CPSC 304
Introduction to Database Systems

Schema Refinement and Normal Forms

Textbook Reference
Database Management Systems
Reading: Chapter 19. Concentrate on sections19.1-19.6 (except 19.5.2)

Databases – the continuing saga

- We've learned that databases are wonderful things
- We've learned how to create a *conceptual design* using ER diagrams
- We've learned how to create a *logical design* by turning the ER diagrams into a relational schema including minimizing the data and relations created
- Now we need to *refine* that schema to reduce duplication of information

Learning Goals

- Debate the pros and cons of redundancy in a database.
- Provide examples of update, insertion, and deletion anomalies.
- Given a set of tables and a set of functional dependencies over them, determine all the keys for the tables.
- Show that a table is/isn't in 3NF or BCNF.
- Prove/disprove that a given table decomposition is a lossless join decomposition. Justify why lossless join decompositions are preferred decompositions.
- Decompose a table into a set of tables that are in 3NF, or BCNF.

Imagine that we've created a perfectly good entity for mailing addresses at UBC:

Instance of the schema on the previous slide

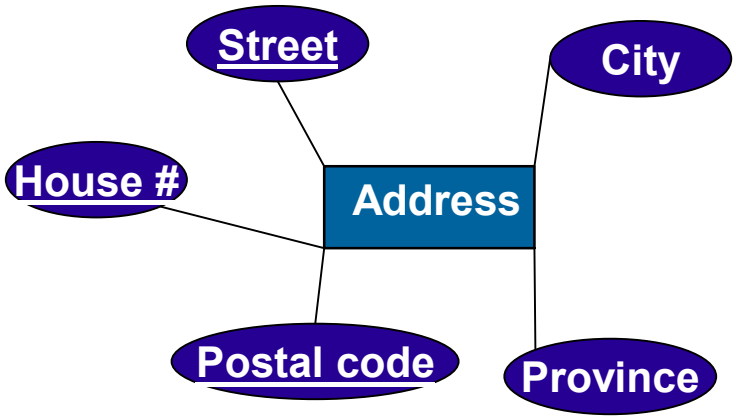
Name	Department	Mailing Location
Jessica Wong	Computer Science	201-2366 Main Mall
Rachel Pottinger	Computer Science	201-2366 Main Mall
Joel Friedman	Computer Science	201-2366 Main Mall
Joel Friedman	Math	121-1984 Mathematics Rd
Mark MacLean	Math	121-1984 Mathematics Rd

What are some problems that might happen with this design?
Based solely on intuition, what might be a better design?

Okay, that's bad. But how do I know for sure if departments have only one address?

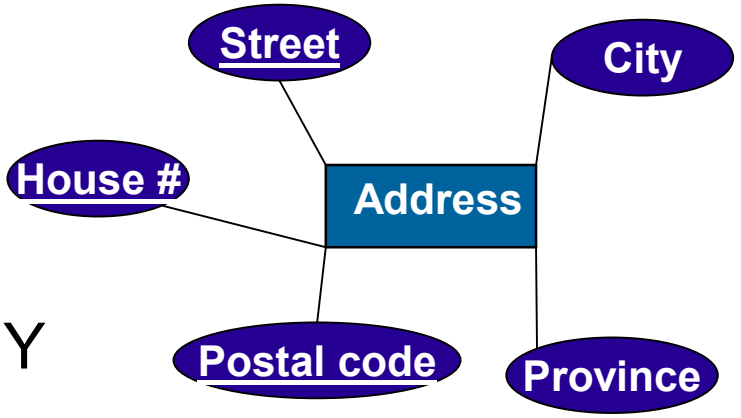
- Databases allow you to say that one attribute determines another through a *functional dependency*.
- So if Department determines Address but not Name, we say that there's a functional dependency from Department to Address. But Department is NOT a key.

- Another example:
  Address(House#, Street, City, Province, PostalCode)



Functional Dependencies (FDs) – technically speaking

- A **functional dependency** **X→Y**
  (where X & Y are sets of attributes)
  holds if for every legal instance,
  for all tuples *t1, t2* :
  if *t1*.X = *t2*.X *then* *t1*.Y = *t2*.Y



- Example:
  PostalCode → City, Province if:
  for each possible t1, t2, if t1.PostalCode = t2.PostalCode then
  t1.{City,Province} = t2.{City,Province})
- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
- Also can be read as X *determines* Y

Let's see some more instances

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A 2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1

Remember: Key is House #, Street, Postal Code

Does City determine Province?

A. Yes

B. No

C. Can't tell

Let's see some more instances

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A 2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1
800	Benvenuto Ave	Victoria	BC	V8M 1J8
165	Duckworth Street	Victoria	NF	A1B 4R5

Which functional dependencies do we care about?

- A FD is a statement about *all* allowable instances.
  - Must be identified by application semantics
  - Given some instance *r1* of R, we can check if r1 violates some FD *f*, but we cannot tell if *f* holds over R!
- There are boring, *trivial* cases:
  - e.g. PostalCode, House# → PostalCode
- We'll concentrate on the non-boring ones and on cases where there's a single attribute on the RHS[1]: (e.g., PostalCode → Province and PostalCode → City)

1. RHS = Right Hand Side. LHS = Left Hand Side

Clicker question: Possible FDs

Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

What FDs cannot be true given the instance above?

