CS 348 Lectures 10-11

Database Design Theory:

Theory of Normal Forms

Semih Salihoğlu

Feb 6th-11th 2025

Lectures on Relational Algebra & SQL (1)

- Main SQL clauses for querying and data manipulation
  - Founded on relational algebra
- Constraints: Primary Keys, Foreign Keys, Not NULL, General Assertions and CHECKs
- Triggers

Achieve Integrity of Database

- Views ***

Ease of Programming

  - Primary ways to get different abstractions on data
  - When materialized also a way to achieve performance
- Indexes ***
  - Fast access to some data

Performance

Lectures on Relational Algebra & SQL (2)

- Recursion: Can be considered a weak point for SQL
  - Not an elegant way to express recursive computations
  - E.g.: Try to express finding shortest paths in a graph
  - GraphDB query languages: better but minor additional support
  - Datalog: (I think) better declarative logic-based language for recursive programming
    - Some DBMSs implement it. Good for coffee/OH chat

```
Example Datalog Program
Parent(A, B) is an external relation w/ tuples (e.g., (Alice, Bob))
Ancestor(X, Y) :- parent(X, Y)
Ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y)
```

- There are alg. (e.g., “semi-naïve datalog algorithm”) that can find “fixed point” states of any set of Datalog rules

Lectures on Relational Algebra & SQL (3)

- SQL Programming Interfaces
  - SLQ programming is almost always through a programming language (PL) or framework.
  - Frameworks (e.g., Ruby on Rails) give basic functionality with no explicit SQL coding
  - For somewhat complex apps, need direct SQL through a PL

Lectures on Relational Algebra & SQL (4)

Primary Takeaway:

SQL is very high-level and a very different style of programming data processing tasks than standard procedural PLs.

Little needs to be known algorithmically to perform tasks.

Next 2 Lectures: Relational Database Design Theory

- Theory of Normal Forms (TNF): Given a set of constraints about the real-world facts that an app will store, how can we formally separate “good” and ”bad” relational db schemas?

Design 1

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000
…	…	…	…	…	…

Design 2

Inst			
iID	name	salary	depName
111	Alice	5000	CS
222	Bob	4000	Physics
…	…	….	…

Dep		
depName	bldng	budget
CS	DC	20000
Physics	PHY	30000

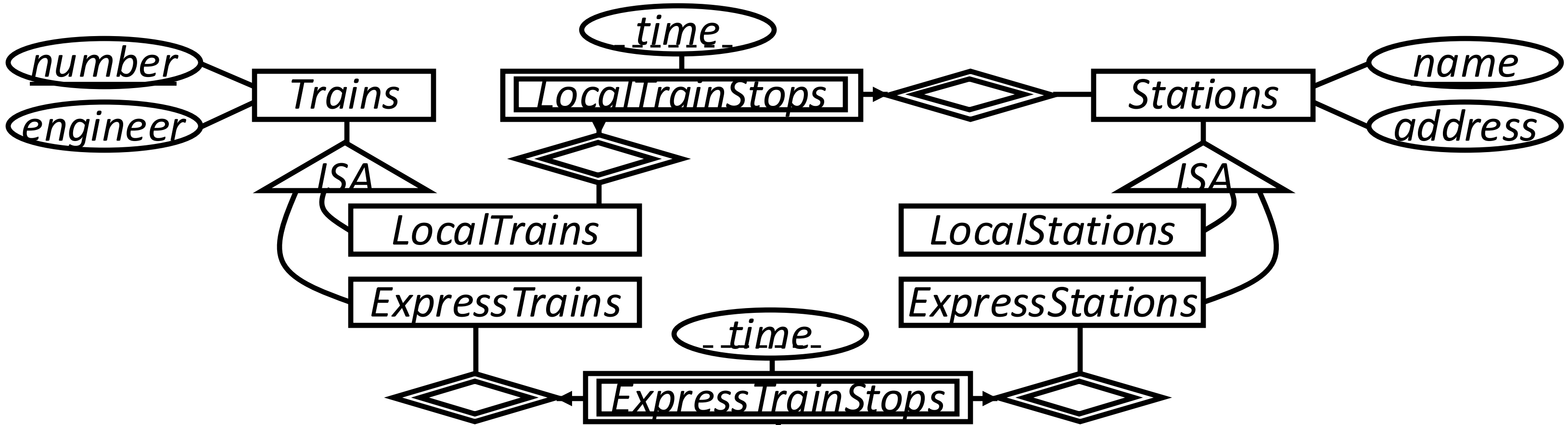
If given a depName: (bldng, budget) is unique, i.e., determined, Design 1, intuitively, is a bad design with redundancy.

- Goal of TNF: make the above intuition formal.

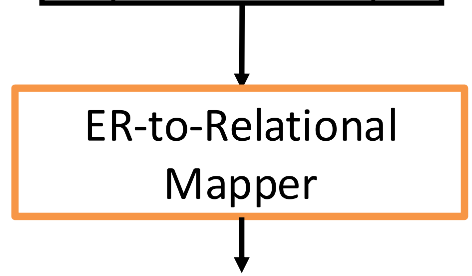
Following 2 Lectures: Entity/Relationship (ER) Model

- Often users do not directly design relational tables
- ER Model: An even higher-level data model
- Close to object-oriented programming
    - In turn, it is close to modeling data as a graph
    - How complex dbs are modeled in practice
        - Analogous to programming frameworks

Following 2 Lectures: Entity/Relationship (ER) Model



Many DBMSs provide an actual E/R Schema Designer. Can be done manually for small/medium-size db schemas.



R1		
$A_{1,1}$	…	$A_{1,m1}$
…	…	…

R2		
$A_{2,1}$	…	$A_{2,m2}$
…	…	…

…

Rn		
$A_{n,1}$	…	$A_{n,mn}$
…	…	…

- Upshot: Does not guarantee good designs as in TNF.

