CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#17: Schema Refinement & Normalization

CMU SCS

Administrivia

- HW6 is due Tuesday March 24th.

CMU SCS

Correction: Implied FDs

Product(name, color, category, dept, price**)**

name	color	category	dept	price
Gizmo	Green	Gadget	Toys	9.99
Widget	Black	Gadget	Toys	49.99
Gizmo	Green	Squirrels	Garden	19.99

Provided FDs

name → color

category → dept

color, category → dept

Implied FDs

name, category → price

CMU SCS

Relational Model: Keys

- **Super Key**:
  - A set of attributes that uniquely identifies a tuple.
- **Candidate Key**:
  - A ***minimal*** set of attributes that uniquely identifies a tuple.
- **Primary Key**:
  - Usually just the candidate key.

Correction: Super Key Example

Super Key!

Produ(name, c r, category, dept, price)

name	color	category	dept	price
Gizmo	Green	Gadget	Toys	9.99
Widget	Black	Gadget	Toys	49.99
Gizmo	Green	Squirrels	Garden	19.99

Provided FDs

name → color

category → dept

color, category → price

Implied FDs

name, category → price

Today's Class

- The dangers of bad database design
- Decomposition Goals
- Normal Forms
- Relational Model vs. NoSQL

Example

Loans(bname, bcity, assets, cname, loanId, amt**)**

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

Tuple meaning: Christos has a loan (L-17) for $1000 with DJ Snake taken out of the Downtown branch in Pittsburgh, which has assets of $9M.

Redundancy Problems

- Update Anomalies
- Insert Anomalies
- Delete Anomalies
- Wasted Space

Decomposition

- Split a single relation **R** into a set of relations **{R_1,…R_n}**.
- Not all decompositions are "good"

Goals of Decomposition

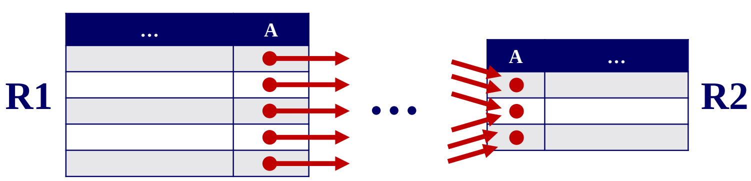
- **Loseless Joins** (chpt. 19.5.1)
  - Want to be able to reconstruct original relation by joining smaller ones using a natural join.
- **Dependency Preservation** (chpt. 19.5.2)
  - Want to minimize the cost of constraints based on FD's.
- **Redundancy Avoidance**
  - Avoid unnecessary data duplication.

Must have this!

Nice to have, but still okay without it.

Lossless Decomposition

- **Main Idea:** Intersecting attributes must form a super key for one of the resulting smaller relations.



|R1|=*n*
A not a key for R1

|R2|≤*n*
A is a key for R2 and
Cardinality with R1 is *n*:1

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt**)**

R1(bname, bcity, assets, cname**)**

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Christos
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Christos
Downtown	Pittsburgh	$9M	DJ Snake

R2(cname, loanId, amt**)**

cname	loanId	amt
Christos	L-17	$1000
Obama	L-23	$2000
Christos	L-93	$500
DJ Snake	L-17	$1000

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

CMU SCS

Lossless Decomposition

R1(bname, bcity, assets, cname) ⋈ **R2**(cname, loanId, amt)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Christos
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Christos
Downtown	Pittsburgh	$9M	DJ Snake

cname	loanId	amt
Christos	L-17	$1000
Obama	L-23	$2000
Christos	L-93	$500
DJ Snake	L-17	$1000

Cannot recover original table with a join!

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-17	$1000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

CMU SCS

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt)

R1(bname, bcity, assets, cname)
R2(cname, loanId, amt)

- This is a **bad** decomposition because it causes a lossy join:
  - The ⋈ adds meaningless tuples.
  - By adding noise, have lost meaningful information as a result of the decomposition.

CMU SCS

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt)

R1(bname, bcity, assets, cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Christos
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Christos
Downtown	Pittsburgh	$9M	DJ Snake

R2(bname, loanId, amt)

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

CMU SCS

Lossless Decomposition

R1(bname, bcity, assets, cname) ⋈ **R2**(bname, loanId, amt)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Christos
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Christos
Downtown	Pittsburgh	$9M	DJ Snake

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

More garbage data!

