Lectures 6:
Design Theory Part II

Today's Lecture

1. Boyce-Codd Normal Form
   - ACTIVITY
2. Decompositions & 3NF
   - ACTIVITY
3. MVDs
   - ACTIVITY

1. Boyce-Codd Normal Form

What you will learn about in this section

1. Conceptual Design
2. Boyce-Codd Normal Form
3. The BCNF Decomposition Algorithm
4. ACTIVITY

Conceptual Design

Back to Conceptual Design

Now that we know how to find FDs, it's a straight-forward process:

1. Search for "bad" FDs

2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs

3. When done, the database schema is *normalized*

Recall: there are several normal forms…

Boyce-Codd Normal Form (BCNF)

- Main idea is that we define "good" and "bad" FDs as follows:

    - X → A is a *"good FD" if X is a (super)key*
        - In other words, if A is the set of all attributes

    - X → A is a *"bad FD"* otherwise

- We will try to eliminate the "bad" FDs!

Boyce-Codd Normal Form (BCNF)

- Why does this definition of "good" and "bad" FDs make sense?

- If X is *not* a (super)key, it functionally determines *some* of the attributes; therefore, those other attributes can be duplicated
  - Recall: this means there is redundancy
  - And redundancy like this can lead to data anomalies!

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is **in BCNF** if:

if $\{A_1, \ldots, A_n\} \rightarrow B$ is a *non-trivial* FD in R

then $\{A_1, \ldots, A_n\}$ **is a superkey** for R

Equivalently: $\forall$ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

In other words: there are no "bad" FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

`{SSN} → {Name,City}`

This FD is *bad* because it is **not** a superkey

$\Longrightarrow$ Not in BCNF

What is the key?
{SSN, PhoneNumber}

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

`{SSN} → {Name,City}`

This FD is now *good* because it is the key

Let's check anomalies:
- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):

BCNF Decomposition Algorithm

BCNFDecomp(R):

Find *a set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

Find a set of attributes X which has non-trivial "bad" FDs, i.e. is not a superkey, using closures

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

If no "bad" FDs found, in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$

Let Y be the attributes that ***X functionally determines*** (+ that are not in X)

And let Z be **the** ***complement*****, the other attributes that it** ***doesn't***

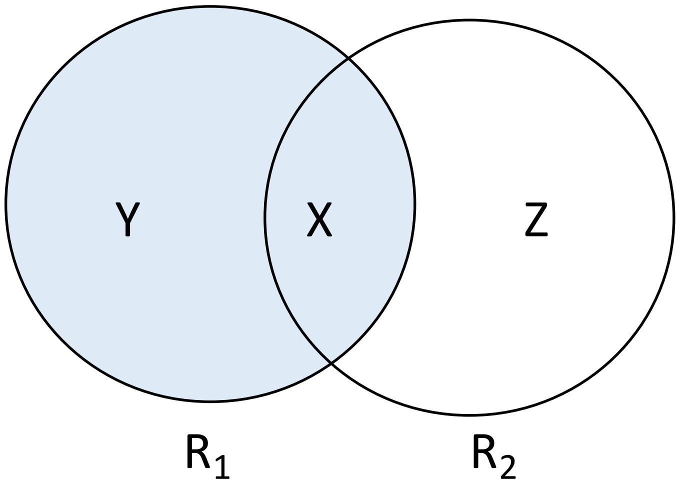
BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

Split into one relation (table) with X plus the attributes that X determines (Y)…



