

CMPT 354

Normalization

Normalization

- Desirable database characteristics
- Database design, revisited
 - Normal Forms
 - First Normal Form
 - Second Normal Form
 - Third Normal Form
 - Boyce-Codd Normal Form
- Normal forms and functional dependencies
- ERD and normal forms
- Fourth Normal Form, multi-valued dependencies

Desirable DB Qualities

- Minimal repetition of data to avoid
 - Inefficiency in space usage and data processing
 - Potential loss of data integrity caused by data inconsistencies
- Lossless joins to ensure that
 - When two tables are joined the appropriate data, and only that data, is returned

Repetition and Inconsistent Data

- Record data about accounts and who owns them
- A customer can own many accounts, and accounts can be owned by many customers
 - Account = {customerID, accNumber, balance, type}

There are multiple rows for each account, making the table larger than it would be if account data was stored separately

accNumber	balance	type	customerID
...			
123	67,000	CHQ	11
123	67,000	CHQ	12
123	73,700	CHQ	13
...			

What happens if we want to add 10% to the balance of accounts owned by customer 13?

Why Lossless Joins?

- Consider the Customer table
 - Customer = {customerID, name, income, age}
- Let's decompose it (badly) into two tables
 - Customer_ID = {customerID, name}
 - Customer_data = {name, income, age}

customerID	name	income	birth
...			
11	Jane	43,000	1997
12	Jane	67,000	1975
...			

customerID	name
...	
11	Jane
12	Jane
...	

name	income	birth
...		
Jane	43,000	1997
Jane	67,000	1975
...		

What happens if we want to recreate the original table?

Lossy Joins

customerID	name
...	
11	Jane
12	Jane
...	



name	income	birth
...		
Jane	43,000	1997
Jane	67,000	1975
...		

customerID	name	income	birth
...			
11	Jane	43,000	1997
11	Jane	67,000	1975
12	Jane	43,000	1997
12	Jane	67,000	1975
...			

There are *more* records than the original table, but *less* information – we don't know how old the two customers are

This is known as a *lossy join*

Obtaining Desirable DB Characteristics

- Create an ERD that follows strict rules and convert it into a relational DB schema
 - Composite attributes are not allowed
 - Set valued attributes are not allowed
 - Relationship sets may only have descriptive attributes
- Perform a decomposition of the information required for the database and
 - Ensure that the resulting tables satisfy one or more *normal forms*

Normal Forms

A Strange Example

Meet Mr. Strange

- You are to create a small DB for *RannCo*®
 - To record information about the use of capital assets by departments within *RannCo*®
- You discuss the requirements for the database with the CIO, Mr. A. Strange
 - A transcript of this discussion follows:

What does RannCo manufacture, Mr. Strange?

Jet packs, hover cars, moving walkways, robot maids, really it's your basic 1950's science fiction technology company

Capital Assets

Right. So, I gather that the database is going to record information about capital assets.

Was that a question?

Yes, look this is supposed to be a transcript of a conversation, so you can't expect perfect grammar ... What is a capital asset anyway?

A capital asset is a long term asset of a company, like a car, or machinery, or hardware. That sort of thing. Capital assets help produce the goods that a company sells, as distinct from the raw materials to make those goods.

Department Data

OK. So what information will you need?

Hmm. Well Right. We need the name of each department, basically the point of this is so that we know what each department owns, you see. We'll want to keep track of each department's manager's name, STM, and phone extension. And of course all of the assets that a department is in charge of.

And what information do you need to record about the assets?

Let me see ...

Say What?

We need to record each asset's name and number, we give each asset a unique number, you see. We also want to keep track of the purchase cost and date, the type of the asset, its CCA class, and its depreciation rate.

CCA class!? Depreciation rate!? You're just making this stuff up aren't you? And what's the difference between an asset's name and type? You name the asset? Like the type is screwdriver and the name is Shirley?

Well, I guess some of those are good questions ...

Explanation

The type of an asset describes the class of asset, car, machinery and so on. Its name is a more detailed description, Toyota Echo, for example. The depreciation rate is an estimation for accounting purposes of how fast the asset loses value over time. For example, if the depreciation rate is 20% then we are estimating that we could resell the asset for only 80% of its original cost at the end of one year. CCA stands for Capital Cost Allowance, and is basically the same as the depreciation rate, except it is the rate recognized by the Canada Revenue Agency for tax purposes.