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Christos	L-23	$2000
Downtown	Pittsburgh	$9M	Obama	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000
Downtown	Pittsburgh	$9M	DJ Snake	L-23	$2000

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt**)**

R1(bname, bcity, assets, cname**)**
R2(bname, loanId, amt**)**

- This one is **lossy** too because **R1⋈R2** has 7 tuples where as **Loans** originally had 4.

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt**)**

R1(bname, assets, cname, loandId**)**

bname	assets	cname	loanId
Downtown	$9M	Christos	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Christos	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId, bcity, amt**)**

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

Lossless Decomposition

R1(bname, assets, cname, loanId**)** ⋈ **R2(**loanId, bcity, amt**)**

bname	assets	cname	loanId
Downtown	$9M	Christos	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Christos	L-93
Downtown	$9M	DJ Snake	L-17

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Compton	$500

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

Lossless Decomposition

Loans(bname, bcity, assets, cname, loanId, amt**)**

R1(bname, assets, cname, loanId**)**
R2(loanId, bcity, amt**)**

- This one is **lossless! R1⋈R2** has 4 tuples
- A decomposition of **R** into **R1∪R2** is loseless iff:
  - **R1∩R2→R1** ***or*** **R1∩R2→R2**
  - Intersecting attributes must form a super key for one of the resulting smaller relations

Dependency Preservation

- **Main Idea:** Original FDs cannot span multiple tables.
- Why does this matter?
  – It would be expensive to check (assuming that our DBMS supports ASSERTIONS).

Dependency Preservation

Loans(bname, bcity, assets, cname, loanId, amt**)**

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

Dependency Preservation

R1(bname, assets, cname, loanId**)**

bname	assets	cname	loanId
Downtown	$9M	Christos	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Christos	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId, bcity, amt**)**

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

Dependency Preservation

```
CREATE ASSERTION bname-bcity
 CHECK (NOT EXISTS
  (SELECT * FROM R1 AS x1, R2 AS y1,
                 R1 AS x2, R2 AS y2
    WHERE x1.loanId = y1.loanId
      AND x2.loanId = y2.loanId
      AND x1.loanId = x2.loanId
      AND x1.bname = x2.bname
      AND y1.bcity <> y2.bcity))
```

bname → bcity

Dependency Preservation

Loans(bname, bcity, assets, cname, loanId, amt**)**

R1(bname, assets, cname, loanId**)**
R2(loanId, bcity, amt**)**

- To ensure that FD checking is efficient, only a single relation should be examined for each FD.

Dependency Preservation

- To test whether the decomposition **R={R$_1$,…R$_n$}** preserves **R**'s FD set **F**:
  1. Compute **F+**
  2. Compute **G** as the union of the set of FDs in **F+** that are covered by **{R$_1$,…R$_n$}**
  3. Compute **G+**
  4. If **F+=G+**, then **{R$_1$,…R$_n$}** is Dependency Preserving

Dependency Preservation

R1(A, B, C) R2(C, D)
F = {A→B, AB →D, C→D}

(A→D) ∈ F+

…2} dep… serving?

…s covered by R1

FDs covered by R2

1. …B, AB… →D}
2. **G** = {A→B} ∪ {C→D}
3. **G+** = {A→B, C→D}
4. **F+ ≠ G+** because (A→D) ∈ (**F+ - G+**)

(A→D) ∉ G+

Decomposition is not DP

Dependency Preservation

R1(A, B, D) R2(C, D)
F = {A→B, AB →D, C→D}

- I… **}** dependency preserving?

FDs covered

G+ cannot have FD's not in F+

1. …B, AB→D, A→D, C→D}
2. …A→B, A→D, AB→D} ∪ {C→D}
3. **G+** = {A→B, AB→D, A→D, C→D}
4. **F+ = G+**

Decomposition is DP

Dependency Preservation

R1(A, B, D) R2(C, D)
F = {A→B, AB →D, C→D}

- Bonus Question: Is the decomposition **R={R1, R2}** lossless?