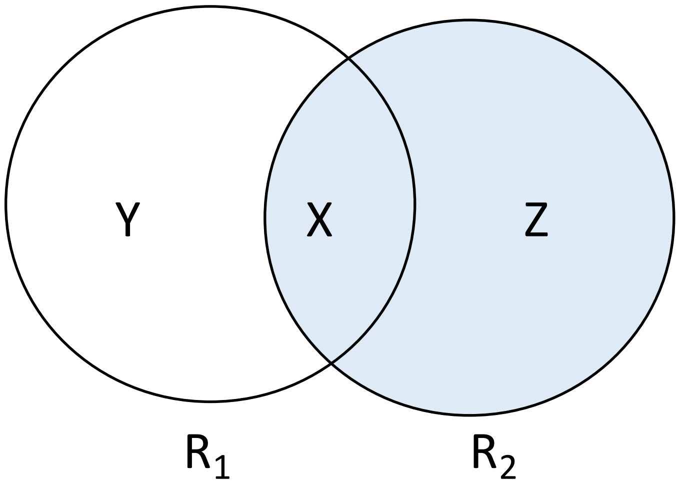
BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

And one relation with X plus the attributes it *does not* determine (Z)



BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

Proceed recursively until no more "bad" FDs!

Example

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

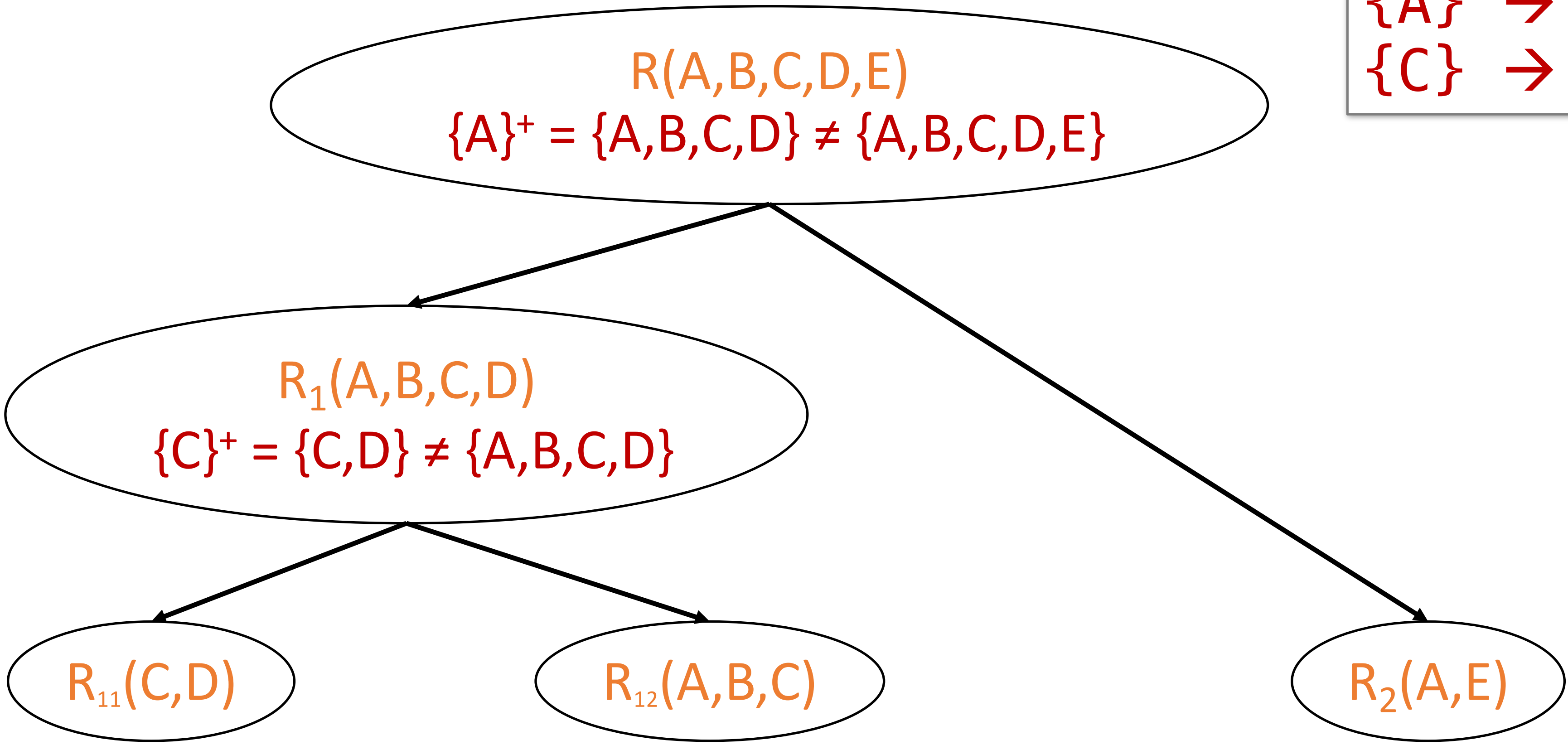
```
R(A,B,C,D,E)
```

```
{A} → {B,C}
{C} → {D}
```