No Stickies Please

Got you. At least I think so. Anything else you can think of?

Hmm. One thing that you might need to know is that some assets are actually shared by two or more departments. In these cases we want to record what percentage of the time the asset is used by each department. But this doesn't happen very often so I guess we could just write this stuff down on yellow post-it notes and stick them on the assets.

That doesn't sound ideal, let's include that in the database. So this is what we've got:

Summary

Here is the information we need to record about capital assets

- Department name
- Manager number
- Manager name
- Manager extension
- Asset number
- Asset name
- Cost
- Purchase date
- Asset type
- Depreciation rate
- CCA
- Department use

I think that's everything — so now go earn your pay and design my new database!

Quick and Dirty

Great. That's pretty much all I need to know. By the way can two departments have the same name?

No.

OK, so we'll use the department name as the primary key and throw everything into one table.

Um, that just sounds like our Excel spreadsheet ...

So we can only have one row for each department.

Remember that one department can have many assets.

Right, good point, well we could give some records more columns for their extra assets, but then the records may be different lengths, so I guess the table wouldn't be in first normal form

First Normal Form

- Maintain all the data in one table but remove repeating groups
 - By adding *rows* for the repeated groups
 - The table has a compound primary key
 - *{department, assetNumber}*
- This results in *fixed length* records
 - Reduces disk fragmentation,
 - Complies with the relational model, and is
 - Required for most DBMSs

One Table

OK, so here is the first normal form database schema for our Department table:

Department = {departmentName, assetNumber, managerSIN, managerName, managerPhone, assetName, cost, purchaseDate, type, depreciation, cca, usage}

The table has a compound primary key consisting of department name and asset number. Sweet, huh?

So can I get paid now?

Well, I suppose that would work. But, whenever we want to add an asset to a department wouldn't we have to repeat all of the department information?

Yeah, but what can you do ...

Disadvantages of INF

- Introduces redundancy
 - Much of the data needs to be repeated
 - All of the department data (like the manager's name) is repeated for each asset that a department owns
- Insert anomalies
- Delete anomalies
- Update anomalies

Disadvantages of INF

- Introduces redundancy
- Insert anomalies
 - A department cannot be inserted if it doesn't have at least one asset
- Delete anomalies
 - Deleting the last asset of a department also deletes the department
- Update anomalies

Disadvantages of INF

- Introduces redundancy
- Insert anomalies
- Delete anomalies
- Update anomalies
 - Many records may have to be changed to change the value of one attribute
 - A change to the department manager's phone number has to be made for each asset the department owns

Second Normal Form

- In 1NF there are *non-key* attributes that depend on only *part* of the compound key, so
- Remove *partial key dependencies*
 - If a set of attributes only depends on part of the key separate the attributes into a new table
 - In each table each non-key attribute should be dependent only on the entire primary key
- Resulting in a *Second Normal Form* decomposition

Asset Types

I've changed the schemata from First Normal Form to Second Normal Form. Here is the new decomposition.

Department = {departmentName, managerSIN, managerName, managerPhone,}

Asset = {assetNumber, assetName, cost, purchaseDate, type, depreciation, cca}

Uses = {departmentName, assetNumber, usage}

The usage information depends on both department name and asset number, so stays in the Uses table,

Are you happy now?

That looks much better. Though, does it matter that the depreciation rate and the CCA rate are the same for all assets of the same type? We would never have two assets with the same type that have different rates.

Oh, #***§!!

Disadvantages of 2NF

- Second Normal Form only considers partial *key* dependencies
 - And ignores any *non-key* dependencies
- Therefore 2NF may still result in the same problems observed with 1NF, that is:
 - Redundancy
 - Insert, delete and update anomalies
 - Here, CCA rate and depreciation depend on asset type, but the type is not part of a primary key

Third Normal Form

- Remove any non-key dependencies
 - Remove the attributes with a non-key dependency from the table and
 - Create a new table containing those attributes and the attribute(s) that they depend on
 - The latter being the primary key of the new table
- *Third Normal Form* decomposition
 - All records are fixed length
 - There are no delete, insert or update anomalies
 - There is very little redundancy

By the way ...

I've changed the schemas from Second Normal Form to Third Normal Form.

Department = {departmentName, managerSIN, managerName, managerPhone}

Asset = {assetNumber, assetName, cost, purchaseDate, type}

AssetType = {type, depreciation, cca}

Uses = {departmentName, assetNumber, usage}

By the way, if something is in 3NF it is also in 2NF and in 1NF. So I guess we are done now. Time for beer!