R1

A	B	D
A3	B_A1	D_B_A3
A1	B_A1	D_B_A1
A1	B_A1	D_B_A1
A2	B_A2	D_B_A2
A3	B_A3	D_B_A3

⋈

R2

C	D
C1	D_B_A3
C2	D_B_A1
C3	D_B_A1
C4	D_B_A2
C5	D_B_A3

Dependency Preservation

R1(A, B, D) R2(C, D)
F = {A→B, AB →D, C→D}

- Bonus Question: Is the decomposition **R={R1, R2}** lossless?

Decomposition is not Lossless

A	B	C	D
A3	B_A3	C1	D_B_A3
A3	B_A3	C1	D_B_A3
A1	B_A1	C2	D_B_A1
A1	B_A1	C2	D_B_A1
A1	B_A1	C3	D_B_A1
A1	B_A1	C3	D_B_A1
A2	B_A2	C4	D_B_A2
A3	B_A3	C5	D_B_A3
A3	B_A3	C5	D_B_A3

Redundancy Avoidance

- **Main Idea:** Want to avoid duplicate entries in a relation for a FD.
- When there exists some FD X→Y covered by relation and X is not a super key

Redundancy Avoidance

R(A, B, C)
F = {B →C}

- The super keys for **R** are all sets of attributes that include A.

A	B	C
A1	B1	C_B1
A2	B1	C_B1
A3	B2	C_B2
A4	B2	C_B2
A5	B2	C_B2
A6	B3	C_B3
A7	B3	C_B3

Decomposition Summary

- **Lossless Joins**
  - Motivation: Avoid information loss.
  - Goal: No noise introduced when reconstituting universal relation via joins.
  - Test: At each decomposition $\mathbf{R=(R_1 \cup R_2)}$, check whether $\mathbf{(R_1 \cap R_2) \to R_1}$ ***or*** $\mathbf{(R_1 \cap R_2) \to R_2}$

Decomposition Summary

- **Dependency Preservation**
  - Motivation: Efficient FD assertions.
  - Goal: No global integrity constraints that require joins of more than one table with itself.
  - Test: $\mathbf{R=(R_1 \cup \ldots \cup R_n)}$ is dependency preserving if closure of FD's covered by each $\mathbf{R_1}$ = closure of FD's covered by $\mathbf{R=F}$

Decomposition Summary

- **Redundancy Avoidance**
  - Motivation: Avoid update, delete anomalies.
  - Goal: Avoid update anomalies, wasted space.
  - Test: For an X→Y covered by $\mathbf{R_n}$, X should be a super key of $\mathbf{R_n}$.

Today's Class

- The dangers of bad database design
- Decomposition Goals
- Normal Forms
- Relational Model vs. NoSQL

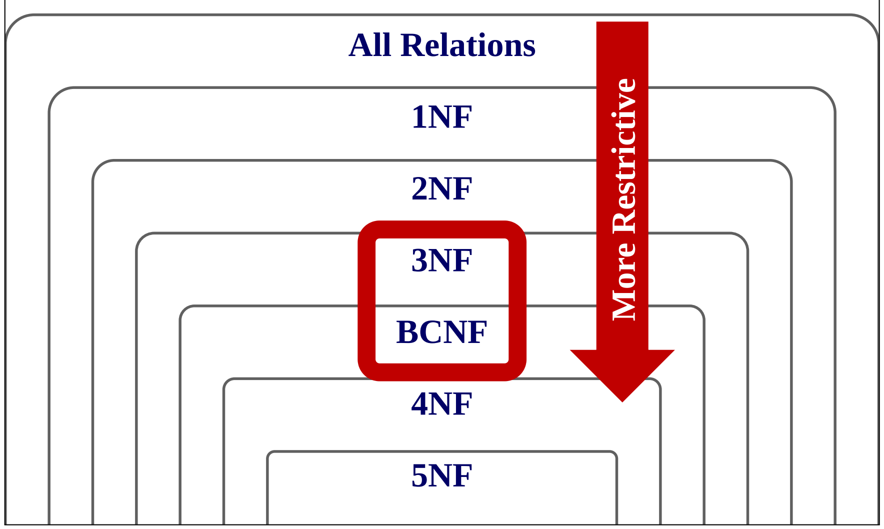
Normal Forms

- Now we know how to derive more FDs, we can then:
  - Search for “bad” FDs
  - If there are such, then decompose the table into two tables, repeat for the sub-tables.
  - When done, the database schema is **normalized**

Normal Forms

- A ***normal form*** is a characterization of a schema decomposition in terms of the properties that satisfies.

