Normalisation

Schema Design

A driving force for the study of dependencies has been schema design.
Schema design aims 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).

数据库模式设计是数据库设计中的一个关键步骤，它的目标是为特定的数据库应用程序选择最合适的模式或结构。
模式的选择受到来自应用程序数据的语义信息的指导，这些信息由用户提供并由依赖关系所捕获。这意味着我们需要了解数据库中数据的含义和关系，以便设计出能够满足应用程序需求的模式。
常见的数据库模式设计方法通常从一个通用关系（universal relation）开始，然后应用分解（decomposition）的过程，创建新的关系，以满足某些正规化形式（例如，范式化）。正规化的目标是通过将数据拆分成更小、更规范化的部分，以减少数据冗余、提高数据的一致性和完整性，并提高查询性能。
因此，数据库模式设计是一个旨在根据应用程序需求创建有效、一致和可维护的数据库结构的过程，依赖关系是帮助指导这一过程的关键工具。理解数据的语义以及如何将数据分解成适当的关系是设计成功的关键。

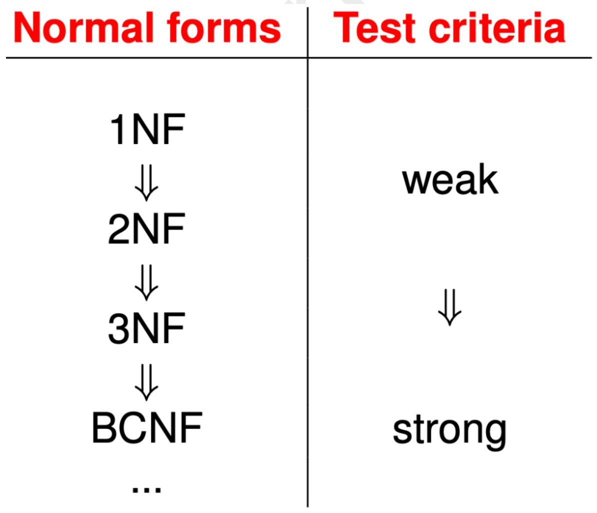
Normal Forms

Normalisation is decomposing a relation into smaller relations in a certain normal form.
Each normal form reduces certain kinds of data redundancy.

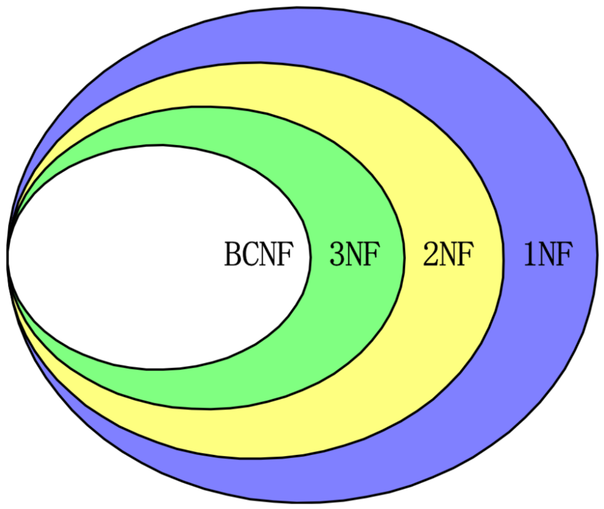
Each normal form does not have certain types of (undesirable) dependencies.

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).

What normal forms will we learn?
Boyce-Codd normal form (BCNF)
Third normal form (3NF)



Normal forms	Test criteria
1NF	weak
⇓	
2NF	
⇓	⇓
3NF	
⇓	
BCNF	strong
...	



数据库正规化的概念涉及到将数据库中的数据组织成不同的表，以便减少数据冗余、提高数据的

一致性和完整性，并减少插入、更新和删除操作中的异常情况。正规化通过将数据拆分成更小、更规范化的部分来实现这些目标。不同的正规化级别由不同的规则和约束来定义。
以下是一些常见的数据库正规化级别以及它们基于的约束：