That looks good. Ah, sorry, I've just realized that we may be missing some information.

Aaaargh!!

to be continued ...

Functional Dependencies

Dependencies

- A *superkey* is a set of attributes that uniquely identifies a record
 - Let R be a relation schema, subset K of R is a superkey if:
 - For all pairs t_1 and t_2 in $R \mid t_1 \neq t_2, t_1[K] \neq t_2[K]$
- Unlike a key, a *functional dependency* is a constraint on *any* set of attributes
 - Functional dependencies can assist us in achieving a desirable decomposition

Decomposition Goals

- Lossless join
 - The decomposition should not result in a *lossy join* if tables are re-combined
 - A lossy join is a join where the resulting table includes data that should not exist
- Dependency preservation
- No redundancy

Decomposition Goals

- Lossless join
- Dependency preservation
 - If a set of attributes depends on an attribute that dependency should be maintained in one table
 - To avoid having to join tables to test whether or not the data is correct
- No redundancy

Decomposition Goals

- Lossless join
- Dependency preservation
- No redundancy
 - A decomposition should contain a minimum amount of redundancy
 - This goal is less important than the preceding two goals

Functional Dependencies

- A functional dependency is an integrity constraint that generalizes the idea of a key
- If R is a relation and X and Y are sets of attributes of R then an instance r of R satisfies the $FD\ X \rightarrow Y$, if
 - For all tuples t_1, t_2 in r , if $t_1.X = t_2.X$ then $t_1.Y = t_2.Y$
 - e.g. $type \rightarrow cca, depreciation$
 - which states that cca and $depreciation$ must be the same for any two assets that have the same $type$
- A functional dependency is a statement about all possible legal instances of a relation

Terminology

- We can test relations to see if they are legal under a given set of functional dependencies
 - If a relation, R , is legal under a set of functional dependencies, F , then R satisfies F
- To specify constraints on a set of legal relations
 - If a schema, R , is to be constrained so that it satisfies a set of FDs, F , then F holds on R

FD Example: $AB \rightarrow C$

Whenever $\{A, B\}$ are the same then C must also be the same

If two tuples differ in *either* A or B fields then they may also differ in the C field

The dependency $AB \rightarrow C$ is violated by this tuple

$AB \rightarrow C$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d1
a1	b2	c2	d1
a2	b1	c3	d3
a1	b1	c2	d1

Which FDs are satisfied in the original relation:

$A \rightarrow C?$, $B \rightarrow C?$, $A \rightarrow D?$, $AB \rightarrow D?$

FDs and Keys

- A primary key constraint is a special case of a FD
- If there is a FD: $X \rightarrow Y$ on R , and Y is the set of all attributes of R , then X is a superkey
 - Note that X may not be a candidate key (or a primary key)
 - The definition of a FD does not require that the set of attributes is minimal

Reasoning About FDs

- A set of FDs, F , can be identified for a relation
 - By enquiring about the problem domain
 - In other words, by talking to people
- Given a set of FDs, additional FDs can usually be identified
 - The additional FDs are *implied* by F
- The set of *all* FDs implied by F is called the *closure* of F , denoted F^+
 - F^+ can be calculated by repeatedly applying *Armstrong's Axioms* to F

Calculating Closure and Cover

- As noted earlier a set of functional dependencies, F , can imply further FDs
 - The set of all FDs implied by F is known as the *closure* of F , or F^+
 - The *minimal set* of FDs from which F^+ can be calculated is known as the *canonical cover*
- We can use axioms, or rules of inference, to reason about FDs
 - Known as Armstrong's axioms

Armstrong's Axioms

■ Reflexivity

- If $X \supseteq Y$, then $X \rightarrow Y$

superset

remember that X and Y are sets of attributes

- That is, if X contains Y then $X \rightarrow Y$

■ Augmentation

- If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Note that $Z \rightarrow Z$

subset

- A functional dependency $X \rightarrow Y$ is referred to as *trivial* where $Y \subseteq X$

■ Transitivity

- If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Armstrong's Axioms

- Named after William W. Armstrong
 - Who earned his PhD from UBC in 1966
 - Dependency Structures of Data Base Relationships (1974)
- Armstrong's axioms are both *sound* and *complete*
 - They are sound because they do not generate any incorrect functional dependencies
 - They are complete because they allow F^+ to be generated from a given F
- Additional rules can be derived from Armstrong's axioms