Still need to ensure final relations adhere to the principles of TNF

Outline For Today

1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3rd Normal Form
5. Multi-valued Dependencies and 4th Normal Form

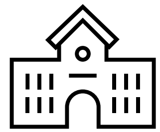
Outline For Today

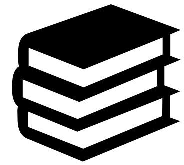
1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3rd Normal Form
5. Multi-valued Dependencies and 4th Normal Form

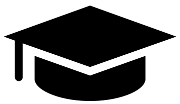
Application Constraints

- Consider a simple university DB:

Instructors


Departments


Courses


Students


- Independent of stored data: there will be external app. constraints. E.g:
  - Each instructor has 1 name, salary, and department
  - Each department has 1 building
  - Each student can have 1 advisor from each department
  - Instructor i's set of addresses are independent of the departments of i
    - *High-level idea: A "good" DB makes such constraints explicit*

Application Constraints

- Instructors: iIDs, names, salaries, departments (w/ unique iIDs)
- Departments: names, building, budget (w/ unique names)

b/c iID is key

- Constraint 1: Each instructor has 1 name, salary, and department ✓
- Constraint 2: Each department has 1 building and 1 associated budget X

b/c depName is not key

- Possible Design: 1 large table InstDep with one row for each instructor

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000
…	…	…	…	…	…

- Problem: redundant data replication. (CS, DC, 20000) repeated k times if there are k instructors in CS.

Problems of Redundancy

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000
…	…	…	…	…	…

- Harder to keep db consistent when facts are stored multiple times. E.g:
- If CS's building changed to E4 => need to update 3 rows
- Suppose Bob is the only instructor in Physics and retires (a delete):
  - Deletion of Bob's tuple: Physics department, which might exist, is deleted unless extra work is done
- If new department (w/out yet an instructor) is added: new row w/ NULLs

Redundancy Is Determined By App. Constraints

- Courses: cID, term, iID, capacity

Course			
cID	term	iID	capacity
CS348	F21	Semih	100
CS341	F21	Lap Chi	80
CS348	S21	Xi	100
CS348	W20	Xi	100
CS350	W19	Salem	130

- Unclear if this is redundant or not. Depends on external app constraint:
- If courses have 1 associated capacity (independent of term): Redundant
- O.w repetition may be necessary and reflects similarity across entities.

Redundancy Is Determined By App. Constraints

- Courses: cID, term, iID, capacity

Course			
cID	term	iID	capacity
CS348	F21	Semih	100
CS341	F21	Lap Chi	80
CS348	S21	Xi	100
CS348	W20	Xi	100
CS350	W19	Salem	130
CS348	W22	David	200

- Unclear if this is redundant or not. Depends on external app constraint:
- If courses have 1 associated capacity (independent of term): Redundant
- O.w repetition may be necessary and reflects similarity across entities.
- Takeaway: Constraints are external to the db/app and need to be inputs in a db design theory.

Solution To Redundancy: Decompositions

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

Inst			
iID	name	salary	depName
111	Alice	5000	CS
222	Bob	4000	Physics
333	Carl	5200	CS
444	Diana	5500	CS

Dep		
depName	bldng	budget
CS	DC	20000
Physics	PHY	30000

Desiderata for Decompositions (1)

- D1 (Lossless): If R is decomposed into R1 and R2, then:

$$R = R1 \bowtie R2$$

- Lossless-ness achieved by decomposing on an appropriate key

Inst			
iID	name	salary	depName
111	Alice	5000	CS
222	Bob	4000	Physics
333	Carl	5200	CS
444	Diana	5500	CS

$\bowtie$

Dep		
depName	bldng	budget
CS	DC	20000
Physics	PHY	30000

RESULT					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

=

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

Example Lossy Decomposition

InstDep					
ID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

ID
111
222
333
444

⋈

name
Alice
Bob
Carl
Diana

⋈

salary
5000
4000
5200
5500

⋈

depName
CS
Physics

⋈

bldng
DC
PHY

⋈

budget
20000
30000

RESULT					
ID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
111	Bob	5200	CS	PHY	30000
…	…	…	…	…	…

X Can’t tell what’s fact and what’s not.

Desiderata for Decompositions (2)

- D2 (Locality of Constraints): If the app had a constraint C, we would prefer to check C in a single relation
- Will discuss more in 3rd Normal Form. Stay tuned.

Outline For Today

1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3rd Normal Form
5. Multi-valued Dependencies and 4th Normal Form

Functional Dependencies (FDs): Generalized Uniqueness Constraints

- Informally: Let X, Y be sets of attributes. A functional dependency $X \rightarrow Y$ holds if for each possible value of attributes X there is only 1 possible Y value, i.e. X "determines" Y uniquely (independent of other values in a tuple).
- Formally: Let t[A] be a tuple t's projection on attributes A

Dfn: Let X, Y be sets of attributes. An fd $X \rightarrow Y$ holds in a relation R, if given t_1 and $t_2 \in R$ s.t. $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$ holds.

Example FDs

- Captures generalized uniqueness constraints (beyond keys):

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000
…	…	…	…	…	…

- Constraint 1: Each iID has 1 name and salary
  - iID → name, salary
- Constraint 2: Each depName has 1 building & 1 associated budget
  - depName → bldng, budget
- Key constraints: Each iID, depName is unique in InstDep
  - iID, depName → name, salary, bldgn, budget

Some FD Vocabulary

- We take FDs as given, i.e., cannot be inferred from a relation instance.
- FDs limit legal instances of a relation $R(A_1, \ldots, A_m)$
- Given a set $\mathcal{F}$ of fds on R, on all *legal instances of R*, each $F \in \mathcal{F}$ hold.

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
111	Alice	5000	Biology	BIO	50000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

- Suppose $\mathcal{F}$: (i) iID $\rightarrow$ name, salary; (ii) depName $\rightarrow$ bldng, budget
- E.g: The above instance is a legal instance
- E.g: iID $\rightarrow$ name, salary holds on the above instance.
- Won't need this vocabulary much in lecture. May see in assignments.

Implied FDs: Armstrong's Axioms

- A set of fds can imply other fds via 3 intuitive rules: Armstrong's Axioms

1. Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$ (trivially)
   - iID, name → iID
   - English: Each iID and name value takes a unique iID value
2. Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$ (trivially)
   - If iID → salary then iID, bldng → salary, bldng
   - English: if each iID takes a unique salary value, then each (iID, bldng) value pair takes a unique (salary, bldng) value

InstDep					
<u>iID</u>	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
…	…	…	…	…	…

Implied FDs: Armstrong's Axioms

3. Transitivity: if X → Y and Y → Z, then X → Z

- Suppose each instructor can be in a single department and each dep has a single budget
- FD1: iID → depName FD2: depName → budget, then

  iID → budget
- English: If each iID value takes a unique depName value, which in turn takes a unique budget value, then each iID value takes a unique budget value.

