – Example

- Consider INTERVIEW:

INTERVIEW				
OfficerID	CustomerID	Date	Time	Room
S1011	P100	12/11/2013	10:00	R15
...	...	...	...	...

1. {OfficerID, Date} $\rightarrow$ {Room}
2. {CustomerID, Date} $\rightarrow$ {OfficerID, Time}
3. {OfficerID, Date, Time} $\rightarrow$ {CustomerID}
4. {Date, Time, Room} $\rightarrow$ {CustomerID}

- **Is INTERVIEW in 3NF? If not, normalise INTERVIEW into lossless and dependency preserving 3NF.**

  - A relation schema R is in **3NF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.
  - We know that {CustomerID, Date}, {OfficerID, Date, Time}, and {Date, Time, Room} are the keys.

INTERVIEW is in 3NF because all the attributes are prime attributes.

The Minimal Cover – More Example

- Let us consider a rela[illegible]a LOTS(PropertyID, County, Lot, Area) with the following FDS:
  - FD1: PropertyID $\rightarrow$ Lot, County, Area
  - FD2: Lot, County $\rightarrow$ Area, PropertyID
  - FD3: Area $\rightarrow$ County

The Minimal Cover – More Example

- Let us consider a relation schema LOTS(PropertyID, County, Lot, Area) with the following FDS:
  - FD1: PropertyID $\rightarrow$ Lot, County, Area
  - FD2: Lot, County $\rightarrow$ Area, PropertyID
  - FD3: Area $\rightarrow$ County

- Let us abbriviate attributes of LOTS with first letter of each attribute and represent our set of dependencies as F: $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$

- The minimal cover of a set of functional dependencies always exists but is not necessarily unique.

The Minimal Cover – More Example

- (Case X) Find a mini f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$

The Minimal Cover – More Example

- (Case X) Find a mini f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P$ $\rightarrow AP, A \rightarrow C\}$

Australian National University

The Minimal Cover – More Example

- (Case X) Find a mini… f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$

  1. **Initialise:** $\{P$ … $\rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.

The Minimal Cover – More Example

- (Case X) Find a mini[illegible]f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow$ [illegible] $\rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.

The Minimal Cover – More Example

- (Case X) Find a mini… f F = {P → LCA, LC → AP, A → C}
  1. **Initialise:** {P – … → AP, A → C}
  2. **Dependent:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  3. **Determinant:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  4. **Remove redundant FD:** P → A is redundant.

The Minimal Cover – More Example

- (Case X) Find a mini[illegible]f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow$ [illegible] $\rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  4. **Remove redundant FD:** $P \rightarrow A$ is redundant.
  5. Thus a minimal cover is $\{P \rightarrow L, P \rightarrow C, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$ or $\{P \rightarrow LC, LC \rightarrow AP, A \rightarrow C\}$.

The Minimal Cover – More Example

- (Case X) Find a mini[illegible]f F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow$ [illegible] $\rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  4. **Remove redundant FD:** $P \rightarrow A$ is redundant.
  5. Thus a minimal cover is $\{P \rightarrow L, P \rightarrow C, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$ or $\{P \rightarrow LC, LC \rightarrow AP, A \rightarrow C\}$.
- (Case Y) Find a minimal cover of F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$

The Minimal Cover – More Example

- (Case X) Find a minimal cover of F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  4. **Remove redundant FD:** $P \rightarrow A$ is redundant.
  5. Thus a minimal cover is $\{P \rightarrow L, P \rightarrow C, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$ or $\{P \rightarrow LC, LC \rightarrow AP, A \rightarrow C\}$.

- (Case Y) Find a minimal cover of F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$

The Minimal Cover – More Example

- (Case X) Find a minimal cover of F = {P → LCA, LC → AP, A → C}
  1. **Initialise:** {P → LCA, LC → AP, A → C}
  2. **Dependent:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  3. **Determinant:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  4. **Remove redundant FD:** P → A is redundant.
  5. Thus a minimal cover is {P → L, P → C, LC → A, LC → P, A → C} or {P → LC, LC → AP, A → C}.

- (Case Y) Find a minimal cover of F = {P → LCA, LC → AP, A → C}
  1. **Initialise:** {P → LCA, LC → AP, A → C}
  2. **Dependent:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.