Additional Rules

■ Union

- If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

- $X \rightarrow Y$ and $X \rightarrow Z$ – assumption

- $X \rightarrow XY$ – augmentation

if it helps, think of this as $XX \rightarrow XY$ but

- $XY \rightarrow YZ$ – augmentation

as X and Y are sets $XX = X$

- $X \rightarrow YZ$ – transitivity

■ Decomposition

- If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

■ Pseudotransitivity

- If $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$

Additional Rules

■ Union

- If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

■ Decomposition

- If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
 - $X \rightarrow YZ$ – assumption
 - $YZ \rightarrow Y, YZ \rightarrow Z$ – reflexivity
 - $X \rightarrow Y$ – transitivity
 - $X \rightarrow Z$ – transitivity

■ Pseudotransitivity

- If $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$

Additional Rules

■ Union

- If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

■ Decomposition

- If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

■ Pseudotransitivity

- If $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$
 - $X \rightarrow Y$ and $WY \rightarrow Z$ – assumption
 - $XW \rightarrow WY$ – augmentation
 - $XW \rightarrow Z$ – transitivity

Example

- Identify additional FDs in F^+
 - $R = (A, B, C, G, H, I)$ $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
 - $A \rightarrow H$
 - transitivity from $A \rightarrow B$ and $B \rightarrow H$
 - $AG \rightarrow I$
 - augmentation of $A \rightarrow C$ with G , to get $AG \rightarrow CG$
 - then transitivity with $CG \rightarrow I$
 - $CG \rightarrow HI$
 - augmentation of $CG \rightarrow I$ to get $CG \rightarrow CGI$,
 - then augmentation of $CG \rightarrow H$ to get $CGI \rightarrow HI$,
 - then transitivity

Generating F^+

$F^+ = F$

repeat

for each FD f in F^+

apply reflexivity and augmentation rules on f

add the resulting FDs to F^+

for each pair of FDs f_1, f_2 in F^+

if f_1 and f_2 can be combined using transitivity

then add the resulting FD to F^+

until F^+ does not change

There may be many functional dependencies

The left and right sides of a functional dependency are both subsets of R

Note - a set of size n has 2^n subsets

Closure of Attribute Sets

- It can be useful to determine what attributes are functionally dependent on a particular attribute set
 - To determine if the attribute set is a superkey
- Compute F^+ and take the union of the right side of each FD whose left side is the relevant attribute set
- This can also be performed without computing F^+

X^+ is the closure of a set of attributes, X , under F

```
result = X
while (there are changes to result)
    for each FD,  $Y \rightarrow Z$  in  $F$ 
        if  $Y \subseteq \text{result}$  then  $\text{result} = \text{result} \cup Z$ 
```

Attribute Set Closure Example

- What is the set of attributes, AG^+ ?
 - $R = (A, B, C, G, H, I)$ $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
 - $\text{result} = AG$
 - $\text{result} = ABCG$ ($A \rightarrow B, A \rightarrow C$)
 - $\text{result} = ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
 - $\text{result} = ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq AGBCH$)
- Is AG a superkey?
 - i.e. does $AG \rightarrow R$? i.e. is $AG^+ \supseteq R$?
- Is any subset of AG a superkey?
 - does $A \rightarrow R$? i.e. is $A^+ \supseteq R$?
 - does $G \rightarrow R$? i.e. is $G^+ \supseteq R$?

Uses of Attribute Closure

- There are several uses of attribute closure
- Testing for a superkey
 - If A^+ contains R then A is a superkey
- Testing functional dependencies
 - To check if a FD $X \rightarrow Y$ is in F^+ check to see if $Y \subseteq X^+$
 - i.e. compute X^+ by using attribute closure, and check to see if it contains Y
- Computing closure of F
 - For each $X \subseteq R$, find the closure X^+ , and for each $Y \subseteq X^+$, output a FD $X \rightarrow Y$

Canonical Cover

- Sets of FDs may contain redundant dependencies
- Individual dependencies may contain unnecessary attributes
- A *canonical cover* of F is a minimal set of FDs that is equivalent to F
 - With no redundant dependencies or parts of dependencies