1. 第一范式（1NF）：第一范式不是基于特定的约束，而是指确保每个表中的每个列都包含原子值，即不包含多个值或复杂的数据结构。
2. 第二范式（2NF）：第二范式是基于关键属性和函数依赖的约束。它要求在表中的每个非关键属性都完全函数依赖于关键属性。这意味着每个非关键属性必须依赖于表中的整个关键，而不是只依赖于部分关键。
3. 第三范式（3NF）：第三范式也是基于关键属性和函数依赖的约束。它要求表中的每个非关键属性都不传递依赖于关键属性。这意味着非关键属性不能依赖于其他非关键属性。
4. 巴斯－科德范式（BCNF）：BCNF同样基于关键属性和函数依赖的约束。它要求在表中的每个非关键属性都完全函数依赖于关键属性，而且对于每个非平凡的函数依赖X → Y，X必须是一个超键。
5. 其他范式（4NF和5NF）：除了上述范式外，还存在其他更高级别的正规化范式，如第四范式（4NF）和第五范式（5NF），它们通常基于其他类型的约束，如多值依赖和连接依赖。这些范式通常在高度复杂的数据库设计中使用，而不是常见的应用程序。

总之，正规化是一种用于设计数据库模式以提高数据质量和性能的方法，不同的正规化级别基于不同的约束来定义。选择适当的正规化级别取决于特定应用程序的需求和数据特性。

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.

Lossless Join – Example

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

Example 1

R1	
Name	StudentID
Mike	123456
Mike	123458

R2	
StudentID	DoB
123456	20/09/1989
123458	25/01/1988

Does the decomposition of R into R1 and R2 has the lossless join property?
Yes, because the natural join of R1 and R2 yields R.

Example 2

The following decomposition from R into R3 and R4 doesn't have the lossless join property. It generates spurious tuples.

R3	
Name	StudentID
Mike	123456
Mike	123458

R4	
Name	DoB
Mike	20/09/1989
Mike	25/01/1988

SELECT * FROM R3 NATURAL JOIN R4		
Name	StudentID	DoB
Mike	123456	20/09/1989
Mike	123456	25/01/1988
Mike	123458	20/09/1989
Mike	123458	25/01/1988

Note: Natural JOIN rely on common attributes if no common parts, it will do cartesian product.

Dependency Preservation – Example

Example 1: Given a FD {StudentID} → {Name} defined on R

R1	
Name	StudentID
Mike	123456
Mike	123458

R2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

Does the above decomposition preserves {StudentID} → {Name}?
Yes, because {StudentID} and {Name} are both in R1 after decomposition and thus {StudentID} → {Name} is preserved in R1.

Example 2

R1	
Name	CourseNo
Mike	COMP2400
Mike	COMP2600

R2	
StudentID	CourseNo
123456	COMP2400
123458	COMP2600

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

Example 3: Given a set of FDs { {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

R1	
Name	Email
Mike	123456@anu.edu.au
Tom	123123@anu.edu.au

R2	
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 R2) and {Email} → {Name} (preserved in R1).

Discussion

If R with a set Σ of FDs is decomposed into R1 with Σ1 and R2 with Σ2,

Lossless join if and only if the common attributes of R1 and R2 are a superkey for R1 or R2;
Dependency preserving if and only if $(\Sigma 1 \cup \Sigma 2)^* = \Sigma^*$ holds.

Consider R={A, B, C} with the set of FDs Σ = {A → B, B → C, A → C}.

Does the decomposition of R into R1 = {A, B} and R2 = {A, C} fulfil lossless join and dependency preserving?
Σ1 ={A→B}andΣ2 ={A→C}
Lossless join? Yes because A is a superkey for R1.
Dependency preserving? No because $(\Sigma 1 \cup \Sigma 2)^* \neq \Sigma^*$ from the fact that {A → B, A → C} $\nvDash$ B → C.

Does the decomposition of R into R1 = {A, B} and R3 = {B, C} fulfil lossless join and dependency preserving?
Σ1 ={A→B}andΣ3 ={B→C}
Lossless join? Yes because B is a superkey for R3.
Dependency preserving? Yes because $(\Sigma 1 \cup \Sigma 3)^* = \Sigma^*$ from the fact that {A → B, B → C} $\vDash$ A → C.

BCNF

Definition

A relation schema R is in BCNF whenever a non-trivial FD X → A holds in R, then X is a superkey.

When a relation schema is in BCNF, all data redundancy based on functional dependency is removed. The aim is to not represent the same fact twice (within a relation)!
Note: this does not necessarily mean a good design.

