Normalisation – Part 1

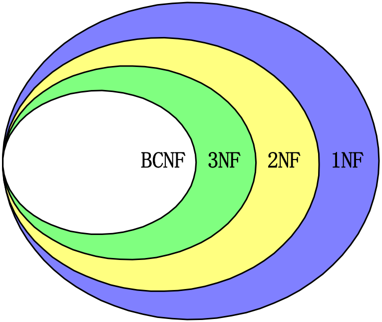
BCNF

Schema Design

- A driving force for **the study of dependencies** has been **schema design**.
- The goal of schema design is to select **the most appropriate schema** for a particular database application.
- The **choice of a schema** is guided by **semantic information** about the application data provided by users and captured by dependencies.
- A common approach starts with a **universal relation** and applies decomposition to create new relations that satisfy certain normal forms (i.e. **normalization**).

Normal Forms

Normal forms	Test criteria
1NF	weak
$\Downarrow$	
2NF	
$\Downarrow$	$\Downarrow$
3NF	
$\Downarrow$	
BCNF	strong
...	



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

Normalisation

- Decomposing a relation into **smaller relations in a certain normal form**
  - Each normal form reduces certain kind of data redundancy.
  - Each normal form does not have certain types of (undesirable) dependencies.
- What normal forms will we learn?
  1. Boyce-Codd normal form (**BCNF**)
  2. Third normal form (**3NF**)

BCNF - Definition

- A relation schema R is in **BCNF** if whenever a non-trivial FD $X \rightarrow A$ holds in R, then X is a **superkey**.
- When a relation schema is in BCNF, all data redundancy based on functional dependency are removed.
  - Note: this does not necessarily mean a good design.

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

Normalisation to BCNF

- Consider the relation schema TEACH with the following FDs:
  - $\{$StudentID, CourseName$\} \rightarrow \{$Instructor$\}$;
  - $\{$Instructor$\} \rightarrow \{$CourseName$\}$.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

- **Is TEACH in BCNF?**
  - Not in BCNF because of $\{$Instructor$\} \rightarrow \{$CourseName$\}$.

Normalisation to BCNF

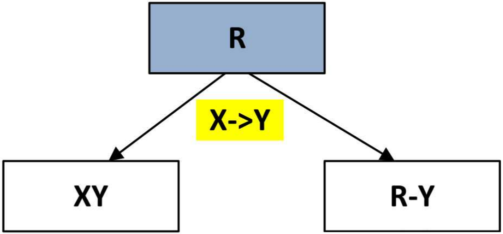
- **Algorithm** for a BCNF-decomposition

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

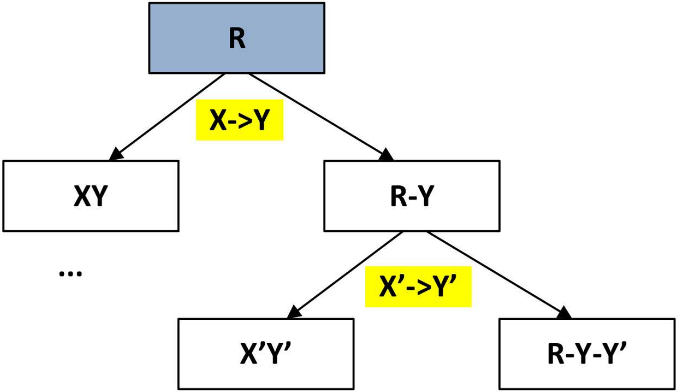
  Output: a set $\mathcal{S}$ of relation schemas in BCNF, each having a set of FDs

  - 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.

Normalisation to BCNF



Normalisation to BCNF



BCNF - Example

- Consider TEACH with the following FDs again:
  - $\{\text{StudentID}, \text{CourseName}\} \rightarrow \{\text{Instructor}\}$;
  - $\{\text{Instructor}\} \rightarrow \{\text{CourseName}\}$.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

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

BCNF - Example

- Consider TEACH with the following FDs again:
  - {StudentID,CourseName} $\rightarrow$ {Instructor};
  - **{Instructor} $\rightarrow$ {CourseName}.**

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

- **Replace TEACH with R_1 and R_2:**

R_1

CourseName	Instructor
Operating Systems	Jane
Databases	Mark

R_2

StudentID	Instructor
u123456	Jane
u234567	Jane
u234567	Mark

BCNF - Example

- Consider the relation schema TEACH with the following FDs:
  - $\{$StudentID,CourseName$\} \rightarrow \{$Instructor$\}$;
  - $\{$Instructor$\} \rightarrow \{$CourseName$\}$.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

R_1

CourseName	Instructor
Operating Systems	Jane
Databases	Mark

R_2

StudentID	Instructor
u123456	Jane
u234567	Jane
u234567	Mark

- **Does this decomposition preserve all FDs on TEACH?**

BCNF - Example

- Consider the relation schema TEACH with the following FDs:
  - **{StudentID,CourseName} → {Instructor}; Lost!**
  - {Instructor} → {CourseName}.