Example

R(A,B,C,D,E)

$\{A\} \rightarrow \{B,C\}$
$\{C\} \rightarrow \{D\}$



DB-WS06a.ipynb

2. Decompositions

Recap: Decompose to remove redundancies

1. We saw that **redundancies** in the data ("bad FDs") can lead to data anomalies

2. We developed mechanisms to **detect and remove redundancies by decomposing tables into BCNF**
   1. BCNF decomposition is *standard practice*- very powerful & widely used!

3. However, sometimes decompositions can lead to **more subtle unwanted effects…**

When does this happen?

Decompositions in General

$$
\begin{tikzcd}
 & R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p) \arrow[dl] \arrow[dr] & \\
R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) & & R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)
\end{tikzcd}
$$

R_1 = the *projection* of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = the *projection* of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Theory of Decomposition

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

Name	Price
Gizmo	19.99
OneClick	24.99
~~Gizmo~~	~~19.99~~

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Sometimes a decomposition is "correct"

I.e. it is a <u>Lossless decomposition</u>

Lossy Decomposition

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

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

However sometimes it isn't

What's wrong here?

Lossless Decompositions

$R(A_1,...,A_n,B_1,...,B_m,C_1,...,C_p)$

$R_1(A_1,...,A_n,B_1,...,B_m)$ $\quad$ $R_2(A_1,...,A_n,C_1,...,C_p)$

A decomposition R to (R1, R2) is **lossless** if R = R1 Join R2

Lossless Decompositions

$R(A_1,...,A_n,B_1,...,B_m,C_1,...,C_p)$

$R_1(A_1,...,A_n,B_1,...,B_m)$ $R_2(A_1,...,A_n,C_1,...,C_p)$

If $\{A_1, ..., A_n\} \rightarrow \{B_1, ..., B_m\}$
Then the decomposition is lossless

Note: don't need
$\{A_1, ..., A_n\} \rightarrow \{C_1, ..., C_p\}$

BCNF decomposition is always lossless. Why?

A problem with BCNF

Problem: To enforce a FD, must reconstruct original relation—*on each insert!*

Note: This is historically inaccurate, but it makes it easier to explain

A Problem with BCNF

Unit	Company	Product
…	…	…

{Unit} → {Company}
{Company,Product} → {Unit}

Unit	Company
…	…

Unit	Product
…	…

{Unit} → {Company}

We do a BCNF decomposition on a “bad” FD:
$\{Unit\}^{+} = \{Unit, Company\}$

We lose the FD {Company,Product} → {Unit}!!

So Why is that a Problem?

<u>Unit</u>	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

`{Unit} → {Company}`

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

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

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

Violates the FD `{Company,Product} → {Unit}`!!

The Problem

- We started with a table R and FDs F
- We decomposed R into BCNF tables R_1, R_2, …
  with their own FDs F_1, F_2, …
- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

Practical Problem: To enforce FD, must reconstruct R—*on each insert!*

Possible Solutions

- Various ways to handle so that decompositions are all lossless / no FDs lost
  - For example 3NF- stop short of full BCNF decompositions. See Bonus Activity!