The Minimal Cover – More Example

- (Case X) Find a minimal cover of F = {P → LCA, LC → AP, A → C}
  1. **Initialise:** {P → LCA, LC → AP, A → C}
  2. **Dependent:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  3. **Determinant:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  4. **Remove redundant FD:** P → A is redundant.
  5. Thus a minimal cover is {P → L, P → C, LC → A, LC → P, A → C} or {P → LC, LC → AP, A → C}.

- (Case Y) Find a minimal cover of F = {P → LCA, LC → AP, A → C}
  1. **Initialise:** {P → LCA, LC → AP, A → C}
  2. **Dependent:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.
  3. **Determinant:** {P → L, P → C, P → A, LC → A, LC → P, A → C}.

The Minimal Cover – More Example

- (Case X) Find a minimal cover of F = $\{P \to LCA, LC \to AP, A \to C\}$
  1. **Initialise:** $\{P \to LCA, LC \to AP, A \to C\}$
  2. **Dependent:** $\{P \to L, P \to C, P \to A, LC \to A, LC \to P, A \to C\}$.
  3. **Determinant:** $\{P \to L, P \to C, P \to A, LC \to A, LC \to P, A \to C\}$.
  4. **Remove redundant FD:** $P \to A$ is redundant.
  5. Thus a minimal cover is $\{P \to L, P \to C, LC \to A, LC \to P, A \to C\}$ or $\{P \to LC, LC \to AP, A \to C\}$.

- (Case Y) Find a minimal cover of F = $\{P \to LCA, LC \to AP, A \to C\}$
  1. **Initialise:** $\{P \to LCA, LC \to AP, A \to C\}$
  2. **Dependent:** $\{P \to L, P \to C, P \to A, LC \to A, LC \to P, A \to C\}$.
  3. **Determinant:** $\{P \to L, P \to C, P \to A, LC \to A, LC \to P, A \to C\}$.
  4. **Remove redundant FD:** $LC \to A$ and $P \to C$ are redundant.

The Minimal Cover – More Example

- (Case X) Find a minimal cover of F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  4. **Remove redundant FD:** $P \rightarrow A$ is redundant.
  5. Thus a minimal cover is $\{P \rightarrow L, P \rightarrow C, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$ or $\{P \rightarrow LC, LC \rightarrow AP, A \rightarrow C\}$.

- (Case Y) Find a minimal cover of F = $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  1. **Initialise:** $\{P \rightarrow LCA, LC \rightarrow AP, A \rightarrow C\}$
  2. **Dependent:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  3. **Determinant:** $\{P \rightarrow L, P \rightarrow C, P \rightarrow A, LC \rightarrow A, LC \rightarrow P, A \rightarrow C\}$.
  4. **Remove redundant FD:** $LC \rightarrow A$ and $P \rightarrow C$ are redundant.
  5. Thus a minimal cover is $\{P \rightarrow L, P \rightarrow A, LC \rightarrow P, A \rightarrow C\}$ or $\{P \rightarrow LA, LC \rightarrow P, A \rightarrow C\}$.

Normal Forms

- **BCNF**: Whenever a r[illegible]D $X \to A$ holds in R, then X is a **superkey** .

Normal Forms

- **BCNF**: Whenever a r[illegible]D $X \rightarrow A$ holds in R, then X is a **superkey**.

Do not represent the same fact more than once within a relation, even if some FDs have to be abandoned!

Normal Forms

- **BCNF**: Whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** .

Do not represent the same fact more than once within a relation, even if some FDs have to be abandoned!

- **3NF**: Whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute** .

Normal Forms

- **BCNF**: Whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey**.

Do not represent the same fact more than once within a relation, even if some FDs have to be abandoned!

- **3NF**: Whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey** or A is a **prime attribute**.

Do not abandon any FDs, even if some facts have to be represented more than once within a relation!

Normalisation Algorithms

BCNF-decomposition

- Repeat until no change
  - Find a problematic FD
  - Split R into two smaller ones and project FDs

Normalisation Algorithms

BCNF-decomposition

- Repeat until no change
  - Find a problematic FD
  - Split R into two smaller ones and project FDs

3NF-decomposition

- Find a minimal cover
- Group FDs in the minimal cover
- Remove redundant ones
- Add a key (if necessary)
- Project FDs

Normalisation Algorithms

BCNF-decomposition

- Repeat until no change
  - Find a problematic FD
  - Split R into two smaller ones and project FDs

3NF-decomposition

- Find a minimal cover
- Group FDs in the minimal cover
- Remove redundant ones
- Add a key (if necessary)
- Project FDs

What properties do these algorithms have?