TEACH

StudentID	CourseName	Instructor
u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

R_1

CourseName	Instructor
Operating Systems	Jane
Databases	Mark

R_2

StudentID	Instructor
u123456	Jane
u234567	Jane
u234567	Mark

- **No. We only have {Instructor} → {CourseName} on R_1.**

Two Properties

- 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.

Normalisation – Part 2

3NF

From BCNF to 3NF

- **Facts**

  (1) There exists an algorithm that can generate **a lossless decomposition into BCNF**.

  (2) However, a BCNF-decomposition that is **both lossless and dependency-preserving** does not always exist.

- 3NF is **a less restrictive normal form** such that a lossless and dependency preserving decomposition can always be found.

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**.

- 3NF allows data redundancy but excludes relation schemas with certain kinds of FDs (i.e., partial FDs and transitive FDs).

Normalisation to 3NF

- Consider the following FDs of 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.
  - ENROL is not in 3NF because {ConfirmedBy_ID} → {StaffName}, {ConfirmedBy_ID} is not a superkey and {StaffName} is not 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

  - 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

$R_1 = X_1A_1 \ldots A_K$ ... $R_n = X_nA$

$X_1 \rightarrow A_1$

...

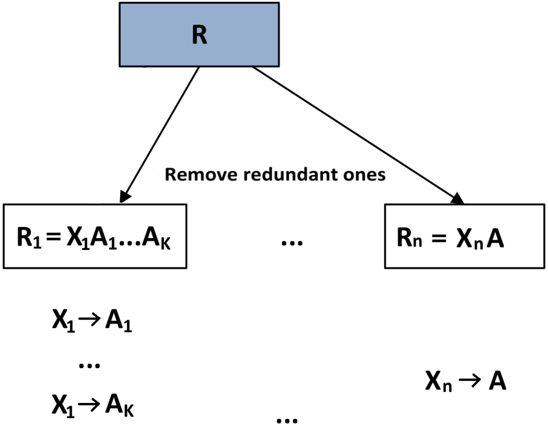
$X_1 \rightarrow A_K$

A minimal cover

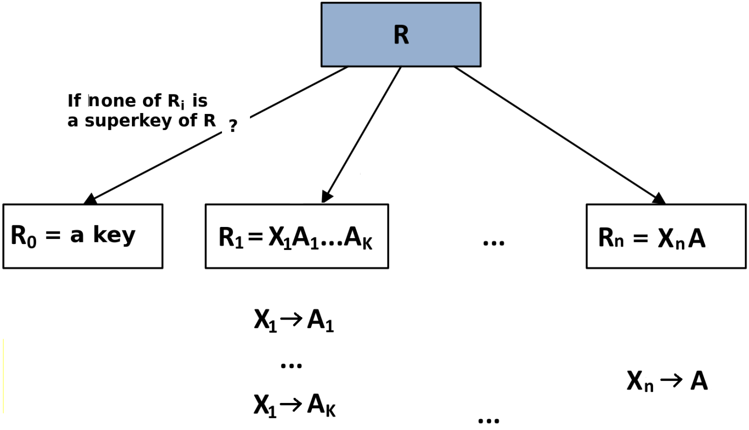
...

$X_n \rightarrow A$

Normalisation to 3NF



Normalisation to 3NF



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

- **Theorem:**

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

- **Examples:** Consider the following set of functional dependencies:

$$\Sigma = \{A \rightarrow BC, B \rightarrow C, B \rightarrow A, C \rightarrow AB\}$$

Σ has two different minimal covers:

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

Minimal Cover - Examples

- The set $\{A \to B, B \to C, A \to C\}$ can be reduced to $\{A \to B, B \to C\}$, because $\{A \to C\}$ is implied by the other two.
- Given the set of FDs $\Sigma = \{B \to A, D \to A, AB \to D\}$, 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 (look good);
  3. determine if $AB \to D$ has any redundant attribute on the left hand side ($AB \to D$ can be replaced by $B \to D$);
  4. look for a redundant FD in $\{B \to A, D \to A, B \to D\}$ ($B \to A$ is redundant);

  Therefore, the minimal cover of Σ is $\{D \to A, B \to D\}$.