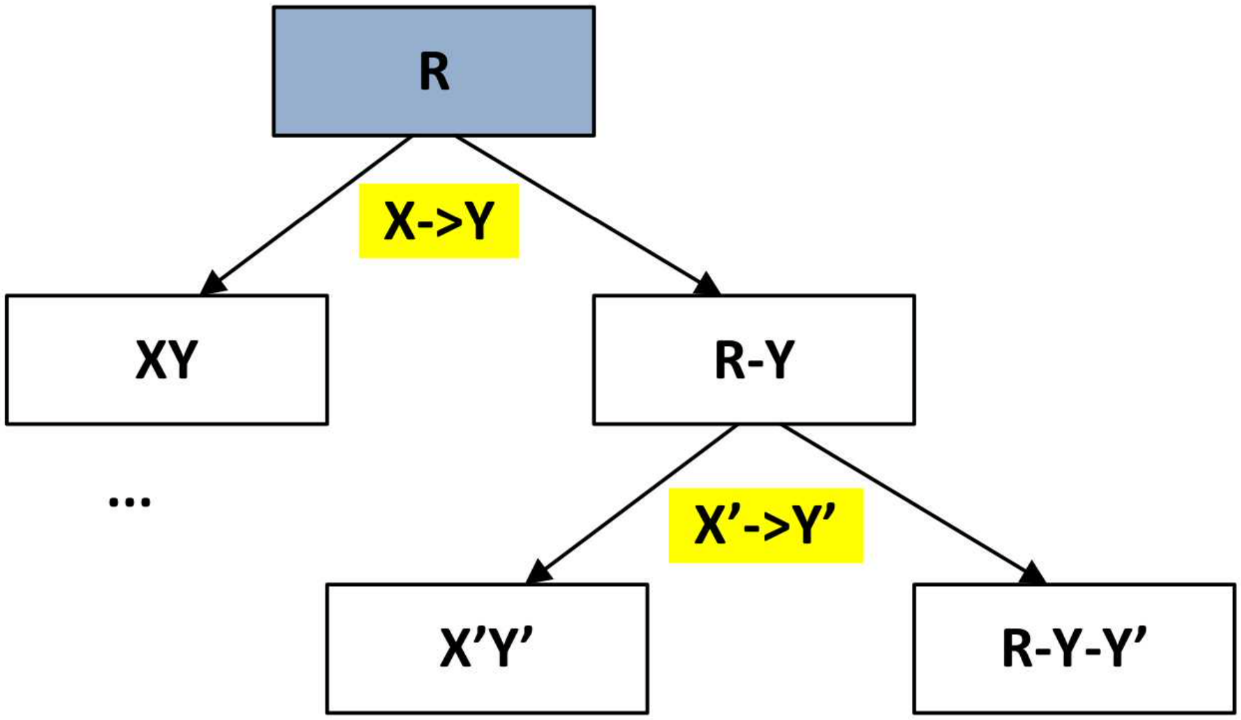
Normalisation to BCNF

Consider the relation schema TEACH with the following FDs:
{StudentID, CourseName} → {Instructor};
{Instructor} → {CourseName}.
Is TEACH in BCNF?
Not in BCNF because of {Instructor} → {CourseName}.

Algorithm for a BCNF-decomposition
Input: a relation schema R′ and a set Σ of FDs on R′.
Output: a set S of relation schemas in BCNF, each having a set of FDs

Start with S = {R′};
Do the following for each R ∈ S iteratively until no changes on S:

1. Find a (non-trivial) FD X → Y on R that violates BCNF, if any;
2. Replace R in S with two relation schemas XY and (R − Y) and project the FDs to these two relation schemas.



BCNF - Example

Consider TEACH with the following FDs again:
{StudentID, CourseName} → {Instructor};
{Instructor} → {CourseName}.
Can we normalise TEACH into BCNF?

StudentID	CourseName	Instructor

u123456	Operating Systems	Jane
u234567	Operating Systems	Jane
u234567	Databases	Mark

Replace TEACH with R1 and R2:

R1			R2	
CourseName	Instructor		StudentID	Instructor
Operating Systems	Jane		u123456	Jane
Databases	Mark		u234567	Jane
			u234567	Mark

Does this decomposition preserve all FDs on TEACH?
No. We only have {Instructor} → {CourseName} on R1.
{StudentID,CourseName} → {Instructor}; Lost!