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
…	…	…	…	…	…

Other Rules Implied by Armstrong's Axioms

1. Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

   Proof:

   i. $X \rightarrow YZ$

   ii. $YZ \rightarrow Y$ (by reflexivity)

   iii. $X \rightarrow Y$ (by transitivity)

2. Union: If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$ (Prove as exercise)

3. Pseudo-transitivity: If $X \rightarrow Y$ and $YZ \rightarrow T$ then $XZ \rightarrow T$ (Prove as exercise)

$\mathcal{F}^+$: Closure of $\mathcal{F}$

Dfn: Let $\mathcal{F}$ be a set of fds. The closure $\mathcal{F}^+$ of $\mathcal{F}$ is the set of all fds implied by $\mathcal{F}$.

- Ex: $\mathcal{F}$: iID→name, depName & depName→bldng
- $\mathcal{F}^+$: $\mathcal{F}$ ∪ iID→iID; iID,email→name,email (trivial ones) … ∪ iID→bldng (transitivity) etc..

InstDep				
iID	name	email	depName	bldng
$\boxed{111}$	Alice	alice@gmail	CS	$\boxed{DC}$
$\boxed{111}$	Alice	alice@hotmail	CS	$\boxed{DC}$
222	Bob	bob@gmail	Physics	PHY
222	Bob	bob@hotmail	Physics	PHY
333	Carl	carl@gmail	CS	DC
…	…	…	…	…

Exercise Showing an FD is in $\mathcal{F}^+$

- Consider an Inst_Proj relation of instructors and their research projects

InstProj						
iID	name	projID	projName	projDep	hours	funds

- (i) iID → name; (ii) projID → projName, projDep;

  (iii) iID, projID → hours; (iv) projDep, hours → funds;

- Prove iID, projID → funds

1. iID, projID → hours (by fd iii)
2. projID → projName, projDep (by fd ii)
3. iID, projID → hours, projName, projDep (by pseudo-transitivity of 1 & 2)
4. iID, projID → funds (by transitivity of 3, and fd iv) (+ decomposition)

How To Compute $\mathcal{F}^+$ from $\mathcal{F}$

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
      add the resulting functional dependency to F^+
until F^+ does not change any further

Figure 8.7 A procedure to compute F^+.

Outline For Today

1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3^{rd} Normal Form
5. Multi-valued Dependencies and 4^{th} Normal Form

Boyce-Codd Normal Form (BCNF)

Dfn (BCNF): Given a set of fds $\mathcal{F}$, a relation R is in BCNF iff:

$\forall$ X $\rightarrow$ Y $\in \mathcal{F}^+$ s.t. XY $\subseteq$ R, one of the following two conditions hold:

1. X $\rightarrow$ Y is trivial (i.e., Y $\subseteq$ X)
2. X is a super key of R (i.e., X $\rightarrow$ R)

Q: Why does X $\rightarrow$ R imply X is super key?

Example Relations in BCNF and not in BCNF (1)

- Note we only need to look at the schema of R and $\mathcal{F}^+$
- As before suppose $\mathcal{F}$: (i) iID→name,salary; (ii) depName→bldng,budget

InstDep					
iID	name	salary	depName	bldng	budget

Q: Is InstDep in BCNF?

A: No b/c iID is not key. Can check by computing $\mathcal{F}^+$ = $\mathcal{F}$ ∪ all trivial fds. B/c can't apply transitivity to the fds in $\mathcal{F}$ to generate more non-trivial fds.

Example Relations in BCNF and not in BCNF (2)

- As before suppose $\mathcal{F}$: (i) iID→name,salary; (ii) depName→bldng,budget

R1			
iID	name	salary	depName

Q: Is R1 in BCNF?

A: No b/c iID is still not key.

R2		
depName	bldng	budget

Q: Is R2 in BCNF?

A: Yes b/c depName is key.

R3		
iID	name	salary

Q: Is R3 in BCNF?

A: Yes

R4	
iID	depName

Q: Is R4 in BCNF?

A: Yes b/c no non-trivial FDs

R2		
depName	bldng	budget

Greedy BCNF Decomposition Algorithm (1)

