Week10_Course

Database System-Schema+Design part4

Schema Design and Refinement

- How do we obtain a good design?
- Functional dependencies & keys
- Desirable properties of schema refinement
- Boyce Codd Normal Form (BCNF)
- Third Normal Form (3NF)
- Fourth Normal Form (4NF)

Persons with several phones:

Name	SSN	Phone
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

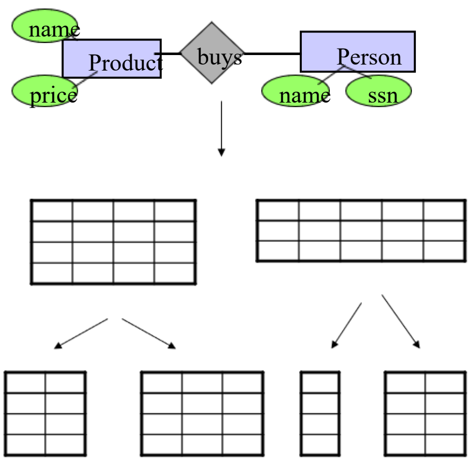
SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN	Phone
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

（SSN,Name） （Name,Phone）？

Relational Schema Design (or Logical Design)

Conceptual Model:



Relational Model:
- create tables
- specify FD's
- find keys

Normalization
- use FDs to decompose tables to achieve better design

Database System - Nankai

Desirable Properties of Schema Refinement

1) minimize redundancy

2) avoid info loss

3) preserve dependency

4) ensure good query performance

Database System - Nankai

Recall: Relation Decomposition

The original relation schema

Name	SSN	Phone
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

Break the relation into two:

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN	Phone
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

Desirable Property #1: Minimize redundancy

Database System - Nankai

Decompositions in General

Let R be a relation with attributes $A_1, A_2, \ldots A_n$

Create two relations R_1 and R_2 with attributes

$B_1, B_2, \ldots B_m$ $\qquad$ $C_1, C_2, \ldots C_t$

Such that:

$$B_1, B_2, \ldots B_m \cup C_1, C_2, \ldots C_t = A_1, A_2, \ldots A_n$$

And

-- R_1 is the projection of R on $B_1, B_2, \ldots B_m$
-- R_2 is the projection of R on $C_1, C_2, \ldots C_t$

Database System - Nankai

Certain Decomposition May Cause Problems

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
DoubleClick	29.99	Camera

Decompose on : **Name, Category** and **Price, Category**

Name	Category
Gizmo	Gadget
OneClick	Camera
DoubleClick	Camera

Price	Category
19.99	Gadget
24.99	Camera
29.99	Camera

When we put it back:

Cannot recover information

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
OneClick	**29.99**	**Camera**
DoubleClick	**24.99**	**Camera**
DoubleClick	29.99	Camera

Lossless Decomposition（无损分解）

A decomposition is *lossless* if we can recover:

R(A,B,C)

Decompose

R1(A,B)   R2(A,C)

Recover

R’(A,B,C)   should be the same as R(A,B,C)

R’ is in general larger than R. Must ensure R’ = R

Desirable Property #2: Lossless decomposition

Put Another Way: "Lossless" Joins

- The main idea: if you decompose a relation schema, then join the parts of an instance via a natural join, you might get more rows than you started with, i.e., spurious tuples
  - This is bad!
  - Called a "lossy join".
- Goal: decompositions which produce only "lossless" joins
  - "non-additive" join is more descriptive

Dependency Preserving

（保持函数依赖）

- Given a relation R and a set of FDs S
- Suppose we decompose R into R1 and R2
- Suppose
  - R1 has a set of FDs S1
  - R2 has a set of FDs S2
  - S1 and S2 are computed from S
- We say the decomposition is dependency preserving if by enforcing S1 over R1 and S2 over R2, we can enforce S over R
- $(S1 \cup S2)^+ = S^+$

An Example

Unit	Company	Product

FD's: Unit → Company; Company, Product → Unit

So, there is a BCNF violation, and we decompose.

Unit	Company

Unit → Company

Unit	Product

No FDs

Database System - Nankai

So What's the Problem?

Unit	Company
Galaga99	UI
Bingo	UI

Unit	Product
Galaga99	databases
Bingo	databases

No problem so far. All *local* FD's are satisfied.

Let's put all the data back into a single table again:

Unit	Company	Product
Galaga99	UI	databases
Bingo	UI	databases

Violates the dependency: company, product -> unit!