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}

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.

INTERVIEW is not in BCNF because {OfficerID, Date} → {Room} and {OfficerID, Date} are not superkey.

We decompose INTERVIEW along 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

Project FDs on two new relation schemas.
INTERVIEW1: {OfficerID, Date} → {Room}
INTERVIEW2: {CustomerID, Date} → {OfficerID, Time}, {OfficerID, Date, Time} → {CustomerID}.

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

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 → B,C → B,B → C}.
Case 1: (Using C → B first)
R1 ={B,C},Σ1 ={B→C,C→B};R2 ={A,C},Σ2 ={A→C}
Case 2: (Using B → C first)
R1′ = {B,C},Σ′1 = {B → C,C → B};R2′ = {A,B},Σ′2 = {A → B};

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?

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 → 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).

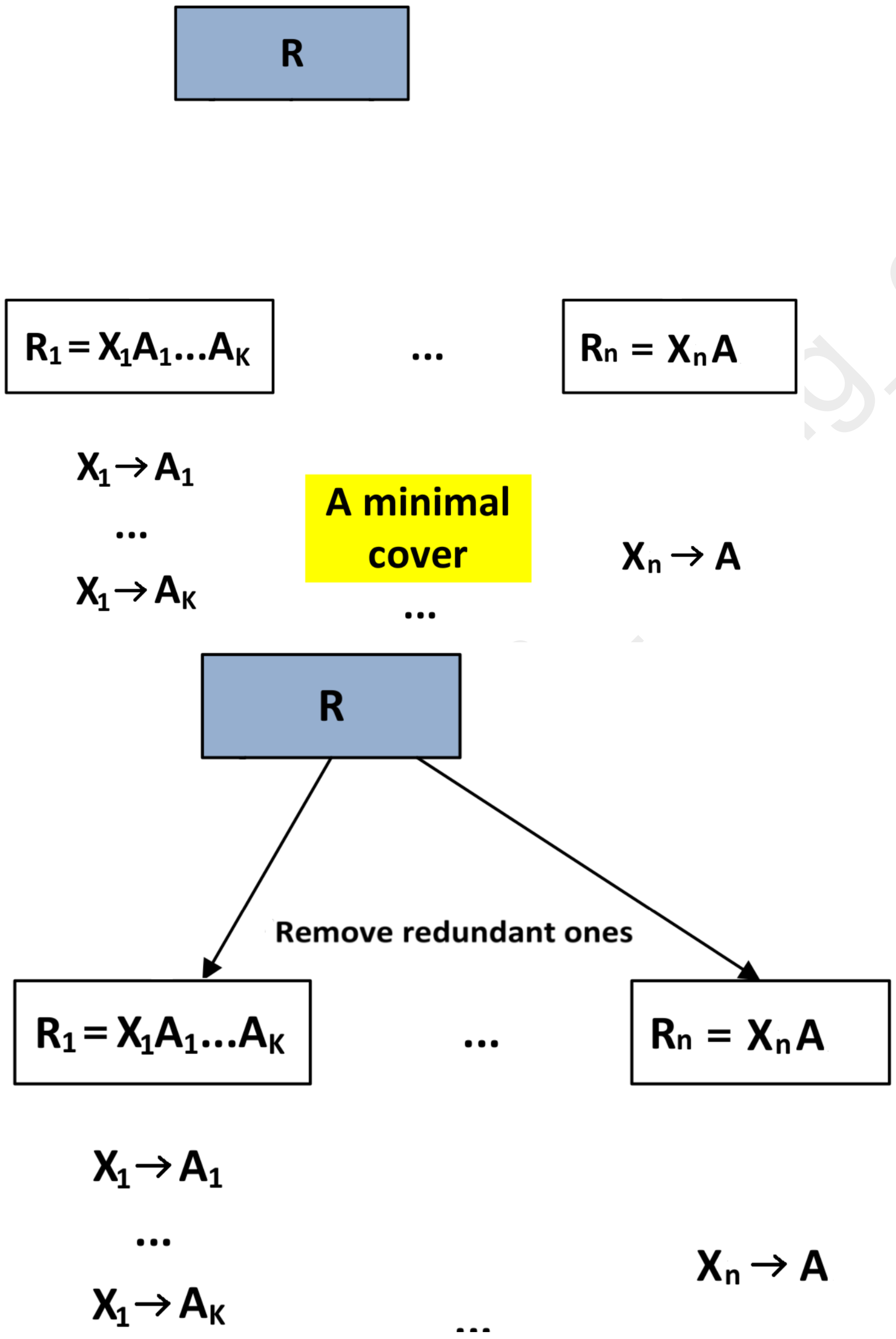
Normalisation to 3NF

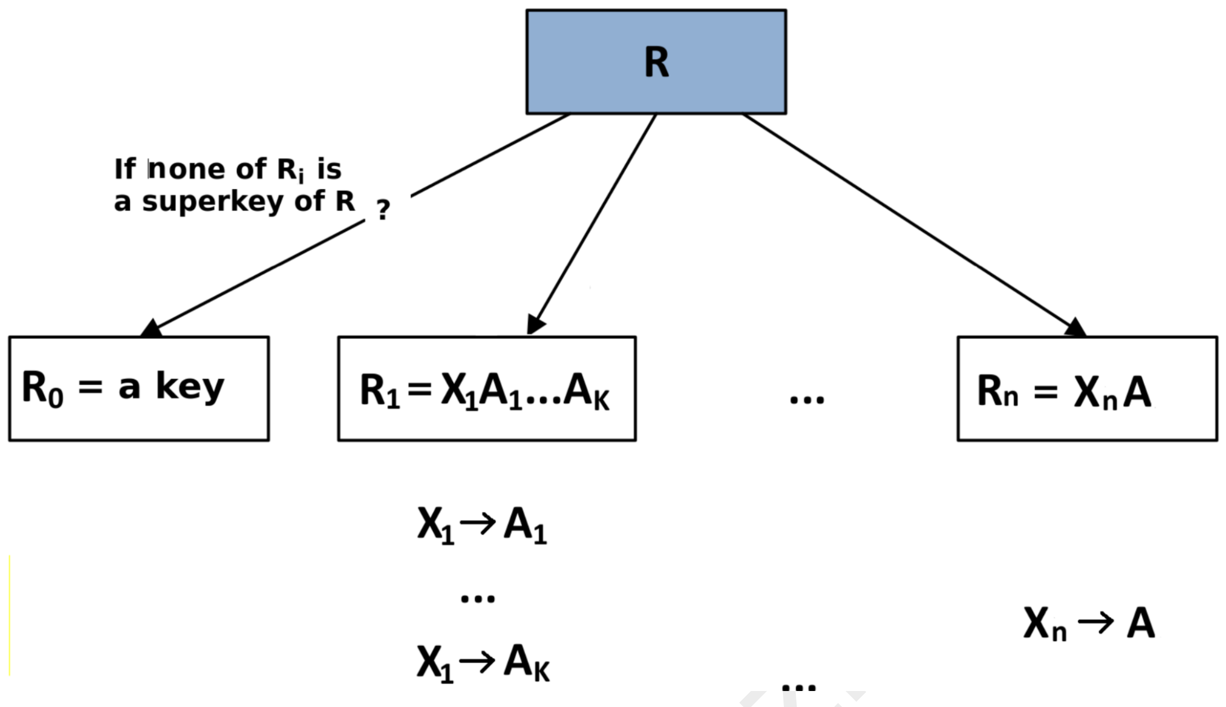
Consider the following FDs of ENROL:
{StudentID, CourseNo, Semester} → {ConfirmedBy_ID, StaffName};
{ConfirmedBy_ID} → {StaffName}.

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 a prime attribute.

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 S of relation schemas in 3NF, each having a set of FDs

1. Compute a minimal cover Σ′ for Σ and start with S =ϕ
2. Group FDs in Σ′ by their left-hand-side attribute sets
3. For each distinct left-hand-side Xi of FDs in Σ′ that includes Xi → A1,Xi → A2,...,Xi → Ak:
   Add Ri =Xi ∪{A1}∪{A2}⋯∪{Ak} to S
4. Remove all redundant ones from S (i.e., remove Ri if Ri ⊆ Rj)
5. If S does not contain a superkey of R, add a key of R as R0 into S.
6. Project the FDs in Σ′ onto each relation schema in S