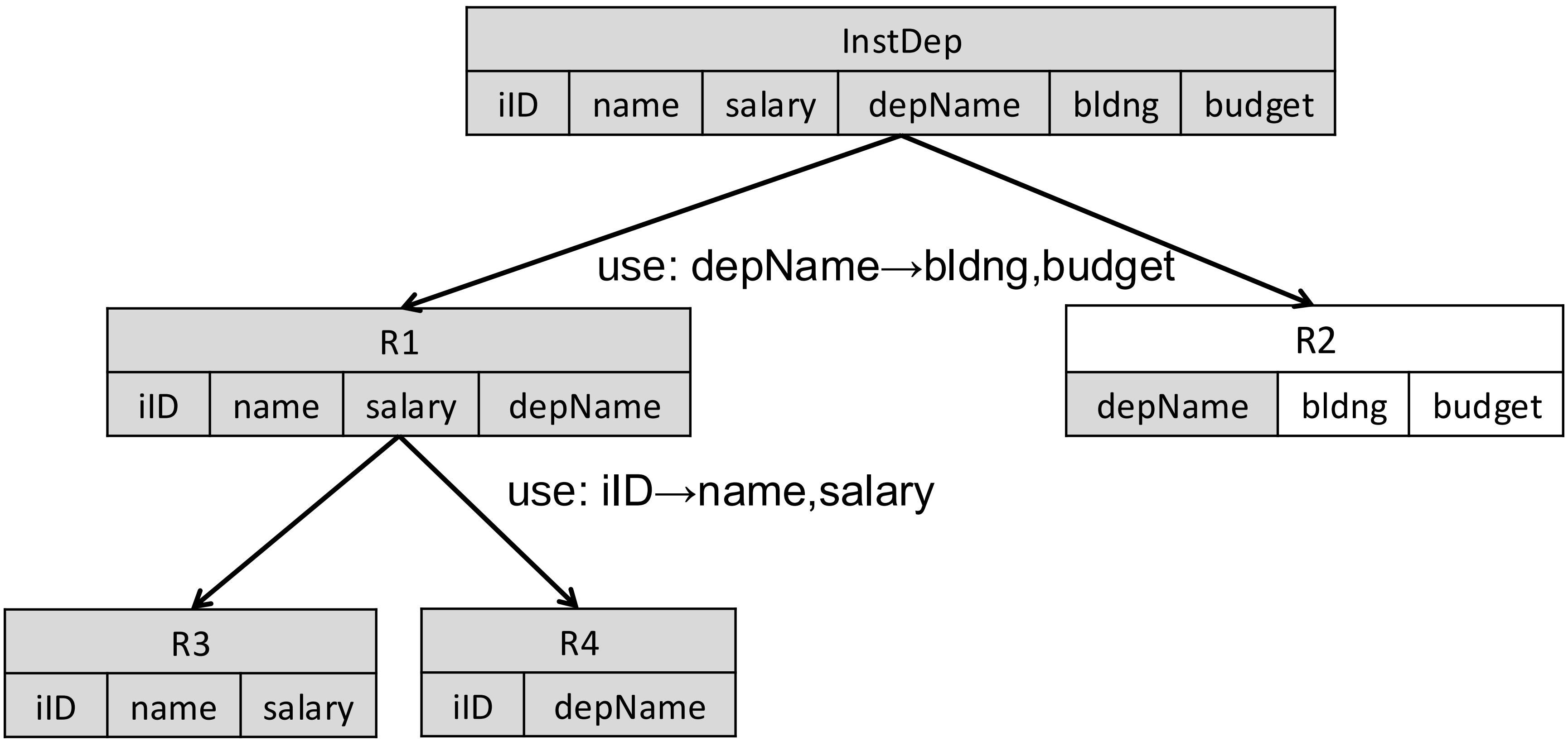
```
Very High-level
Input: R, 𝓕⁺
rels = { R }
1. find an fd X → Y violating BCNF on a relation Rᵢ ∈ rels
2. Split Rᵢ into Rᵢ₁ = X ∪ (Rᵢ – Y) and Rᵢ₂ = X ∪ Y;
   rels = (rels – Rᵢ) ∪ Rᵢ₁ ∪ Rᵢ₂
3. repeat 1-2 until no such fd can be found.
```

- Several properties of the alg (won't formally prove):

1. Always returns a set of relations R_1, …, R_k s.t. R_j is in BCNF and this is a lossless decomposition of R, i.e., $R_1 \bowtie \ldots R_j = R$. (Why? Stay tuned.)
2. The output is *not* unique. (Exercise: show a simple example with a relation and two fds to demonstrate this)

Greedy BCNF Decomposition Algorithm (2)

- $\mathcal{F}$: (i) iID→name,salary; (ii) depName→bldng,budget
- $\mathcal{F}^+ = \mathcal{F} \cup$ all trivial fds.



Why Do We Consider $\mathcal{F}^+$ instead of $\mathcal{F}$?

- After a split some non-trivial direct or implied fds from $\mathcal{F}$ (through transitivity) could remain and cause redundancy
- Ex: $\mathcal{F}$: (i) iID→depName; (ii) depName→bldng

InstDep			
iID	email	depName	bldng
111	a@gmail	CS	DC
111	a@hotmail	CS	DC

use: iID→depName

iID	depName
111	CS

iID	email	bldng
111	a@gmail	DC
111	a@hotmail	DC

still repetition

need: iID→bldg ∈ $\mathcal{F}^+$

iID	bldng
111	DC

iID	email
111	a@gmail
111	a@hotmail

Why is BCNF A Lossless Decomposition? (1)

```
Very High-level
Input: R, 𝓕+
rels = { R }
1. find an fd X → Y violating BCNF on a relation Ri ∈ rels
   s.t. XY ∈ attr(Ri) (attr(Ri) is the attributes/cols of Ri)
2. Split Ri into Ri1 = X ∪ (Ri − Y) and Ri2 = XY;
     result = (result − Ri) ∪ Ri1 ∪ Ri2
3. repeat 1-2 until no such fd can be found.
```

- By construction of the algorithm, X, which are the join attributes or R_{i1} and R_{i2}, i.e., $R_{i1} \cap R_{i2}$, is a key in one of the split relations.
- Since it's a key, the join is 1-1:
  - i.e., each tuple t in R_{i1} will join with 1 tuple t' in R_{i2}.

Why is BCNF A Lossless Decomposition? (2)

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000
555	Ed	2500	Physics	DC	20000

use: depName→bldng,budget

R1			
iID	name	salary	depName
111	Alice	5000	CS
222	Bob	4000	Physics
333	Carl	5200	CS
444	Diana	5500	CS
555	Ed	2500	Physics

R2		
depName	bldng	budget
CS	DC	20000
Physics	PHY	30000

Since CS is key in R2, each R1 tuple can join with only 1 R2 tuple.

Why is BCNF Decomp Alg. *Not* “Depend. Preserving”?

- Decomposing on non-trivial fd_1 can “break” a non-trivial fd_2
- Ex: fd_1: iID → depName: an instructor belongs to 1 dep

  fd_2: sID, depName → iID: a student has 1 advisor from each dep.

Note: Not in BCNF. Why?

DeptAdvisor

sID	iID	depName
s1	111	CS
s1	555	Physics
s2	111	CS

use: iID → depName

R1

sID	iID
s1	111
s1	555
s2	111

R2

iID	depName
111	CS
555	Physics

- Can no longer check fd_2 in a single relation. Need to join R1 and R2.

Outline For Today

1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3rd Normal Form
5. Multi-valued Dependencies and 4th Normal Form

Dependency Preservation (1)

Dfn (Restriction): Let $\mathcal{F}$ be a set of fds. The restriction of $\mathcal{F}$ to a relation R_i is the set of fds in $\mathcal{F}^+$ that contain all its attributes in R_i.