A. B → C
B. B → D
C. D → B
D. All of the above can be true
E. None of the above can be true

Clicker question: Possible FDs

Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

What FDs cannot be true given the instance above?

A. B → C — fine

B. B → D — not possible (2nd and 4th), so correct answer

C. D → B — fine

D. All of the above can be true

E. None of the above can be true

Naming the Evils of Redundancy

Let's consider Postal Code → City, Province

House #	Street	City	Province	Postal Code
101	Main Street	Vancouver	BC	V6A 2S5
103	Main Street	Vancouver	BC	V6A 2S5
101	Cambie Street	Vancouver	BC	V6B 4R3
103	Cambie Street	Vancouver	BC	V6B 4R3
101	Main Street	Delta	BC	V4C 2N1
103	Main Street	Delta	BC	V4C 2N1

- Update anomaly: Need to change more than 1 thing to stay consistent. Can we change Delta's province?
- Insertion anomaly: attributes cannot be inserted without the presence of other attributes. What if we want to insert that V6T 1Z4 is in Vancouver?
- Deletion anomaly: attributes are lost because of deletion of other attributes. If we delete all addresses with V6A 2S5, we lose that V6A 2S5 is in Vancouver!

Once more, with feeling

House #	Street	Postal Code
101	Main Street	V6A 2S5
103	Main Street	V6A 2S5
101	Cambie Street	V6B 4R3
103	Cambie Street	V6B 4R3
101	Main Street	V4C 2N1
103	Main Street	V4C 2N1

City	Province	Postal Code
Vancouver	BC	V6A 2S5
Vancouver	BC	V6B 4R3
Delta	BC	V4C 2N1

- Did we lose anything?
- Are our problems fixed?

As it turns out, this is a pretty optimal way to split the table up, though there's still a little redundancy (e.g., Vancouver being in BC is in the table twice). Let's see why…

What do we need to know to split apart addresses without losing information?

- FDs tell us when we're storing redundant information
- Reducing redundancy helps eliminate anomalies and save storage space
- We'd like to split apart tables without losing information
- But first, we need to know both what FDs are *explicit* (given) and what FDs are *implicit* (can be derived)
- Among other things, this can help us derive additional keys from the given keys (spare keys are handy in databases, just like in real life – we'll see why shortly)

The Key is the Thing

- As a reminder, a key is a *minimal* set of attributes that uniquely identify a relation
  - i.e., a key is a minimal set of attributes that *functionally determines* all the attributes
  - e.g., House#, Street, PostalCode is a key for Address
- A superkey for a relation uniquely identifies the relation, but does not have to be minimal
  - i.e.,: key $\subseteq$ superkey
  - E.g.,:
    - House#, Street, PostalCode is a key and a superkey
    - House#, Street, PostalCode, city is a superkey, but not a key

Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  B → C
  C → B
  D → A,B,C

Which of the following is a minimal key for the above relation?

A. B
B. C
C. B,D
D. All of the above
E. None of the above

Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  B→C
  C→B
  D→A,B,C

Which of the following is a minimal key for the above relation?

A. B — Does not determine all
B. C — Does not determine all
C. B,D — Not minimal
D. All of the above
E. None of the above — The right answer

Clicker question: Possible Superkeys

- Assume the same relation R(A,B,C,D) and FDs
  B → C
  C → B
  D → A,B,C

Which of the following are not a superkey for the above relation?

A. D
B. B,D
C. B,C,D
D. All are superkeys
E. None are superkeys

Clicker question: Possible Superkeys

- Assume the same relation R(A,B,C,D) and FDs
  B→C
  C→B
  D→A,B,C

Which of the following are not a superkey for the above relation?

A. D
B. B,D
C. B,C,D
D. All are superkeys
E. None are superkeys

D is a key. Therefore, all of the answers are superkeys

Let's summarize:

- In a functional dependency, a set of attributes determines other attributes, e.g., AB→C, means A and B together determine C
- A trivial FD determines what you already have, eg., AB→B
- A key is a minimal set of attributes determining the rest of the attributes of a relation
  For example, R(House #, Street, City, Province, Postal Code)
- A superkey is a set of attributes determining the rest of the attributes in the relation, but does NOT have to be minimal (e.g., the key above, or adding in either of City and Province)
- Given a set of (explicit) functional dependencies, we can determine others…

Deriving Additional FDs: the basics

- Given some FDs, we can often infer additional FDs:
  - *studentid* → *city*, *city* → *acode* implies *studentid* → *acode*
- An FD *fd* is ***implied by*** a set of FDs *F* if *fd* holds whenever all FDs in *F* hold.
  - ***closure of F*** : the set of all FDs implied by *F*.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $Y \subseteq X$, then $X \rightarrow Y$
    e.g., city,major→city
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
    e.g., if sid→city, then sid,major → city,major
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
    sid → *city*, *city* → *areacode* implies *sid* → *areacode*
- These are *sound* and *complete* inference rules for FDs.

Deriving Additional FDs: the extended dance remix

- A couple of additional rules (that follow from axioms):
  - ***Union***: If X→Y and X→Z, then X→Y Z
    e.g., if sid→acode and sid→city, then sid→acode,city
  - ***Decomposition***: If X→Y Z, then X→Y and X→Z
    e.g., if sid→acode,city then sid→acode, and sid→city

Example: Derive union rule from axioms (Reflexivity, Augmentation, & Transitivity)

Union: If X→Y and X→Z, then X→Y Z
e.g., if sid→acode and sid→city, then sid→acode,city

1. X→Y Given
2. X→Z Given
3. X→XY 1, augmentation
4. XY→ZY 2, augmentation
5. X→ZY 3, 4, transitivity

Ex: Derive Decomposition from axioms (Reflexivity, Augmentation, & Transitivity)

Decomposition: If X→Y Z, then X→Y and X→Z
e.g., if sid→acode,city then sid→acode, and sid→city

1. X→YZ Given
2. YZ→Y Reflexivity
3. YZ→Z Reflexivity
4. X→Y 1, 2, transitivity
5. X→Z 1, 3, transitivity

All 5 Rules

- ***Reflexivity***: If $Y \subseteq X$, then $X \rightarrow Y$
  e.g., city,major→city
- ***Augmentation***: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  e.g., if sid→city, then sid,major → city,major
- ***Transitivity***: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
  sid → city, *city → areacode* implies *sid → areacode*
- ***Union***: If X→Y and X→Z, then X→Y Z
  e.g., if sid→acode and sid→city, then sid→acode,city
- ***Decomposition***: If X→Y Z, then X→Y and X→Z
  e.g., if sid→acode,city then sid→acode, and sid→city

Example: Supplier-Part DB

- Suppliers supply parts to projects.
  - SupplierPart(sname,city,status,p#,pname,qty)
    - supplier attributes: sname, city, status
    - part attributes: p#, pname
    - supplier-part attributes: qty
- Functional dependencies:
  - fd1: sname → city
  - fd2: city → status
  - fd3: p# → pname
  - fd4: sname, p# → qty
- How can we show that (sname, p#) is a key?

Supplier-Part Key: Part 1: Determining all attributes

Show that (sname, p#) is a superkey of SupplierPart(sname,city,status,p#,pname,qty)
Proof has two parts:

a. Show: sname, p# is a (super)key

1.	sname, p# → sname, p#	reflex
2.	sname → city	fd1
3.	sname → status	2, fd2, trans
4.	sname,p# → city, p#	2, aug
5.	sname,p# → status, p#	3, aug
6.	sname,p# → sname, p#, status	1, 5, union
7.	sname,p# → sname, p#, status, city	4, 6, union
8.	sname,p# → sname, p#, status, city, qty	7, fd4, union
9.	sname,p# → sname, p#, status, city, qty, pname	8, fd3, union

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

Supplier-Part Key: Part 2: Minimalness

b. Show: (sname, p#) is a *minimal* key of SupplierPart(sname,city,status, p#,pname,qty)

1. p# does not appear on the RHS of any FD therefore except for p# itself, nothing else determines p#
3. specifically, sname → p# does not hold
4. therefore, sname is not a key
5. similarly, p# is not a key

fd1:	sname → city
fd2:	city → status
fd3:	p# → pname
fd4:	sname, p# → qty

Specifically, for combinations of sname and p#:

- sname,p# →sname, p#, city, status, pname, qty
- sname → sname, city, status
- p# → p#, pname

Do you, by any chance, have anything less painful?

- Scared you're going to mess up? *Closure* is a fool-proof method of checking FDs and finding implicit FDs.
- *Closure of* a set of attributes *X* is denoted X^+
- X^+ includes all attributes of the relation IFF X is a (super)key
- Algorithm for finding Closure of X:
  Let Closure = X
  Until Closure doesn't change do
  if $a_1, \ldots, a_n \rightarrow C$ is a FD and $\{a_1, \ldots, a_n\} \in$ Closure
  Then add C to Closure

fd1:	sname → city
fd2:	city → status
fd3:	p# → pname
fd4:	sname, p# → qty

SupplierPart(sname,city,status,p#,pname,qty)

Ex: $sname,p\#^+$ =

$sname^+$ =

$p\#^+$ =

So seeing if a set of attributes is a key means checking to see if it's closure is all the attributes – pretty simple

FD exercise

For relation R(ABCDE) with FDs

AB → C

B → E

DE → C

Find the closure of AB, B, and DE

$AB^{+} = \{ A, B, C, E \}$

$B^{+} = \{ B, E \}$

$DE^{+} = \{ D, E, C \}$

Let's take a minute to appreciate what closures give us

- Consider relation R(A,B,C,D) with FDs
  AB→C
  C→D
- Closures give us:
  AB^+ = {A,B,C,D}
  C^+ = {C,D}
- This tells us all the FDs, both explicit and implied:
  AB→A
  AB→B
  AB→C
  AB→D
  C→C
  C→D
- But how can we be sure that we've found all the keys (I promise you we will need this later) without resorting to Armstrong's axioms?