R
$R_1 = X_1A_1…A_K$
…
$R_n = X_nA$
$X_1 \to A_1$
…
$X_1 \to A_K$
A minimal cover
$X_n \to A$
…
R
Remove redundant ones
$R_1 = X_1A_1…A_K$
…
$R_n = X_nA$
$X_1 \to A_1$
…
$X_1 \to A_K$
$X_n \to A$
…



Minimal Cover - Recap

Example 1: Σ1 ={X →Y,Y →Z,X →Z} and Σ2 ={X →Y,Y →Z}
If $\Sigma1^* = \Sigma2^*$, then Σ1 is not minimal

Example 2: Σ1 ={X →Y,XY →Z} and Σ2 ={X →Y,X →Z}
If $\Sigma1^* = \Sigma2^*$, then Σ1 is not minimal

The set {A → B, B → C, A → C} can be reduced to {A → B, B → C}, because {A → C} is implied by the other two.

Given the set of FDs Σ
{StudentID, CourseNo, Semester} → {ConfirmedBy ID, StaffName}
{ConfirmedBy ID} → {StaffName}
we can compute the minimal cover of Σ as follows:
1 start from Σ;
2 checks whether all the FDs in Σ have only one attribute on the right-hand side;
    {StudentID, CourseNo, Semester} → { ConfirmedBy_ID, StaffName } can be replaced by
    {StudentID, CourseNo, Semester} → { ConfirmedBy_ID }
    {StudentID, CourseNo, Semester} → { StaffName }
3 checks whether all the FDs in Σ have a 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

All look good!

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

5 Therefore, the minial cover of Σ is { {StudentID, CourseNo, Semester} → {ConfirmedBy ID}, {ConfirmedBy ID} → {StaffName}}

3NF – Example

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

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

Hence, we have:
R1={StudentID, CourseNo, Semester, ConfirmedBy_ID} with {StudentID, CourseNo, Semester} → {ConfirmedBy_ID}
R2={ConfirmedBy_ID, StaffName} with {ConfirmedBy_ID} → {StaffName}
Omit R0 because R1 is a superkey of ENROL.

Let us do some exercises for the 3NF-decomposition algorithm.

Exercise 1: R = {A, B, C, D} and Σ = {A → B, B → C, AC → D}:

{A → B, B → C, A → D} is a minimal cover.
R1 = ABD, R2 = BC (omit R0 because R1 is a superkey of R)
The 3NF-decomposition is {ABD, BC}.

Exercise2:
R={A,B,C,D}andΣ={AD→B, AB→C, C→B}: Σ is its own minimal cover.
R1 =ABD,R2 =ABC,R3 =CB (omit R3 because R3 ⊆R2 and omit R0 because R1 is a superkey of R)
The 3NF-decomposition is {ABD, ABC}.

Summary

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.

1. 第一范式（1NF）：1NF要求表中的每个列包含原子值，即不包含多个值或复杂的数据结构。这是关系数据模型的基本要求。
2. 第二范式（2NF）：2NF是数据库设计理论发展历史中的一个中间结果，它要求每个非关键属性完全函数依赖于关键属性。它有时在设计中用于消除部分依赖。
3. 第三范式（3NF）：3NF是一种流行的正规化级别，要求表中的每个非关键属性不传递依赖于关键属性。它能够保持数据的无损性和依赖关系，但仍可能存在冗余。
4. 巴斯－科德范式（BCNF）：BCNF是另一个常见的正规化级别，要求每个非关键属性完全函数依赖于关键属性，并且对于每个非平凡的函数依赖X → Y，X必须是一个超键。BCNF能够保持数据的无损性，但不一定能够保留所有依赖关系。

3NF和BCNF的比较：3NF只能最小化（但不一定消除）冗余，因此在某些情况下可能仍然存在更新异常。BCNF能够消除冗余，这意味着每个属性都只取决于关键属性，没有多余的依赖。
总之，1NF是关系数据模型的基本要求，而3NF和BCNF是在实际数据库设计中常见的正规化级别。它们的选择取决于特定应用程序的需求和数据特性。虽然还有其他正规化级别，但它们在实践中较少使用。正规化有助于提高数据库的数据质量、一致性和性能，并减少数据异常。

Normalisation Algorithms

BCNF-decomposition

- Repeat until no changes
  - 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?