Database System - Nankai

Preserving FDs

- What if, when a relation is decomposed, the X of an $X \rightarrow Y$ ends up only in one of the new relations and the Y ends up only in another?
- Such a decomposition is not “dependency-preserving.”
- Desirable Property #3: always have FD-preserving decompositions
- We will talk about "Desirable Property #4: Ensure Good Query Performance" later

Review

- When decomposing a relation R, we want to decomposition to
  - minimize redundancy
  - avoid info loss
  - preserve dependencies (i.e., constraints)
  - ensure good query performance
- These objectives can be conflicting
- Various normal forms achieve parts of the objectives

多选题 1分

互动交流一

已知关系模式：学生（身份证号，学号，姓名

将其分解为两个关系模式：学生身份（身份证号

学生信息（学号，姓

下面哪个描述是正确的？（多选题）

A 这个分解是无损分解

B 这个分解不是无损分解

C 这是保持无损连接的分解

D 这不是保持无损连接的分解

提交

Database System - Nankai

单选题 1分

互动交流二

已知关系模式：学生（身份证号，学号，姓名

将其分解为两个关系模式：学生身份（身份证号

学生信息（学号，姓

下面哪个描述是正确的？

A 这个分解是保持函数依赖的分解

B 这个分解是丢失函数依赖的分解

提交

m - Nankai

多选题 1分

互动交流三

已知关系模式：学生（身份证号，学号，姓名，
将其分解为两个关系模式：学生身份（身份证号
学生信息（学号，

下面哪个描述是正确的？ （多选题）

A 这个分解是无损分解

B 这个分解不是无损分解

C 这是保持无损连接的分解

D 这不是保持无损连接的分解

提交

Database System - Nankai

多选题 1分

互动交流四

已知关系模式：学生（身份证号，学号，姓名，
将其分解为两个关系模式：学生身份（身份证号
学生信息（学号，

下面哪个描述是正确的？

A 这个分解是保持函数依赖的分解

B 这个分解是丢失函数依赖的分解

提交

Database System - Nankai

互动交流五

Consider a relation $R = (A, B, C, D, E)$ with FD' s $A \rightarrow C$, $CD \rightarrow B$, $B \rightarrow E$ and $E \rightarrow D$.

We decompose R into $R_1(B, C, D)$ and $R_2(A, C, E)$.

What are the FD' s that hold in R_1 and R_2?

What are the keys of R_1 and R_2?

请用弹幕回答

Database System - Nankai

Week10_Course

Database System-Schema+Design part5

Database System - Nankai

Schema Design and Refinement

- How do we obtain a good design?
- Functional dependencies & keys
- Desirable properties of schema refinement
- Boyce Codd Normal Form (BCNF)
- Third Normal Form (3NF) and 3NF Decomposition
- Fourth Normal Form (4NF)

Normal Forms

First Normal Form = all attributes are atomic
Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)
Third Normal Form (3NF)
Fourth Normal Form (4NF)
Others...

$R \in 4NF \gg R \in BCNF \gg R \in 3NF \gg R \in 2NF \gg R \in 1NF$

What We Want to Do with Normal Forms

- Take a relation schema…
- Test it against a normalization criterion…
- If it passes, fine!
  - Maybe test again with a higher criterion
- If it fails, decompose into smaller relations
  - Each of them will pass the test
  - Each can then be tested with a higher criterion

Persons with several phones:

Name	SSN	Phone
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN	Phone
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

Database System - Nankai

Boyce-Codd Normal Form

A relation R is in BCNF if and only if:

Whenever there is a nontrivial FD $A_1, A_2, \ldots A_n \rightarrow B$

for R , it is the case that $\{A_1, A_2, \ldots A_n\}$ is a super-key for R.

Remember: *nontrivial* means A is not a member of set X. $(X \rightarrow A)$

Remember, a *superkey* is any superset of a key (not necessarily a proper superset).

In English (though a bit vague):

Whenever a set of attributes of R is determining another attribute, it should determine all attributes of R.

Database System - Nankai

Example

Name	SSN	Phone
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

Person(Name, SSN, Phone)

FDs: {SSN → Name}

What are the dependencies?

SSN → Name

What are the keys?

Is it in BCNF?

Database System - Nankai

Decompose it into BCNF

SSN	Name
123-321-99	Fred
909-438-44	Joe

FDs: SSN → Name

SSN	Phone
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

FDs: No

如果一个关系模式只含有两个属性，它是不是BCNF范式?

如果一个关系模式没有函数依赖，它是不是BCNF范式?

Database System - Nankai

Decomposition into BCNF

- Given: relation *R* with FD’s *F*.
- Look among the given FD’s for a BCNF violation *X*→*B*.
  - If any FD following from *F* violates BCNF, then there will surely be an FD in *F* itself that violates BCNF.
- Compute X^+.
  - Not all attributes, or else X is a superkey.

Database System - Nankai

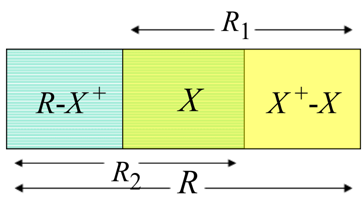
Decompose *R* Using *X* -> *B*

- Replace *R* by relations with schemas:
  1. $R_1 = X^+$.
  2. $R_2 = R - (X^+ - X)$.

Project given FD’s *F* onto the two new relations.

3. Compute the *closure* of *F* = all nontrivial FD’s that follow from *F*.
4. Use only those FD’s whose attributes are all in R_1 or all in R_2.