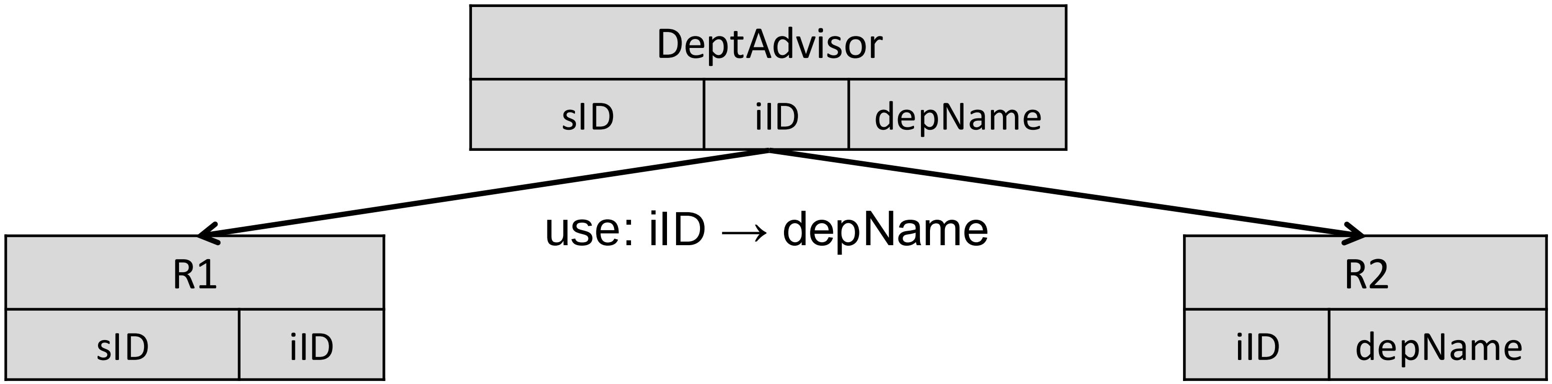
Dfn (Dependency Preservation): Let $\mathcal{F}$ be a set of fds on a relation R. Let D={R_1, …, R_k} be a decomposition of R with F_1', …, F_k' as the restrictions of $\mathcal{F}$ onto R_i for i=1,…,k. Let $\mathcal{F}'$=F_1' ∪ … ∪ F_k'. D is dependency preserving iff $\mathcal{F}^+ = \mathcal{F}'^+$.

- Note some fds ∈ $\mathcal{F}^+$ may not be "localized" in a single relation in D but every fd ∈ $\mathcal{F}^+$ must be implied by a set of fds that are localized to a single relation.

Dependency Preservation (2)

- $\mathcal{F}$ = fd_1: iID → depName: an instructor belongs to 1 dep

  fd_2: sID → iID: a student has 1 advisor.
- Note $\mathcal{F}^+$ contains sID → depName.



- Cannot check sID → depName in a single relation but that's OK.
- We can still check it without any joins:
  - fd_1: iID → depName can be checked in R2
  - fd_2: sID → iID can be checked in R1
  - fd_1 and fd_2 imply sID → depName

3^{rd} Normal Form (3NF): Relaxation of BCNF To Allow Some "Reasonable" Redundancy

Dfn: Given a set of fds $\mathcal{F}$, a relation R is in 3NF iff:

$\forall$ X $\rightarrow$ Y $\in \mathcal{F}^+$ s.t. XY $\subseteq$ R, one of the following three conditions hold:

1. X $\rightarrow$ Y is trivial (i.e., Y $\subseteq$ X)
2. X is a super key of R (i.e., X $\rightarrow$ R)
3. Each attribute in Y – X is part of a candidate key

- Recall candidate key K: a set of "minimal" attributes K that form a key for R, i.e., no proper subset of K is a key for R.

3NF Example

- Recall non-BCNF example:

  fd_1: iID → depName: an instructor belongs to 1 dep

  fd_2: sID, depName → iID: a student has 1 advisor from each dep.

DeptAdvisor		
sID	iID	depName
s1	111	CS
s1	555	Physics
s2	111	CS

- ∉ BCNF but ∈ 3NF b/c iID → depName is a non-trivial, non-key fd but:
  - sID,depName -> iID is a key. Moreover a *candidate key.*
  - B/c: depName nor sID alone is a key (check $\mathcal{F}^+$)

Key point: 3NF relations can have redundant repetition (e.g., due to iID → depName) that BCNF does not.

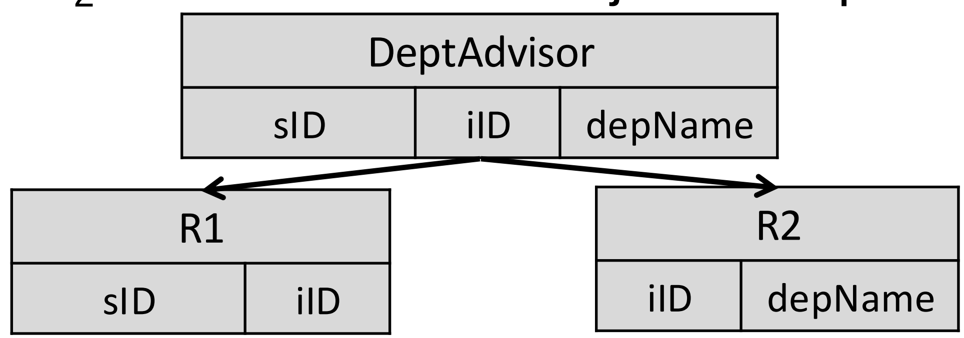
But this repetition allows us to verify every fd without any joins.

Intuition Behind 3rd Rule in 3NF (1)

- BCNF requires that in the final relations fds are either *trivial* or *keys*.
- Trivial fds can't lead to repetition. Neither can keys by definition.
- If an fd is non-trivial and non-key, a relation R needs to be decomposed
- Problem: Not every relation R and $\mathcal{F}^+$ has a dependency preserving decomposition into BCNF. i.e., a decomposition using fd$_1$ can "break" the attributes of another non-trivial fd$_2$ into 2 relations so a join is required.

Ex: $\boxed{\text{fd}_1\text{: iID} \rightarrow \text{depName}}$

fd$_2$: sID, depName $\rightarrow$ iID



- 3NF's 3rd condition allows non-trivial, non key fds X$\rightarrow$Y if each attribute $A_i \in Y–X$ is part of a candidate key.
- E.g., depName is part of a cand. key, so we don't need to decompose.