The Universe of Relations



Normal Forms

- 1st Normal Form = All Tables are Flat
- 2nd Normal Form = Obsolete
- 3rd Normal Form = **Today**
- Boyce-Codd Normal Form = **Today**
- 4th and 5th = See textbook
- 6th = Most (normal) people never need this.

First Normal Form

- All types must be atomic.

Loans(bname, bcity, assets, cname, loanId, amt**)**

Violates 1NF

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	[Christos, DJ Snake]	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500

First Normal Form

- All types must be atomic.

Loans(bname, bcity, assets, cname, loanId, amt**)**

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Christos	L-17	$1000
Downtown	Pittsburgh	$9M	Obama	L-23	$2000
Compton	Los Angeles	$2M	Christos	L-93	$500
Downtown	Pittsburgh	$9M	DJ Snake	L-17	$1000

Valid 1NF!

Second Normal Form

- 1NF and non-key attribute fully depend on the candidate key.

R1(bname, assets, cname, loandId**)**

bname	assets	cname	loanId
Downtown	$9M	Christos	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Christos	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId, bcity, amt**)**

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Functional Dependencies:
bname → bcity, assets
loanId → amt, bname

Boyce-Codd Normal Form

- BCNF guarantees no redundancies and no lossless joins (but not DP).
- A relation **R** with FD set **F** is in BCNF if for all non-trivial X→Y in **F+**:
  - X→**R** (i.e., X is a super key)

Boyce-Codd Normal Form

R(A, B, C)
F = {A→B, B →C}

- Is **R** in BCNF?
- Consider the non-trivial dependencies in **F+**:
  1. A→B, A→**R** (A is a key)
  2. A→C, A→**R** (A is a key)
  3. B→C, B↛A (B is not a key)

R is not in BCNF

Boyce-Codd Normal Form

R1(A, B) **R2**(B, C)
F = {A→B, B →C}

- Are **R1**, **R2** in BCNF?
- Test **R1**
  1. A→B, A→**R1** (A is a key)
- Test **R2**
  1. B→C, B→**R2** (B is a key)

R1, R2 are in BCNF

Boyce-Codd Normal Form

- Given a schema **R** and a set of FDs **F**, we can always decompose R into **{R_1,…R_n}** such that
  - **{R_1,…R_n}** are in BCNF
  - The decompositions are lossless.
- But some BCNF decompositions might lose dependencies.

BCNF Decomposition Algorithm

- Given a relation **R** and a FD set **F**:
  1. Compute **F+**
  2. **Result** ← {**R**}
  3. While some **R_i** ∈ **Result** not in BCNF, do:
     a) Choose (X→Y) ∈ **F+** such that (X→Y) is covered by **R_i** and X↛**R_i**
     b) Decompose **R_i** on (X→Y):
        $R_{i,1}$ ← X ∪ Y
        $R_{i,2}$ ← **R_i** - Y
     c) **Result** ← (**Result** - {**R_i**}) ∪ {**$R_{i,1}$**, **$R_{i,2}$**}

$R_{i,1}$ includes Y

$R_{i,2}$ doesn't include Y

BCNF Example

R(name, ssn, phone#, city)
F = {ssn→name, city}

name	ssn	phone#	city
Christos	123-45-6789	555-555-5555	Pittsburgh
Christos	123-45-6789	666-666-6666	Pittsburgh
Lil' Fame	987-65-4321	777-777-7777	Brooklyn
Lil' Fame	987-65-4321	888-888-8888	Brooklyn

- Step #1:
  - **F+** ←{ssn→name, ssn→city, (ssn→name,city)}

BCNF Example

R(name, ssn, phone#, city)
F = {ssn→name, city}

name	ssn	phone#	city
Christos	123-45-6789	555-555-5555	Pittsburgh
Christos	123-45-6789	666-666-6666	Pittsburgh
Lil' Fame	987-65-4321	777-777-7777	Brooklyn
Lil' Fame	987-65-4321	888-888-8888	Brooklyn

- Step #3: **R** is not in BCNF
  - **3(a):** We choose (ssn→name,city) as the FD to split on because ssn does not get us the phone# (i.e., it is not the super key).