Decomposition Picture



Database System - Nankai

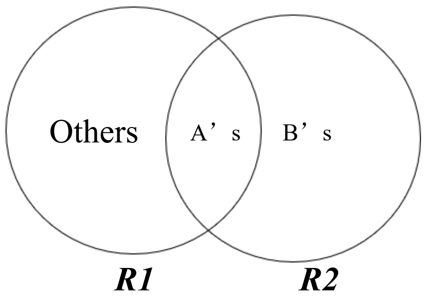
BCNF Decomposition

Find a dependency that violates the BCNF condition:

$$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$$

Heuristics: choose $B_1, B_2, \ldots B_m$ "as large as possible"

Decompose:



Any 2-attribute relation is in BCNF.

Continue until there are no BCNF violations left.

Database System - Nankai

Example

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

F = name → addr, name → favBeer, beersLiked → manf

- Pick BCNF violation name → addr.
- Close the left side: $\{name\}^+$ = {name, addr, favBeer}.
- Decomposed relations:
  1. Drinkers1(<u>name</u>, addr, favBeer)
  2. Drinkers2(<u>name</u>, <u>beersLiked</u>, manf)

Database System - Nankai

雨课堂 Rain Classroom

Example, Continued

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.
- Projecting FD's is complex in general, easy here.
- For Drinkers1(name, addr, favBeer), relevant FD's are name → addr and name → favBeer.
  - Thus, {name} is the only key and Drinkers1 is in BCNF.

Database System - Nankai

Example, Continued

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked->manf, and the only key is {name, beersLiked}.
  - Violation of BCNF.
- $beersLiked^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:
  1. Drinkers3(beersLiked, manf)
  2. Drinkers4(name, beersLiked)

Database System - Nankai

Example, Concluded

- The resulting decomposition of *Drinkers* :

1. Drinkers1(name, addr, favBeer)
2. Drinkers3(beersLiked, manf)
3. Drinkers4(name, beersLiked)

w Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

Thus,

- BCNF removes certain types of redundancy
- For examples of redundancy that it cannot remove, see "multivalued redundancy" later
- BCNF avoids info loss

However

- BCNF is not always dependency preserving
- In fact, some times we cannot find a BCNF decomposition that is dependency preserving
- Can handle this situation using 3NF
- See next few slides for example

Summary：Boyce Codd Normal Form (BCNF)

Database System - Nankai

单选题 1分

互动交流一

已知关系模式：学生（学号、姓名、系、系主任）
它存在的函数依赖是: {学号 → 姓名，学号 → 系，
请确定它是否属于BCNF范式?

A 它属于BCNF范式

B 它不属于BCNF范式

提交

Database System - Nankai

多选题 1分

互动交流二

已知关系模式：学生（学号、姓名、系、系主任
它存在的函数依赖是：学号 → 姓名，学号 → 系，
如果将它分解为：R1（学号、姓名、系主任），
，R1和R2中存在的函数依赖分别为：

A R1: {学号 → 姓名}

B R1: {学号→姓名，学号→系主任

C R2: {学号 → 系，系 → 系主任}

D R2: {学号 → 系主任，学号→系}

提交

m - Nankai

多选题 1分

互动交流三

已知关系模式：学生（学号、姓名、系、系主任
它存在的函数依赖是：学号 → 姓名，学号 → 系，
如果将它分解为：R1（学号、姓名、系主任），
，R1和R2的keys分别为：

A R1的keys: 学号

B R1的keys:（学号，系主任）

C R2的keys：学号

D R2的keys：（学号，系）

提交

m - Nankai

多选题 1分

互动交流四

已知关系模式：学生（学号、姓名、系、系主任
它存在的函数依赖是：学号 → 姓名，学号 → 系，
如果将它分解为：R1（学号、姓名、系主任），R
请判断分解后的R1和R2是否属于BCNF范式？

A R1 ∈ BCNF 成立

C R2 ∈ BCNF 成立

B R1 ∈ BCNF 不成立

D R2 ∈ BCNF 不成立

提交

m - Nankai

单选题 1分

互动交流五

The following three questions refer to a relation R(A, B, C, D,E) with functional dependencies A→B, BC→D, and E→C.

If we project R onto S(B, C, D, E), which of he following functional dependencies holds in S and also does not violate the BCNF condition for S?

A $BC \rightarrow D$

B $BE \rightarrow D$

C $B \rightarrow E$

D $E \rightarrow C$

提交

se System - Nankai

主观题 10分

互动交流六

已知关系模式：学生（学号、姓名、系、系主任）

它存在的函数依赖是：学号 → 姓名，学号 → 系，系 → 系主任

请按照课堂讲授的方法，将其分解为一组满足BCNF范式的关系模式。

提交

往年的期末考题

Consider a relation $R = (A, B, C, D)$ with FD's $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow B$

- List all keys for R.
- Is R in BCNF? If yes, briefly explain why. Otherwise,
- Write down two functional dependencies that causes this relation to violate BCNF.
- decompose further until all decomposed relations are in BCNF, and then show your final results.