Normalisation to 3NF – Example

- Consider ENROL again:
  - {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} $\rightarrow$ {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 again:
  - {StudentID, CourseNo, Semester} $\rightarrow$ {ConfirmedBy_ID, StaffName}
  - {ConfirmedBy_ID} $\rightarrow$ {StaffName}

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}
  - R_2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} $\rightarrow$ {StaffName}
  - Omit R_0 because R_1 is a superkey of ENROL.

3NF - Exercises

- Let us do some exercises for the 3NF-decomposition algorithm.
  - **Exercise 1:** $R = \{A, B, C, D\}$ and $\Sigma = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$:
  - **Exercise 2:** $R = \{A, B, C, D\}$ and $\Sigma = \{AD \rightarrow B,\ AB \rightarrow C,\ C \rightarrow B\}$:

3NF - Exercises

- Let us do some exercises for the 3NF-decomposition algorithm.
  - **Exercise 1:** $R = \{A, B, C, D\}$ and $\Sigma = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$:
    - $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ is a minimal cover.
    - $R_1 = ABD$, $R_2 = BC$ (omit R_0 because R_1 is a superkey of R)
    - The 3NF-decomposition is $\{$ABD, BC$\}$.
  - **Exercise 2:** $R = \{A, B, C, D\}$ and $\Sigma = \{AD \rightarrow B,\ AB \rightarrow C,\ C \rightarrow B\}$:
    - Σ is its own minimal cover.
    - $R_1 = ABD$, $R_2 = ABC$, $R_3 = CB$ (omit R_3 because $R_3 \subseteq R_2$ and omit R_0 because R_1 is a superkey of R)
    - The 3NF-decomposition is $\{$ABD, ABC$\}$.

Two Properties

- **Facts**

  (1) There exists an algorithm that can generate **a lossless decomposition into BCNF**.

  (2) However, a BCNF-decomposition that is **both lossless and dependency-preserving** does not always exist.

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

Normalisation – Part 3

Summary and Discussion

Summary of Normal Forms

- 1NF, 3NF and BCNF are popular in practice. Other normal forms are rarely used.

  1NF: only atomic values for attributes (*part of the definition for the relational data model*);

  2NF: an intermediate result in the history of database design theory;

  3NF: lossless and dependencies can be preserved;

  BCNF: lossless but dependencies may not be preserved.

- 3NF can only **minimise (not necessarily eliminate) redundancy**. So a relation schema in 3NF may still have update anomalies.
- A relation schema in BCNF **eliminates redundancy**.

Why Denormalisation?

- **Do we need to normalize relation schemas in all cases** when designing a relational database?
- The normalisation process **may degrade performance** when data are frequently queried.
- Since relation schemas are decomposed into many smaller ones after normalisation, queries need to **join many relations together** in order to return the results.
- Unfortunately, **join operation is very expensive**.
- When data is **more frequently queried rather than being updated** (e.g., data warehousing system), a weaker normal form is desired (i.e., **denormalisation**).

Denormalisation

- **Denormalisation** is a **design process** 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)

Trade-offs

- A good database design is to **find a balance** between desired properties, then normalise/denormalise relations to a desired degree.

Trade-offs – Data Redundancy vs. Query Efficiency

- Normalisation: **No Data Redundancy but No Efficient Query Processing**
- Data redundancies are eliminated in the following relations.

STUDENT		
Name	StudentID	DoB
Tom	123456	25/01/1988
Michael	123458	21/04/1985

COURSE	
CourseNo	Unit
COMP2400	6
COMP8740	12

ENROL		
StudentID	CourseNo	Semester
123456	COMP2400	2010 S2
123456	COMP8740	2011 S2
123458	COMP2400	2009 S2

- 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

- However, 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;
```

Discussion

- Both normalisation and denormalisation are useful in database design.
  - **Normalisation**: obtain database schema avoiding redundancies and data inconsistencies
  - **Denormalisation**: join normalized relation schemata for the sake of better query processing
- Some problems of (de-)normalisation:
  - FDs **cannot handle null values**.
  - To apply normalisation, FDs must be **fully specified**.
  - The algorithms for normalisation **are not deterministic**, leading to different decompositions.