Redundant Dependencies

- Dependencies can be derived from other FDs
 - e.g. $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C\}$
 - Because it can be obtained through transitivity
- Parts of a functional dependency may be redundant
 - e.g. on RHS: $\{A \rightarrow B, B \rightarrow C, A \rightarrow \textcolor{red}{C}D\}$ can be simplified
 - $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ since $A \rightarrow C$ can be derived
 - e.g. on LHS: $\{A \rightarrow B, B \rightarrow C, A\textcolor{red}{C} \rightarrow D\}$ can be simplified
 - $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ since $A \rightarrow C$ (through transitivity) so C is not necessary on the left hand side of the dependency
 - For example $sin \rightarrow name$ and $sin, name \rightarrow birthDate$
 - As *birth date* cannot be determined by *name* alone the inclusion of *name* on the left hand side is unnecessary

Extraneous Attributes

- Some FDs contain *extraneous* attributes
 - extraneous - not constituting a vital element or part
- Consider a FD $X \rightarrow Y$ in a set F of FDs
 - Attribute a is extraneous in X if $a \in X$ and if F implies FDs:
 $(F - \{X \rightarrow Y\}) \cup \{(X - a) \rightarrow Y\}$ use F to determine $(X - a) \rightarrow Y$
 - e.g. if $F = \{A \rightarrow D, D \rightarrow C, AB \rightarrow C\}$, B is extraneous in $AB \rightarrow C$
 - Because $\{A \rightarrow D, D \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$
 - Attribute a is extraneous in Y if $a \in Y$ and the set of FDs:
 $(F - \{X \rightarrow Y\}) \cup \{X \rightarrow (Y - a)\}$ logically implies F reproduce F using F'
 - e.g. if $F = \{A \rightarrow C, AB \rightarrow CD\}$, C is extraneous in $AB \rightarrow CD$
 - Because $AB \rightarrow C$ can be inferred even after deleting C

Testing for Extraneousness

- Consider a FD $X \rightarrow Y$ in a set F of FDs
- To test if attribute $a \in X$ is extraneous in X
 - compute $(\{X\} - a)^+$ using the FDs in F
 - check that $(\{X\} - a)^+$ contains Y
 - if it does, a is extraneous in X
- To test if attribute $a \in Y$ is extraneous in Y
 - compute X^+ using only the dependencies in:
$$F' = (F - \{X \rightarrow Y\}) \cup \{X \rightarrow (Y - a)\}$$
 - check that X^+ contains a
 - if it does, a is extraneous in Y

check that the attribute closure of the LHS still implies Y after removing a

check that the attribute closure of the LHS still includes a after removing it from the RHS in F'

Canonical Cover

- A canonical cover for F is a set of FDs, F_c , such that
 - F logically implies all dependencies in F_c , and
 - F_c logically implies all dependencies in F , and
 - No functional dependency in F_c contains an extraneous attribute, and
 - Each left side of a functional dependency in F_c is unique
 - Standardized format for cover

Computing Canonical Cover

- To compute a canonical cover for F :
 - repeat
 - Use the union rule to replace any dependencies in F
 $X \rightarrow Y$ and $X \rightarrow Z$ with $X \rightarrow YZ$
 - Find a functional dependency $X \rightarrow Y$ with an
extraneous attribute either in X or in Y
delete such extraneous attributes from $X \rightarrow Y$
 - until F does not change
- Note that the union rule may become applicable after the deletion of an extraneous attributes

Canonical Cover Example 1

- Compute the canonical cover, F_c , of R
 - $R = (A, B, C, G, H, I)$ $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- First use the union rule to combine dependencies
 - Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
 - The set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- Check for extraneous attributes
 - A is extraneous in $AB \rightarrow C$
 - To confirm this show that the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
 - Which it is since $B \rightarrow C$ already exists in F

Canonical Cover Example 2

- Compute the canonical, F_c , cover of R
 - $R = (A, B, C, G, H, I)$ $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- Continue to check for extraneous attributes
 - Set is now $\{A \rightarrow BC, B \rightarrow C\}$
 - C is extraneous in $A \rightarrow BC$
 - Show that $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other functional dependencies
 - Which it is, using transitivity on $A \rightarrow B$ and $B \rightarrow C$
 - The attribute closure of A may be used in more complex cases
- $F_c = \{A \rightarrow B, B \rightarrow C\}$