Intuition Behind 3rd Rule in 3NF (2)

- Question: Why does the 3rd condition guarantee every relation has a dependency preserving decomposition?

3rd cond: fd*: $X \rightarrow Y$ is OK if each $A_i \in Y - X$ is part of a candidate key.

High-level Intuition: $Y - X$ are the "repeated" values due to fd*.

Let $A_i \in Y - X$. If A_i is part of a candidate key in the relation R, then:

$\exists$ an fd_{ck}: $Z, A_i \rightarrow R$. So if we decompose according to fd*:

R will split, so we need a join to check fd_{ck}.

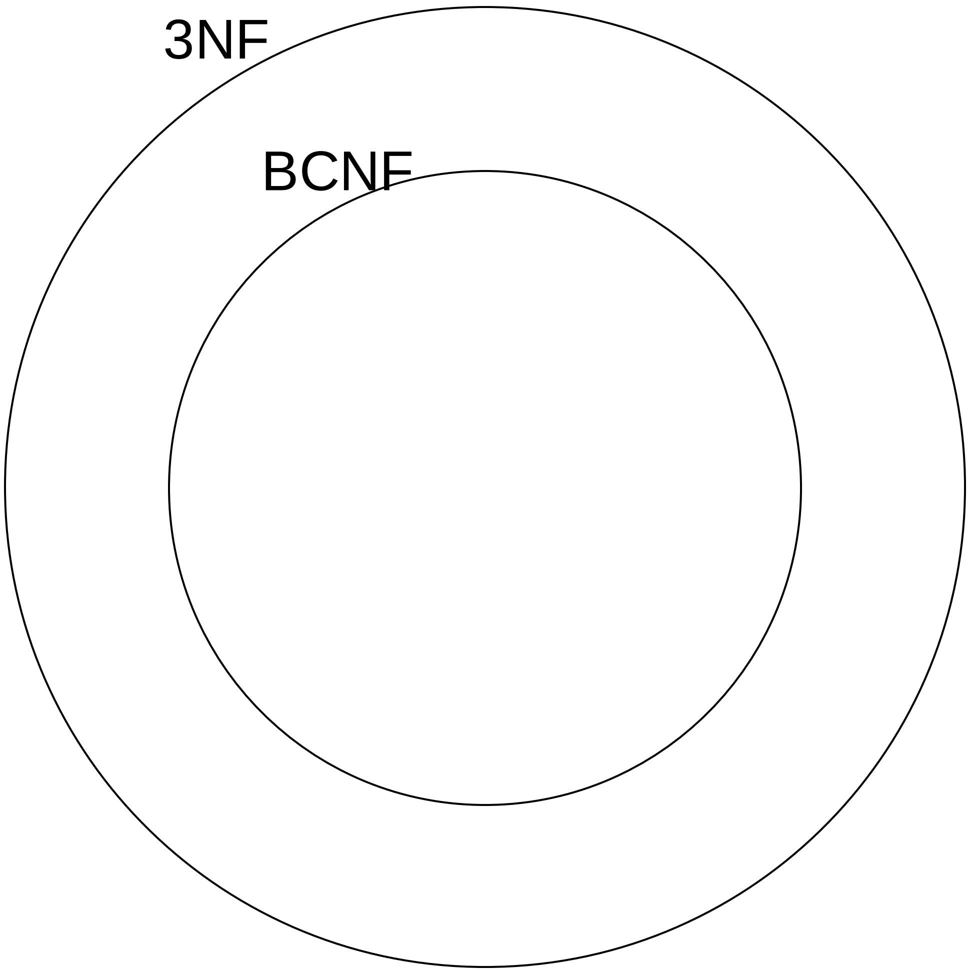
The 3rd rule allows us to not split in these cases.

Note: This is not a proof.

A simpler proof for Question: study a 3NF decomposition alg. (next slide) and observe that it is dependency preserving by construction and is guaranteed to output relations in 3NF.

3NF and BCNF Venn Diagram

- Every relation in BCNF is in 3NF but not vice versa.



3NF Bottom-Up Decomp. Alg (1): Minimal/Canonical Covers

- A set of FDs $\mathcal{F}$ is minimal if:
  1. Every right-hand side of a FD in $\mathcal{F}$ is a single attribute
  2. For no $X \rightarrow A$ is the set $\mathcal{F} - \{X \rightarrow A\}$ equivalent to $\mathcal{F}$, i.e., Let $\mathcal{F}' = \mathcal{F} - \{X \rightarrow A\}$, then $\mathcal{F}'^{+} = \mathcal{F}^{+}$.
  3. For no $X \rightarrow B$ and Z a proper subset of X is the set $(\mathcal{F} - \{X \rightarrow B\}) \cup \{Z \rightarrow B\}$ equivalent to $\mathcal{F}$
- Ex: R(A, B, C, D, E, F, G)

$\mathcal{F}$: $\boxed{fd_1: A \rightarrow BC}$ ← Fails Condition 1

$fd_2: D \rightarrow E$

$\boxed{fd_3: A,D \rightarrow F}$ ← Fails Condition 2: e.g., fd_2 and fd_4 imply fd_3

$fd_4: A, E \rightarrow F$

$\boxed{fd_5: D,E \rightarrow G}$ ← Fails Condition 3: difficult to directly see, but you can try that having instead fd_5: D→G has the same closure, i.e., is the same set of fds.

3NF Bottom-Up Decomp. Alg (2)

```
let F_c be a canonical cover for F;
i := 0;
for each functional dependency α → β in F_c
      i := i + 1;
      R_i := α β;
if none of the schemas R_j, j = 1, 2, . . . , i contains a candidate key for R
   then
      i := i + 1;
      R_i := any candidate key for R;
/* Optionally, remove redundant relations */
repeat
      if any schema R_j is contained in another schema R_k
         then
           /* Delete R_j */
           R_j := R_i;
           i := i - 1;
until no more R_js can be deleted
return (R_1, R_2, . . . , R_i)
```

Figure 8.12 Dependency-preserving, lossless decomposition into 3NF.