⇓	⇓
Lossless join	Lossless join + dependency preservation

What do you need to compute using FDs?

⇓
SOME superkeys (check)

⇓
SOME superkeys (check)
ALL candidate keys
ONE minimal cover

Denormalisation

Why？

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).

反规范化（Denormalization）是一种在数据库设计中使用的技术，用于提高查询性能，尽管它在一定程度上牺牲了数据的一致性和冗余。
为什么需要反规范化？

- 性能优化：在某些情况下，标准的数据库正规化可能导致性能下降。这通常发生在需要频繁查询数据的情况下，因为查询需要合并多个关系，并执行昂贵的连接操作。
- 减少连接操作：标准正规化后，数据被拆分成多个小的关系，这意味着查询需要执行许多连接操作才能返回结果。连接操作在性能上可能非常昂贵。
- 数据仓库系统：在数据仓库系统等场景中，数据通常更频繁地查询而不是更新。在这种情况下，更弱的正规化形式（即反规范化）可能是更合适的选择，因为它可以提高查询性能。

虽然反规范化可以提高查询性能，但也要注意它可能导致数据冗余和一致性问题。因此，反规范化应该在权衡性能需求和数据一致性之间做出明智的选择。在需要频繁更新的情况下，可能需要更多的规范化。在需要频繁查询而数据变化较少的情况下，反规范化可以是一个有用的选项。设计数据库时，需要根据具体的应用程序需求和查询模式来考虑是否进行反规范化。

定义

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 the efficiency of queries)

反规范化（Denormalization）是数据库设计的一个过程，通常在正规化（Normalization）过程之后，特别是在物理设计阶段进行。它的主要目的是为了提高某些查询的性能，尽管这可能引入数据冗余。

反规范化的要点包括：

- 发生时间：反规范化通常发生在正规化之后，作为性能优化的一部分。正规化用于减少数据的冗余和提高数据的一致性，而反规范化则用于改善查询性能。
- 物理设计阶段：反规范化通常在数据库的物理设计阶段进行，这是数据库设计的最后阶段。在这个阶段，考虑到数据库引擎和查询优化的特定需求，可以决定如何重新组织数据以提高查询性能。
- 减少关系数目：反规范化的主要目标之一是减少需要进行连接的关系数目。这可以通过将数据冗余引入数据库设计中来实现，以便某些查询可以更高效地执行，而不必执行大量的连接操作。

需要区分以下三种情况：

非规范化（Unnormalised）：在这种情况下，数据库没有经过系统化的设计过程，数据可能包含冗余并且不一致。

规范化（Normalized）：在规范化后，数据经过系统设计，冗余被最小化以减少数据不一致性。

反规范化（Denormalized）：在规范化设计的基础上，反规范化引入了一些数据冗余，以提高某些查询的性能。这是在性能需求明确的情况下的权衡决策。

Trade-offs – Data Redundancy vs. Query Efficiency

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

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;

Denormalisation: Data Redundancy but Efficient Query Processing
If a student enrolled in 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.

正规化和反规范化都在数据库设计中发挥着重要作用，它们各自解决了不同的需求和问题：
正规化：

- 正规化的主要目标是设计数据库模式，以避免数据中的冗余和减少数据不一致性。通过将数据分解成更小、更规范化的关系，正规化有助于维护数据的一致性和完整性。
- 正规化使用函数依赖（FDs）等规则来确保数据库的逻辑结构是精确、准确的，从而减少了数据更新时可能出现的异常情况。

反规范化：

- 反规范化旨在通过合并正规化的关系模式来提高查询性能。它通过引入一些数据冗余来减少需要进行连接的关系数量，从而加快查询处理。

- 反规范化适用于那些需要频繁查询而数据变化较少的情况，因为它可以改善查询性能，减少连接操作。

然而，正规化和反规范化都有一些问题和限制：

- FDs无法处理空值：函数依赖不适用于处理包含空值的情况，因为它们基于属性之间的确定性关系。
- 完全规定的FDs：为了应用正规化，函数依赖必须得到完全规定，这可能需要在设计过程中进行一些额外的工作。
- 非确定性的分解算法：正规化的算法通常是非确定性的，这意味着可能存在多种不同的分解方式，这可能导致不同的结果。