BCNF Example

R(name, ssn, phone#, city)
F = {ssn→name, city}

name	ssn	phone#	city
Christos	123-45-6789	555-555-5555	Pittsburgh
Christos	123-45-6789	666-666-6666	Pittsburgh
Lil' Fame	987-65-4321	777-777-7777	Brooklyn
Lil' Fame	987-65-4321	888-888-8888	Brooklyn

- Step #3: **R** is not in BCNF
  - **3(b):** Split R based on (ssn→name,city) such that **R1**=(name, ssn, city) and **R2**=(ssn, phone#)

BCNF Example

R1(name, ssn, city) R2(ssn, phone#)
F = {ssn→name, city}

name	ssn	city
Christos	123-45-6789	Pittsburgh
Lil' Fame	987-65-4321	Brooklyn

ssn	phone#
123-45-6789	555-555-5555
123-45-6789	666-666-6666
987-65-4321	777-777-7777
987-65-4321	888-888-8888

- Step #3: **R** is not in BCNF
  - **3(c):** The resulting schema is now **R**={**R1**, **R2**}

BCNF Example

R1(name, ssn, city) R2(ssn, phone#)
F = {ssn→name, city}

name	ssn	city
Christos	123-45-6789	Pittsburgh
Lil' Fame	987-65-4321	Brooklyn

ssn	phone#
123-45-6789	555-555-5555
123-45-6789	666-666-6666
987-65-4321	777-777-7777
987-65-4321	888-888-8888

- Step #3: Check if {**R1**, **R2**} are not in BCNF
  - Lossless?
  - Anomalies?

A Problem with BCNF

R1(unit, comp) R2(unit, product)
F = {unit→comp, (comp,product→unit)}

unit	comp
Basketball	Christos Inc.
Baseball Bat	Christos Inc.

	product
Baseball Bat	Sports Equipment

Super Key: **(unit, product)**

We keep unit→comp
but we lose (comp,product→unit)

- At this point we don't have any problems:
  - We're in BCNF and all local FDs are satisfied.

A Problem with BCNF

R1(unit, comp) R2(unit, product)
F = {unit→comp, (comp,product→unit)}

unit	comp
Basketball	Christos Inc.
Baseball Bat	Christos Inc.

⋈

unit	product
Basketball	Sports Equipment
Baseball Bat	Sports Equipment

unit	comp	product
Basketball	Christos Inc.	Sports Equipment
Baseball Bat	Christos Inc.	Sports Equipment

This violates (comp,product→unit)

A Problem with BCNF

- We started with a relation **R** and its dependency set **FD**.
- We decomposed **R** into BCNF relations {$R_1,\ldots R_n$} with their own {$FD_1,\ldots,FD_n$}.
- We can reconstruct R from {$R_1,\ldots R_n$}.
- But we cannot reconstruct FD from {$FD_1,\ldots,FD_n$}.

Third Normal Form

- 3NF preserves dependencies but may have some anomalies.
- A relation **R** with FD set **F** is in 3NF if for every X→Y in **F+**:
  - X→Y is trivial, ***or***
  - X is a super key, ***or***
  - Y is part of a candidate key

3NF Decomposition Algorithm

- Given a relation **R** and a FD set **F**:
  1. Compute **Fc**
  2. **Result** ← ∅
  3. For each FD (X→Y) ∈ **Fc**, add a relation $\mathbf{R_i}$(X, Y) to **Result**
  4. If **Result** is not lossless, add a table with an appropriate key.

3NF Example

R(A, B, C)
F = {A→B, B→C}

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A3	B_A3	C_B_A3
A2	B_A2	C_B_A2

- Step #1: Compute canonical cover
  - **Fc** ←{A→B, B→C}

3NF Example

R(A, B, C)
F = {A→B, B→C}

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2

- Step #3: Split **R** based on its FDs
  - **R1**(A, B) because A→B
  - **R2**(B, C) because B→C

3NF Example

R1(A, B) R2(B, C)
F = {A→B, B→C}

A	B
A1	B_A1
A2	B_A2
A2	B_A2

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A2	C_B_A2

- Step #3: Split **R** based on its FDs
  - **R1**(A, B) because A→B
  - **R2**(B, C) because B→C

3NF Example

R1(A, B) R2(B, C)
F = {A→B, B→C}

A	B
A1	B_A1
A2	B_A2
A2	B_A2

⋈

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A2	C_B_A2

- Step #4: Check whether **{R1, R2}** is lossless.