Outline For Today

1. Application Constraints and Decompositions
2. Functional Dependencies
3. Boyce-Codd Normal Form (BCNF) & BCNF Decomposition Alg.
4. Dependency Preservation and 3rd Normal Form
5. Multi-valued Dependencies and 4th Normal Form

Restatement of FDs: Conditional Independence Among Attribute Sets (1)

- Recall the informal dfn of FDs: Let X, Y be sets of attributes. A functional dependency $X \rightarrow Y$ holds if for each possible value of attributes X there is only 1 possible value Y value. I.e. X “determines” Y uniquely (independent of other values in a tuple).
- Equivalently: Let “RST= R- $(X \cup Y)$. $X \rightarrow Y$ means given a set of X values $(x_1, \ldots, x_k)$:
  1. Y values of tuples w/ $(x_1, \ldots, x_k)$: are independent of values in RST.
  2. And there is only 1 set of Y values.

Restatement of FDs: Conditional Independence Among Attribute Sets (2)

- Let "RST= R- (X ∪ Y). X → Y means given a set of X values (x_1, …, x_k):
  1. Y values of tuples w/ (x_1, …, x_k): are independent of values in RST.
  2. And there is only 1 set of Y values.

InstDep					
iID	name	salary	depName	bldng	budget
111	Alice	5000	CS	DC	20000
222	Bob	4000	Physics	PHY	30000
333	Carl	5200	CS	DC	20000
444	Diana	5500	CS	DC	20000

- Ex: depName → bldng, budget
  - RST: iID, name, salary

Multivalued Dependencies (MVDs): Generalized Conditional Independence Constraints

- MVDs remove uniqueness constraint: given X, Y is independent of RST.

Dfn (MVD): Let X, Y be sets of attributes. An MVD $X \twoheadrightarrow Y$ holds in R,

if given t_1 and $t_2 \in R$ s.t. $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$,

then $\exists$ t_3, t_4 s.t:

1. $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
2. $t_3[Y] = t_1[Y]$ and $t_3[RST] = t_2[RST]$
3. $t_4[Y] = t_2[Y]$ and $t_4[RST] = t_1[RST]$

	X	Y	RST
t_1	x*	y_1	rst_1
t_2	x*	y_2	rst_2
…	…	…	…

Multivalued Dependencies (MVDs): Generalized Conditional Independence Constraints

- MVDs remove uniqueness constraint: given X, Y is independent of RST.

Dfn (MVD): Let X, Y be sets of attributes. An MVD $X \twoheadrightarrow Y$ holds in R,

if given t_1 and $t_2 \in R$ s.t. $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$,

then $\exists$ t_3, t_4 s.t:

1. $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
2. $t_3[Y] = t_1[Y]$ and $t_3[RST] = t_2[RST]$
3. $t_4[Y] = t_2[Y]$ and $t_4[RST] = t_1[RST]$

	X	Y	RST
t_1	x*	y_1	rst_1
t_2	x*	y_2	rst_2
t_3	x*	y_1	rst_2
t_4	x*	y_2	rst_1

Multivalued Dependencies (MVDs): Generalized Conditional Independence Constraints

- Example Constraint: given an instructor i, i's emails are independent of the departments and buildings of these departments (but not unique).
  - iID $\twoheadrightarrow$ email

InstDep				
iID	name	email	depName	bldng
111	Alice	alice@gmail	CS	DC
111	Alice	alice@hotmail	CS	DC
111	Alice	alice@uw.ca	Physics	PHY
222	Bob	bob@hotmail	Physics	PHY
333	Carl	carl@gmail	CS	DC
…	…	…	…	…

Let x* be a set of values for X attributes (possibly > 1)

for any X values , R contains exactly:

$$\{x^*\} \text{ X } \{\mathbf{\Pi}_Y(\boldsymbol{\sigma}_{X=x^*}(R)\} \text{ X } \{\mathbf{\Pi}_{RST}(\boldsymbol{\sigma}_{X=x^*}(R)\}$$

Cartesian Product

Multivalued Dependencies (MVDs): Generalized Conditional Independence Constraints

- Example Constraint: given an instructor i, i's emails are independent of the departments and buildings of these departments (but not unique).
  - iID ↠ email

InstDep				
iID	name	email	depName	bldng
111	Alice	alice@gmail	CS	DC
111	Alice	alice@hotmail	CS	DC
111	Alice	alice@uw.ca	Physics	PHY
222	Bob	bob@hotmail	Physics	PHY
333	Carl	carl@gmail	CS	DC
111	Alice	alice@uw.ca	CS	DC
111	Alice	alice@gmail	Physics	PHY
111	Alice	alice@hotmail	Physics	PHY

Let x* be a set of values for X attributes (possibly > 1)

for any X values , R contains exactly:

$$\{x^*\} \underbrace{\mathrm{X}}_{\text{Cartesian Product}} \{\Pi_Y(\sigma_{X=x^*}(R)\} \mathrm{X} \{\Pi_{RST}(\sigma_{X=x^*}(R)\}$$

Note: ∈ BCNF b/c InstDep has no non-trivial, non-key fds. Yet has repetition.

FDs are Specialized MVDs

- More formal way to see FDs are specialized MDs (i.e., other than the informal definition that says MDs remove the uniqueness constraints)

Dfn (MVD): Let X, Y be sets of attributes. An MVD $X \twoheadrightarrow Y$ holds in a relation R, if given t_1 and $t_2 \in R$ s.t. $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$, then $\exists$ t_3, t_4 s.t:

1. $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
2. $t_3[Y] = t_1[Y]$ and $t_3[RST] = t_2[RST]$
3. $t_4[Y] = t_2[Y]$ and $t_4[RST] = t_1[RST]$

- Suppose $X \rightarrow Y$ holds, then by dfn if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$, so the condition of MVD trivially holds.
- I.e. no non-trivial Cartesian product needs to be taken since $\{\mathbf{\Pi}_Y(\boldsymbol{\sigma}_{X=x^*}(R)\}$ has size 1)

Closure of MVDs

- Given a set of mvds D, $D's$ closure D^+ is the set of mvds logically implied by D.
- Similar to Armstrong's Axioms, there are a set of inference rules to compute D^+.
- More rules than Armstrong's 3 axioms. See Appendix B.1.1 of the text book's 6^{th} edition here (or C.1.1 of 7^{th} edition).