- Usually a tradeoff between redundancy / data anomalies and FD preservation…

BCNF still most common- with additional steps to keep track of lost FDs…

3. MVDs

What you will learn about in this section

1. MVDs
2. ACTIVITY

Multi-Value Dependencies (MVDs)

- A multi-value dependency (MVD) is another type of dependency that could hold in our data, ***which is not captured by FDs***

- Formal definition:
  - Given a relation **R** having attribute set **A**, and two sets of attributes $\mathbf{X}, \boldsymbol{Y} \subseteq \boldsymbol{A}$
  - The ***multi-value dependency (MVD)*** $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if
  - ***for any tuples*** $\boldsymbol{t_1}, \boldsymbol{t_2} \in \boldsymbol{R}$ s.t. $\boldsymbol{t_1}[X] = \boldsymbol{t_2}[X]$***,*** there exists a tuple $\boldsymbol{t_3}$ s.t.:
    - $\mathbf{t_1[X] = t_2[X] = t_3[X]}$
    - $\mathbf{t_1[Y] = t_3[Y]}$
    - $\mathbf{t_2[A \backslash Y] = t_3[A \backslash Y]}$
      - *Where A \ B means "elements of set A not in set B"*

Multi-Value Dependencies (MVDs)

- One less formal, literal way to phrase the definition of an MVD:

- ***The MVD*** $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if for any pair of tuples with the same X values, the "swapped" pair of tuples with the same X values, but the other permutations of Y and A\Y values, is also in R

Ex: X = {x}, Y = {y}:

x	y	z
1	0	1
1	1	0

For $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ to hold must have…

x	y	z
1	0	1
1	1	0
1	0	0
1	1	1

Note the connection to a local *cross-product…*

Multi-Value Dependencies (MVDs)

- Another way to understand MVDs, in terms of *conditional independence:*

- ***The MVD*** $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if given X, Y is conditionally independent of A \ Y and vice versa…

Here, given x = 1, we know for ex. that:
y = 0 → z = 1

I.e. z is conditionally ***dependent*** on y given x

x	y	z
1	0	1
1	1	0

Here, this is not the case!

I.e. z is conditionally ***independent*** of y given x

x	y	z
1	0	1
1	1	0
1	0	0
1	1	1

Multiple Value Dependencies (MVDs)

A “real life” example…

Grad student CA thinks:
“Hmm… what is real life??
Watching a movie over the
weekend?”

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

Are there any functional dependencies that might hold here?

No…

And yet it seems like there is some pattern / dependency…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

Given a set of movies and snacks…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

Given a set of movies and snacks…

Any movie / snack combination is possible!

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write $\{A\} \twoheadrightarrow \{B\}$ if for any tuples t_1, t_2 s.t. $t_1[A] = t_2[A]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write $\{A\} \twoheadrightarrow \{B\}$ if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write $\{A\} \twoheadrightarrow \{B\}$ if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write $\{A\} \twoheadrightarrow \{B\}$ if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$
- and $t_3[R\backslash B] = t_2[R\backslash B]$

Where R\B is "R minus B" i.e. the attributes of R not in B.

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_2	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
t_3	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_1	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

Note this also works!

Remember, an MVD holds over *a relation or an instance*, so defn. must hold for every applicable pair…

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_2	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
t_3	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_1	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

This expresses a sort of dependency (= data redundancy) that we *can't* express with FDs

**Actually, it expresses* *<u>conditional independence</u>* *(between film and snack given movie theatre)!*

Comments on MVDs

- *For AI nerds*: MVD is conditional independence in graphical models!

See the MVDs IPython notebook
for more examples!

[DB-WS07b.ipynb]

Summary

- Constraints allow one to reason about **redundancy** in the data

- Normal forms describe how to **remove** this redundancy by **decomposing** relations
  - Elegant—by representing data appropriately certain errors are essentially impossible
  - For FDs, **BCNF** is the normal form.

- **A tradeoff for insert performance: 3NF**