3NF Example

R1(A, B) R2(B, C)
F = {A→B, B→C}

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2

- Step #4: Check whether **{R1, R2}** is lossless.
  - Nope!
  - Add **R3** based on the join attribute A→C

3NF Example

R1(A, B) R2(B, C) R2(A, C)
F = {A→B, B→C}

A	B
A1	B_A1
A2	B_A2
A2	B_A2

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A2	C_B_A2

A	C
A1	C_B_A1
A2	C_B_A2
A2	C_B_A2

- Step #4: Check whether **{R1, R2}** is lossless.
  - Nope!
  - Add **R3** based on the join attribute A→C

BCNF vs. 3NF

- **BCNF**:
  - No Anomalies, but may lose some FDS.
  - In practice, this is what you want.
- ***3NF:***
  - Keeps all FDs, but may have some anomalies.

Today's Class

- The dangers of bad database design
- Decomposition Goals
- Normal forms
- Relational Model vs. NoSQL

The Rise of NoSQL

- Prior to the early 2000s, few people needed a high-performance DBMS.
- Key tenants of the NoSQL movement:
  - Joins are slow, so we will denormalize tables.
  - Transactions are slow and we need to be on-line 24/7, so let's drop ACID.

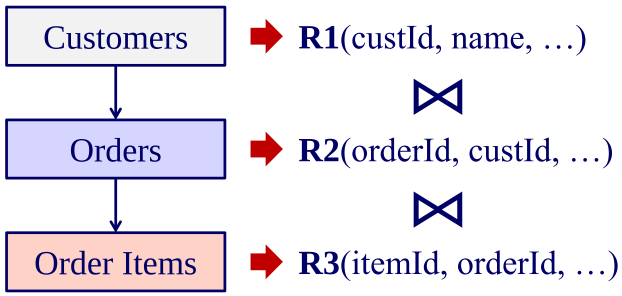
Next Week

MongoDB

- Document Model = JSON / XML
- No server-side joins:
  - "Pre-join" collections by embedding related documents inside of each other.

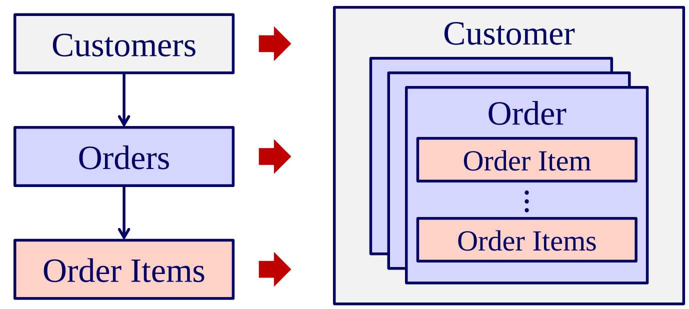
BCNF Example

- A customer has orders and each order has order items.



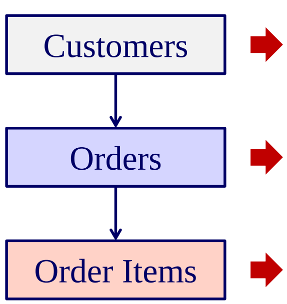
Denormalization Example

- A customer has orders and each order has order items.



Denormalization Example

- A customer has orders and each order has order items.




```
{
  "custId" : 1234,
  "custName" : "Christos",
  "orders" : [
    {
      "orderId" : 9999,
      "orderItems" : [
        {
          "itemId" : "XXXX",
          "price"  : 19.99
        },
        {
          "itemId" : "YYYY",
          "price"  : 29.99
        },
      ]
    }
  ]
}
```


Denormalization Example

- No joins is not a by-product of using the document model, but it makes logical denormalization more "natural":
- MongoDB isn't the only document DBMS:
  - CouchDB
  - RavenDB
  - RethinkDB
  - MarkLogic (XML)

Actually supports joins!

Next Week

- Physical Database Design + Tuning
- The (Awesome) World of Transactions