4NF: Avoiding All Repetition Due to MVDs

Dfn (4NF): Given a set of mvds D (which by dfn include all fds), a relation R is in 4NF iff:

$\forall$ X$\twoheadrightarrow$Y $\in D^+$ s.t. XY $\subseteq$ R, one of the following two conditions hold:

1. X $\twoheadrightarrow$ Y is trivial (i.e., Y $\subseteq$ X)
2. X is a super key of R (i.e., X $\rightarrow$ R)

4th Normal Form Decomposition Algorithm

- Simply replace FDs in the BCNF Decomposition Alg with MVDs

```
Very High-level
Input: R, D+
rels = { R }
1. find an mvd X ↠ Y violating 4NF on a relation Ri ∈ rels
   s.t. XY ∈ attr(Ri) (attr(Ri) is the attributes/cols of Ri)
2. Split Ri into Ri1 = X ∪ (Ri – Y) and Ri2 = XY;
     result = result - Ri ∪ Ri1 ∪ Ri2
3. repeat 1-2 until no such mvd can be found.
```

4NF Decomposition Example

InstDep				
iID	name	email	depName	bldng
111	Alice	alice@gmail	CS	DC
111	Alice	alice@hotmail	CS	DC
111	Alice	alice@uw.ca	Physics	PHY
222	Bob	bob@hotmail	Physics	PHY
333	Carl	carl@gmail	CS	DC
111	Alice	alice@uw.ca	CS	DC
111	Alice	alice@gmail	Physics	PHY
111	Alice	alice@hotmail	Physics	PHY

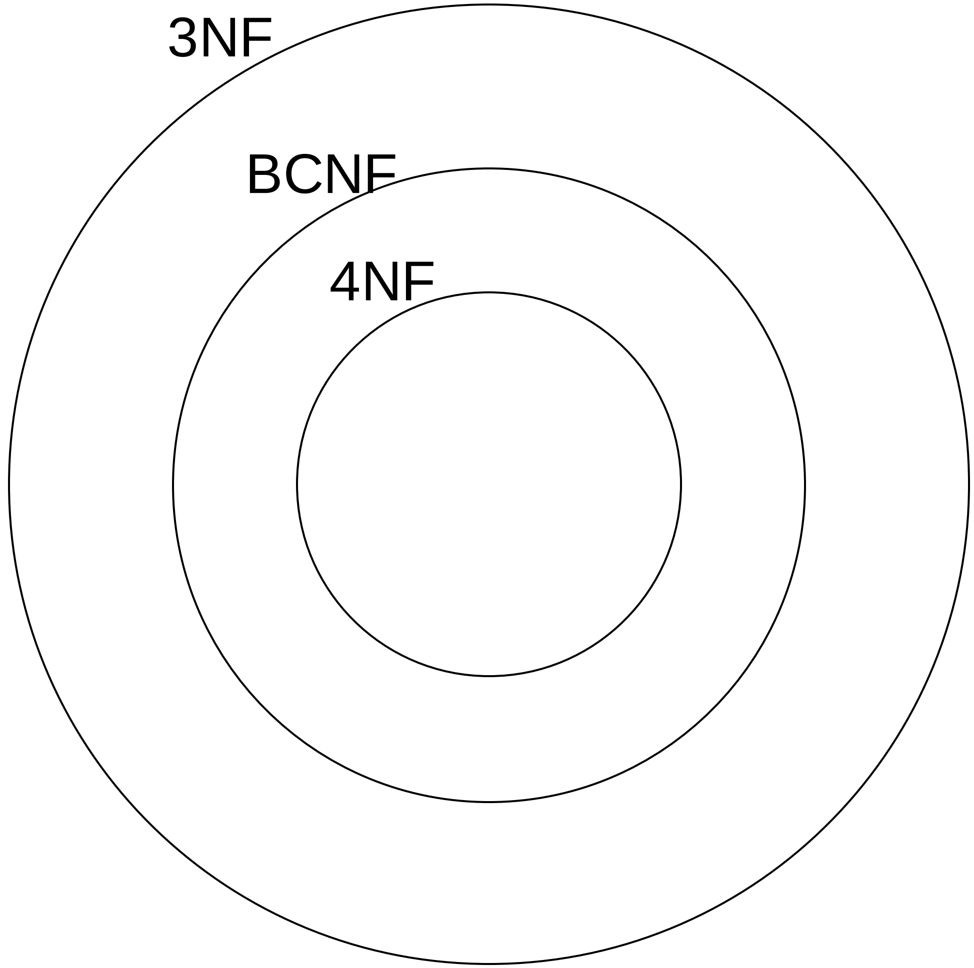
iID $\twoheadrightarrow$ email

R1			
iID	name	depName	bldng
111	Alice	CS	DC
111	Alice	Physics	PHY
222	Bob	Physics	PHY
333	Carl	CS	DC

R2	
iID	email
111	alice@gmail
111	alice@hotmail
111	alice@uw.ca
222	bob@hotmail
333	carl@gmail

Venn Diagram of Normal Forms

- Every relation in 4NF is in BCNF but not vice versa (see previous slide).



Enforcing FDs/MVDs in Practice

- Key constraints are specialized fds that RDBMSs can enforce
- But no direct support in RDBMSs for general FDs and MVDs.
- But can use table level CHECKs and triggers to enforce these
- There is also little support for specifying fds/mvds and decomposing relations in practice.
- Users manually decompose relations if they observe repetition in design
- But TNF is very useful when thinking about application constraints and their implications for redundancy
- Try to target relations in 4NF or BCNF in practice. Often many natural designs are already in these forms.

Summary

- Theory of Normal Forms (TNF): Given a specification of the real-world facts that an app will store, how can we formally separate “good” and ”bad” relational db schemas?
- Ultimate Goal: Remove redundancy/repetition in design by factoring out ”conditionally” independent parts.
- Redundancy depends on app constraints. Same exact relation can sometimes be redundant or not depending on app constraints.
- FDs: Generalized Uniqueness Constraints
- BCNF: Using FDs and the schema of R, can formally state whether R has redundant repetition due to uniqueness constraints.
- 3NF: Allows some redundancy to “localize” checking of fds to 1 relation
- 4NF: Most strict. Also does not allow repetition due to non-unique conditional independence relationships.