Decomposition

Boyce Codd Normal Form

- BCNF is a desirable normal form that can be found by identifying functional dependencies
 - BCNF eliminates all redundancy that can be discovered by studying a set of FDs
 - Although BCNF ignores multi-valued dependencies
 - Generally BCNF is preferable to 3NF
- 3NF allow some redundancy
 - Let's look at the definitions of each in terms of functional dependencies

BCNF Definition

- Boyce-Codd Normal Form is defined in terms of functional dependencies
- A relational schema, R , is in BCNF with respect to a set of FDs F if for *all* FDs in F^+ of the form
 - $X \rightarrow Y$ where $X \subseteq R$ and $Y \subseteq R$, \subseteq means *is a subset of*
- At least one of the following holds
 - $X \rightarrow Y$ is trivial (i.e., $Y \subseteq X$), or such as $\{name\} \rightarrow \{name\}$
 - X is a superkey for R

Third Normal Form Definition

- A relational schema, R , is in 3NF with respect to a set of FDs F if
- For all functional dependencies in F^+ of the form
 - $X \rightarrow Y$ where $X \subseteq R$ and $Y \subseteq R$,
- At least one of the following holds
 - $X \rightarrow Y$ is trivial (i.e., $Y \subseteq X$), or
 - X is a superkey for R , or
 - $(Y - X)$ is contained in a candidate key for R
- Note that the only difference between 3NF and BCNF is the last condition

BCNF Description

- In a BCNF decomposition the only FDs are those where the key of the table determines attributes
 - Except for trivial dependencies
- Each table represents either an entity set or a relationship set
 - Identified by the key, and
 - Described by the remaining attributes
- A database design is in BCNF if each table schema is in BCNF

BCNF Decomposition

- Assume that there is some schema R and a non-trivial dependency $X \rightarrow Y$ which causes a violation of BCNF
 - Because X is not a key for the entire table (R)
- R should be decomposed into
 - $(X \cup Y)$ and $(R - (Y - X))$
- e.g. $\text{Asset} = \{\text{assetNumber}, \text{assetName}, \text{cost}, \text{purchaseDate}, \text{type}, \text{depreciation}, \text{cca}\}$
 - Where $(\text{type} \rightarrow \text{type}, \text{cca}, \text{depreciation})$
 - Decompose asset into
 - $\{\text{type}, \text{cca}, \text{depreciation}\}$ and
 - $\{\text{Asset} - (\text{type}, \text{cca}, \text{depreciation} - \text{type})\}$, i.e.
 - $\{\text{assetNumber}, \text{assetName}, \text{cost}, \text{purchaseDate}, \text{type}\}$

Computing BCNF

```
result = R
done = false
compute  $F^+$ 
while (not done)
    if (there is a schema  $R_i$  in result not in BCNF)
         $X \rightarrow Y$  is a nontrivial FD that holds on  $R_i$ 
        such that  $X \rightarrow R_i^*$  is not in  $F^+$ , and  $X \cap Y = \emptyset$ 
        result = (result -  $R_i$ )  $\cup$  ( $R_i - Y$ )  $\cup$  ( $X, Y$ )
    else done = true
end while
```

*that is: X is not a key for the schema

when complete all R_i are in BCNF, and
the decomposition is a lossless-join

BCNF Decomposition Example

- $R = (A, B, C)$ $F = \{A \rightarrow B, B \rightarrow C\}$, Key = $\{A\}$
 - R is not in BCNF since $B \rightarrow C$ but B is not a superkey for R
- Decomposition
 - $R_1 = (B, C)$
 - $R_2 = (A, B)$
- Note that there may be more than one BCNF decomposition for the same data
 - Depending on the order in which the FDs are applied

3NF Decomposition

Let F_c be a canonical cover for F note that the algorithm looks at dependencies in the canonical cover
 $i = 0$

for each functional dependency $X \rightarrow Y$ in F_c
if none of the schemas R_j , $1 \leq j \leq i$ contains XY

$i = i + 1$

$R_i = XY$

if none of the schemas R_j , $1 \leq j \leq i$ contains a
candidate key for R

$i = i + 1$

$R_i = \text{any candidate key for } R$

return (R_1, R_2, \dots, R_i)

More Dependencies

We also need to record the SM of the employee who is responsible for monitoring an asset

That doesn't sound so bad, we can just add that to the asset table, as it just depends on the asset number

Unfortunately its not that simple. When an asset is jointly owned there is a responsible person for each of the owning departments.

And, of course, an employee can only belong to one department

So, in a sense, we could say that the department depends on the employee

I suppose we could say that ...

Transitive Dependencies

- Consider the *Uses* table
 - $Uses = \{\underline{assetNumber}, \underline{departmentName}, respPerson, usage\}$
 - The FDs that relate to this table are
 - $(dn, an \rightarrow us, rp)$
 - $(rp \rightarrow dn)$ and by pseudotransitivity
 - $(rp, an \rightarrow us)$
 - *Uses* is therefore not in BCNF
- The BCNF decomposition is:
 - $Uses = \{\underline{assetNumber}, \underline{respPerson}, usage\}$
 - $WorksIn = \{\underline{respPerson}, departmentName\}$
 - However *departmentName* also depends on *respPerson*

Transitive Dependencies

- A BCNF decomposition would create a new table for each functional dependency
 - $Uses = \{\underline{assetNumber}, \underline{respPerson}, usage\}$
 - In this case *usage* depends on the compound key of *departmentName* and *assetNumber*, as does *respPerson*
 - $WorksIn = \{\underline{respPerson}, departmentName\}$
 - However *departmentName* also depends on *respPerson*
- The 3NF decomposition would ignore the transitive dependency*
 - *since, in $rp \rightarrow dn$, *dn* is part of a candidate key for *Uses*
 - $Uses = \{\underline{departmentName}, \underline{assetNumber}, usage, respPerson\}$
- In this case the 3NF decomposition is preferred

BCNF Decomposition of Uses

Uses = {*dName*, *assetNum*, *usage*, *respPerson*}

$F = (dp, an \rightarrow us, rp), (rp \rightarrow dp), (rp, an \rightarrow us)$

which gives the following BCNF decomposition, sample data included ...

an	rp	use
11	Zak	30
11	Ann	70
12	Sue	40
12	Bob	60
13	Zak	37
13	Ann	63

rp	dn
Zak	1
Sue	1
Bob	2
Ann	2

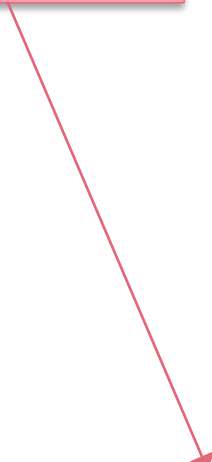
{*respPerson*, *dName*}

{*assetNumber*, *respPerson*, *usage*}

BCNF Decomposition of Uses

Now let's add a record that states that Joe is responsible for asset number 11, and change the use percentage so that it all adds up correctly

Why is this a problem?



an	rp	use
11	Zak	20
11	Ann	70
12	Sue	40
12	Bob	60
13	Zak	37
13	Ann	63
11	Joe	10

rp	dn
Zak	1
Sue	1
Bob	2
Ann	2
Joe	1

Because it violates $dn, an \rightarrow rp$

Motivation for 3NF

- If a relation is in BCNF it must also be in 3NF
- The third condition of 3NF is a minimal relaxation of BCNF to ensure dependency preservation
 - The third condition is when the right hand side of a FD is part of a candidate key for the relation
 - $Uses = \{departmentName, assetNumber, usage, respPerson\}$
 - In this relation *departmentName* is part of a key for the relation so the decomposition is in 3NF with respect to $(rp \rightarrow dn)$
 - In this particular case the 3NF decomposition is preferable to the BCNF decomposition

3NF Decomposition of Uses

$Uses = \{departmentName, assetNumber, usage, respPerson\}$

$F = (dp, an \rightarrow us, rp), (rp \rightarrow dp), (rp, an \rightarrow us)$

gives a 3NF decomposition, where rp is part of a compound key

an	dn	rp	use
11	1	Zak	30
12	1	Sue	40
13	1	Zak	37
11	2	Ann	60
12	2	Bob	37
13	2	Ann	63

Attempting to insert Joe, working in dept 1, as the responsible person for asset #11, will fail, as it violates the primary key which comes from the FD

$(dp, an \rightarrow us, rp)$

But notice the redundant data

$\{\underline{assetNumber}, \underline{departmentName}, respPerson, usage\}$

3NF vs. BCNF

- A BCNF decomposition removes redundancy
 - Except for multi-valued dependencies
 - But does not guarantee a dependency preserving decomposition
 - In the example the BCNF decomposition did not preserve the dependency $(an, dp \rightarrow rp, us)$
- A dependency preserving 3NF decomposition can always be found
 - But 3NF allows some repetition
 - In the example the 3NF decomposition repeated the department for each responsible person