Normalisation Algorithms

BCNF-decomposition

- Repeat until no change
  - Find a problematic FD
  - Split R into two smaller ones and project FDs

3NF-decomposition

- Find a minimal cover
- Group FDs in the minimal cover
- Remove redundant ones
- Add a key (if necessary)
- Project FDs

What properties do these algorithms have?

$\Downarrow$ Lossless join

$\Downarrow$ Lossless join + dependency preservation

Normalisation Algorithms

BCNF-decompositio

- Repeat until no char

  - Find a problematic FD

  - Split R into two smaller ones and project FDs

3NF-decomposition

- Find a minimal cover
- Group FDs in the minimal cover
- Remove redundant ones
- Add a key (if necessary)
- Project FDs

What do you need to compute using FDs?

Normalisation Algorithms

BCNF-decompositi

- Repeat until no char

  - Find a problematic FD

  - Split R into two smaller ones and project FDs

3NF-decomposition

- Find a minimal cover
- Group FDs in the minimal cover
- Remove redundant ones
- Add a key (if necessary)
- Project FDs

What do you need to compute using FDs?

$\Downarrow$

SOME superkeys (check)

$\Downarrow$

SOME superkeys (check)

ALL candidate keys

ONE minimal cover

Australian National University

Denormalisation

- **Do we need to norm**[illegible]**ion schemas in all cases** when designing a relational database[illegible]
- **Denormalisation** is [illegible]**rocess** that
  - happens after the normalisation process,
  - is often performed during the physical design stage, and
  - reduces the number of relations that need to be joined for certain queries.
- We need to distinguish:
  - **Unnormalised** – there is no systematic design.
  - **Normalised** – redundancy is reduced after a systematic design (to minimise data inconsistencies).
  - **Denormalised** – redundancy is introduced after analysing the normalised design (to improve efficiency of queries)

Australian National University

Trade-offs – Data Redundancy vs. Query Efficiency

- Normalisation: **No D[illegible]dancy but No Efficient Query Processing**
- Data redundancies a[illegible]ed in the following relations.

STUDENT		
Name	StudentID	DoB

COURSE	
CourseNo	Unit

ENROL		
StudentID	CourseNo	Semester

Trade-offs – Data Redundancy vs. Query Efficiency

- Normalisation: **No Data Redundancy but No Efficient Query Processing**
- Data redundancies a[illegible]ed in the following relations.

STUDENT		
Name	StudentID	DoB

COURSE	
CourseNo	Unit

ENROL		
StudentID	CourseNo	Semester

- However, the query for "list the names of students who enrolled in a course with 6 units" requires 2 join operations.


```
SELECT Name, CourseNo
FROM ENROL e, COURSE c, STUDENT s
WHERE e.StudentID=s.StudentID AND e.CourseNo=c.CourseNo
AND c.Unit=6;
```

Trade-offs – Data Redundancy vs. Query Efficiency

- Denormalisation: **Data Redundancy but Efficient Query Processing**
- If a student enrolled 15 courses, then the name and DoB of this student need to be stored repeatedly 15 times in ENROLMENT.

ENROLMENT					
Name	StudentID	DoB	CourseNo	Semester	Unit
Tom	123456	25/01/1988	COMP2400	2010 S2	6
Tom	123456	25/01/1988	COMP8740	2011 S2	12
Michael	123458	21/04/1985	COMP2400	2009 S2	6

Trade-offs – Data Redundancy vs. Query Efficiency

- Denormalisation: **Data Redundancy but Efficient Query Processing**
- If a student enrolled 15 courses, then the name and DoB of this student need to be stored repeatedly 15 times in ENROLMENT.

ENROLMENT					
Name	<u>StudentID</u>	DoB	<u>CourseNo</u>	<u>Semester</u>	Unit
Tom	123456	25/01/1988	COMP2400	2010 S2	6
Tom	123456	25/01/1988	COMP6740	2011 S2	12
Michael	123458	21/04/1985	COMP2400	2009 S2	6

- The query for "list the names of students who enrolled a course with 6 units" can be processed efficiently (no join needed).


```
SELECT Name, CourseNo FROM ENROLMENT WHERE Unit=6;
```

(credit cookie) Raymond F. Boyce (1947-1974)

TURED ENGLISH QUERY LANGUAGE

by

ld D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

“SEQUEL: A Structured English Query Language”,
D.D. Chamberlin and R.F. Boyce,
Proc. ACM SIGMOD Workshop on Data Description, Access and Control,
Ann Arbor, Michigan (May 1974)

http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf