Introduction to Database Systems CSE 414

Lecture 20: Design Theory

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
- Unit 3: Non-relational data
- Unit 4: RDMBS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
  - E/R diagrams
  - Schema normalization
- Unit 7: Transactions

Database Design Process

Conceptual Model:

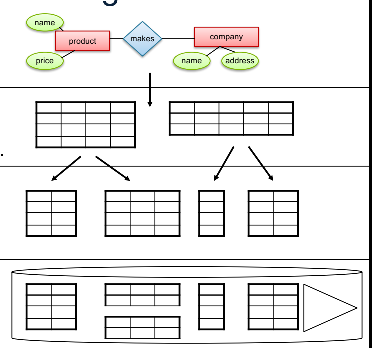
Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

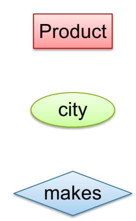
Physical storage details

Physical Schema



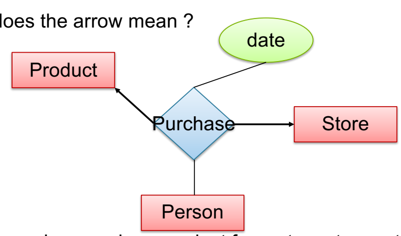
Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object
- Attribute
- Relationship



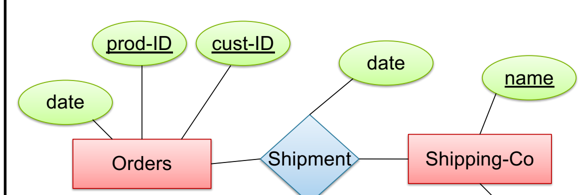
Arrows in Multiway Relationships

Q: What does the arrow mean ?



A: Any person buys a given product from at most one store AND every store sells to every person at most one product

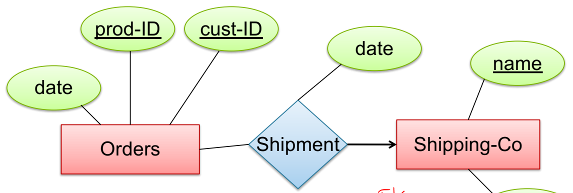
N-N Relationships to Relations



Orders(prod-ID,cust-ID, date)
Shipment(prod-ID,cust-ID, name, date)
Shipping-Co(name, address)

prod-ID	cust-ID	name	date
Gizmo55	Joe12	UPS	4/10/2011
Gizmo55	Joe12	FEDEX	4/9/2011

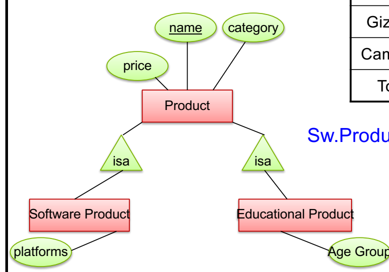
N-1 Relationships to Relations



Orders(<u>prod-ID,cust-ID,</u> date1, name, date2)
Shipping-Co(<u>name</u>, address)

Remember: no separate relations for many-one relationship

Subclasses to Relations



Product

<u>Name</u>	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

<u>Name</u>	platforms
Gizmo	unix

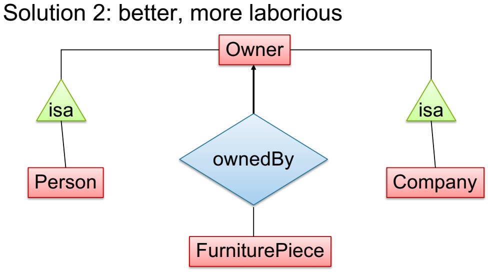
Ed.Product

<u>Name</u>	Age Group
Gizmo	toddler
Toy	retired

Other ways to convert are possible

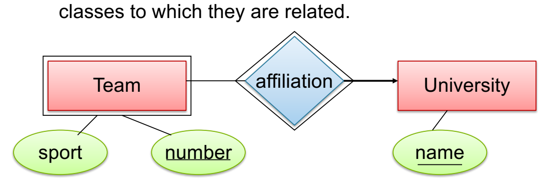
Modeling Union Types with Subclasses

Solution 2: better, more laborious



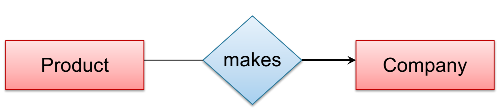
Weak Entity Sets

Entity sets are weak as their key comes from other classes to which they are related.

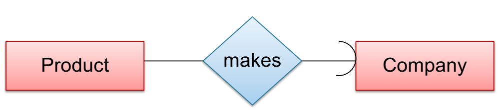


Team(sport, <u>number, universityName</u>)
University(<u>name</u>)

Referential Integrity Constraints

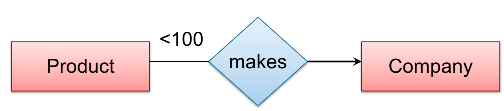


Each product made by at most one company.
Some products made by no company



Each product made by *<u>exactly</u>* one company.

Other Constraints



Q: What does this mean ?
A: A Company entity cannot be connected by relationship to more than 99 Product entities

Constraints in SQL

Constraints in SQL:
- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

simplest

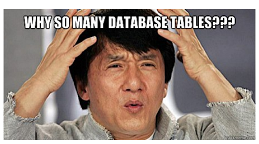
Most complex

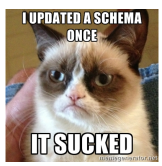
- The more complex the constraint, the harder it is to check and to enforce

What happens when data changes?

- SQL has three policies for maintaining referential integrity:
- NO ACTION reject violating modifications (default)
- CASCADE after delete/update do delete/update
- SET NULL set foreign-key field to NULL
- SET DEFAULT set foreign-key field to default value
  - need to be declared with column, e.g., `CREATE TABLE Product (pid INT DEFAULT 42)`

What makes good schemas?





Relational Schema Design

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

One person may have multiple phones, but lives in only one city

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?

Relational Schema Design

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

Anomalies:
- Redundancy = repeat data
- Update anomalies = what if Fred moves to "Bellevue"?
- Deletion anomalies = what if Joe deletes his phone number?

Relation Decomposition

Break the relation into two:

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

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

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

Anomalies have gone:
- No more repeated data
- Easy to move Fred to "Bellevue" (how ?)
- Easy to delete all Joe's phone numbers (how ?)

Relational Schema Design (or Logical Design)

How do we do this systematically?

- Start with some relational schema
- Find out its ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally: ($A_1 \ldots A_n$ **determines** $B_1 .. B_m$)

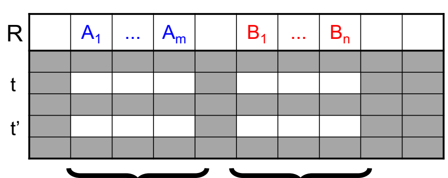
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

$\forall t, t' \in R$,

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



if t, t' agree here then t, t' agree here

Example

An FD holds, or does not hold on an instance:

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

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

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

Position → Phone

Example

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

But not Phone → Position

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Do all the FDs hold on this instance?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	49
Gizmo	Stationary	Green	Office-supp.	59

What about this one ?

Buzzwords

- FD **holds** or **does not hold** on an instance
- If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**
- If we say that R satisfies an FD, we are **stating a constraint on R**

Why bother with FDs?

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

Anomalies:
- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number?

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds: name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$, s.t. $A_1, \ldots, A_n \rightarrow B$

Example:
1. name → color
2. category → department
3. color, category → price

Closures:

$name^+ = \{name, color\}$

$\{name, category\}^+ = \{name, category, color, department, price\}$

$color^+ = \{color\}$

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}$^+$ =
{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

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

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}$^+$ X = {A, B, }

Compute {A, F}$^+$ X = {A, F, }

Example

In class:

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

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}$^+$ X = {A, B, C, D, E }

Compute {A, F}$^+$ X = {A, F, }

Example

In class:

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

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}$^+$ X = {A, B, C, D, E }

Compute {A, F}$^+$ X = {A, F, B, C, D, E }

Example

In class:

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

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}$^+$ X = {A, B, C, D, E }

Compute {A, F}$^+$ X = {A, F, B, C, D, E }

What is the key of R?

Practice at Home

Find all FD's implied by:

A, B → C
A, D → B
B → D

Step 1: Compute X$^+$, for every X:

A$^+$ = A, B$^+$ = BD, C$^+$ = C, D$^+$ = D
AB$^+$ =ABCD, AC$^+$=AC, AD$^+$=ABCD,
BC$^+$=BCD, BD$^+$=BD, CD$^+$=CD
ABC$^+$ = ABD$^+$ = ACD$^+$ = ABCD (no need to compute– why ?)
BCD$^+$ = BCD, ABCD$^+$ = ABCD

Step 2: Enumerate all FD's X → Y, s.t. Y ⊆ X$^+$ and X ∩ Y = ∅ :

AB → CD, AD→BC, ABC → D, ABD → C, ACD → B

Keys

R(A, ..., An, B)

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$
- A **key** is a minimal superkey
  - A superkey and for which no subset is a superkey

Computing (Super)Keys

- For all sets X, compute X^+
- If X^+ = [all attributes], then X is a superkey
- Try reducing to the minimal X's to get the key

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more distinct keys

Key or Keys ?

Can we have more than one key ?

Given R(A,B,C) define FD's s.t. there are two or more distinct keys

A → B
B → C
C → A

or

AB → C
BC → A

or

A → BC
B → AC

what are the keys here ?

Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key
- X → A is not OK otherwise
  - Need to decompose the table, but how?

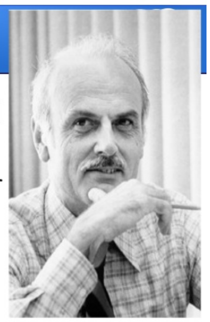
Boyce-Codd Normal Form

Boyce-Codd Normal Form

Dr. Raymond F. Boyce

Edgar Frank "Ted" Codd

"A Relational Model of Data for Large Shared Data Banks"



Boyce-Codd Normal Form

There are no "bad" FDs:

Definition. A relation R is in BCNF if:

Whenever X→ B is a non-trivial dependency, then X is a superkey.

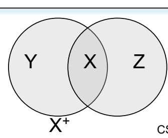
Equivalently:

Definition. A relation R is in BCNF if:

∀ X, either $X^+ = X$ or $X^+ =$ [all attributes]

BCNF Decomposition Algorithm

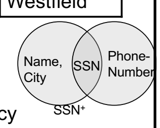
```
Normalize(R)
  find X s.t.: X ≠ X+ and X+ ≠ [all attributes]
  if (not found) then "R is in BCNF"
  let Y = X+ - X;    Z = [all attributes] - X+
  decompose R into R1(X ∪ Y) and R2(X ∪ Z)
  Normalize(R1); Normalize(R2);
```



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



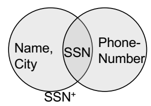
The only key is: {SSN, PhoneNumber}
Hence SSN → Name, City is a "bad" dependency

In other words:
SSN+ = SSN, Name, City and is neither SSN nor All Attributes

Example BCNF Decomposition

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

SSN → Name, City



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

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?