Lossless Join Decompositions

- Consider decomposing a relation schema, R , with a set of FDs, F , into two relations, X and Y
 - If the original relation can be recovered by joining X and Y it is a lossless-join decomposition with respect to F
- A decomposition is only lossless if and only if F^+ contains either $X \cap Y \rightarrow X$ or $X \cap Y \rightarrow Y$
 - i.e. the attributes common to X and Y must contain a key for either X or Y

Dependency-Preserving Decompositions

- Consider decomposing a relation schema, R , with a set of FDs, F , into two relations, X and Y
 - The projection of F , F_X , on X is the set of FDs in F^+ that only involve attributes of X
 - A FD $A \rightarrow B$ is only in F_X if all the attributes of A and B are in X
- A decomposition of R is dependency preserving if $(F_X \cup F_Y)^+ = F^+$
 - If that is the case, then only the dependencies of F_X and F_Y need to be enforced
 - As all the FDs in F^+ will be satisfied

Schema Refinement

- In practice a DB designer usually uses an ER design (or something similar) for an initial design
- Ideally a good ER design should lead to a collection of tables with no redundancy problems
 - ER design is complex and subjective, and
 - Certain constraints cannot be expressed in ER diagrams
- A decomposition derived from an ER diagram may need further refinement
 - To ensure that it is in 3NF or BCNF

Constraints on Entity Sets

- When creating ERDs it is easy to miss dependencies within an entity set
 - e.g. *type* → *cca, depreciation*
 - e.g. *level* → *salary*
- This problem becomes particularly relevant in practice when designing large schemas
 - Many real-world DBs may have hundreds of tables
- A correct ERD would create additional entity sets for the dependencies
 - But, a knowledge of FDs and DB design is required to recognize this

Identifying Attributes

- Identifying FDs can make it easier to associate attributes with the correct entity set
 - In some cases attributes may be associated with the wrong entity set
- For example, employee parking lots
 - Assume that each employee is assigned a parking lot where they have to park
 - It seems reasonable to make *lot* an attribute of Employee
 - However, the employees are assigned the lots based on the department that they work in
 - Therefore *dept* \rightarrow *lot*, and the *lot* attribute should be an attribute of department rather than employee

Summary

- Identifying functional dependencies can assist a DB designer in producing a good schema
 - That is in BCNF, or 3NF, and is
 - A lossless-join decomposition, dependency preserving with minimal redundancy
- Functional dependencies can be used in conjunction with ER diagrams to refine the schema
 - FDs are particularly useful in cases where there is difficulty in deciding how some information should be modeled

How Good is BCNF?

- It is still possible to have a schema in BCNF (or 3NF) that is not sufficiently normalized
 - For example: $\text{Classes} = \{\text{course}, \text{teacher}, \text{book}\}$
 - A teacher, t , is qualified to teach course c , which requires textbook b
 - The table is supposed to list the set of teachers competent to teach a course, and
 - The set of books which are required for that course
- An instance of this schema follows ...

Courses

course	teacher	book
necromancy	Amazo	Bones
necromancy	Amazo	Tombs
necromancy	Samael	Bones
necromancy	Samael	Tombs
woodworking	Larch	Planes
woodworking	Larch	Trees
woodworking	Larch	Tools

Only trivial FDs hold, so the table is in BCNF

Whenever a new teacher is added, one row must be inserted for each book

And, multiple teacher rows are added when a book is added

Leading to redundancy

This occurs because the books are *independent* of the teachers

A *multivalued* dependency

Multivalued Dependencies

The following decomposition avoids the redundancy

course	teacher
necromancy	Amazo
necromancy	Samael
woodworking	Larch

course	book
necromancy	Bones
necromancy	Tombs
woodworking	Planes
woodworking	Trees
woodworking	Tools

A multi-valued dependency $X \twoheadrightarrow Y$ holds over R , if for every instance of R , each X value is associated with a set of Y values, and this set is independent of the values in other attributes

Other Normal Forms

- Further normal forms exist which deal with issues not covered by functional dependencies
- *Fourth Normal Form* deals with multi-valued dependencies
 - There is a 4NF decomposition algorithm similar to the BCNF decomposition algorithm
 - And a set of rules for inferring additional MVDs
- *Fifth Normal Form* addresses more complex (and rarer) situations where 4NF is not sufficient