Renaming

- Does not change the relational instance
- Changes the relational schema only
- Notation: $\rho_{S(B1,\ldots,Bn)}(R)$
- Input schema: $R(A1, \ldots, An)$
- Output schema: $S(B1, \ldots, Bn)$
- Example:

$$\rho_{LastName,\ SocSocNo}\ \underbrace{(Employee)}_{}$$

Employee(__ , __)

1, context change

drinker(name, addr)

WhoLikesBud(~~name~~, year)

drinker

2. Name clash.

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$$\rho_{LastName,\ SocSocNo}\ (\mathbf{Employee})$$

LastName	SocSocNo
John	999999999
Tony	777777777

Behind the Scene: Why Codd invented rel. algebra?

- Codd proposed R.A. right up front-- in the 1970 CACM paper on relational model.
- As a query language? No.
- For defining "data health" by derivability.

> In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

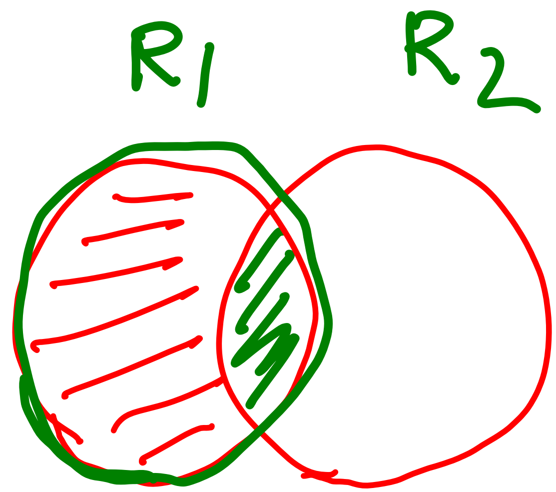
> Suppose θ is a collection of operations on relations and each operation has the property that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation R is *θ-derivable* from a set S of relations if there exists a sequence of operations from the collection θ which, for all time, yields R from members of S.

Set Operations: Intersection

$\cup$, $-$

$\cap$

- Difference: all tuples both in R1 and in R2
- Notation: $R_1 \cap R_2$
- Input: R_1, R_2 must have the same schema
- Output: $R_1 \cap R_2$ has the same schema as R_1, R_2
- Example
  - UnionizedEmployees - RetiredEmployees
- Intersection is derived:
  - $R_1 \cap R_2 = R_1 - (R_1 - R_2)$



Joins

- Theta join
- Natural join
- Equi-join
- Semi-join
- Inner join
- Outer join

Advanced

Theta Join

- A join that involves a predicate
- Notation: $R_1 \bowtie_\theta R_2$, where θ is a condition
- Input schemas: $R_1(A_1, \dots, A_n), R_2(B_1, \dots, B_m)$
  - $\{A_1, \dots, A_n\} \cap \{B_1, \dots, B_m\} = \emptyset$
- Output schema: $S(A_1, \dots, A_n, B_1, \dots, B_m)$
- Derived operator:

$$R_1 \bowtie_\theta R_2 = \sigma_\theta \, (R_1 \times R_2)$$

Example

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Natural Join

- Notation: $R_1 \bowtie R_2$
- Input Schema: $R_1(A_1, \ldots, A_n), R_2(B_1, \ldots, B_m)$
- Output Schema: $S(C_1, \ldots, C_p)$
  - $\{C_1, \ldots, C_p\} = \{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$
- Meaning: combine all pairs of tuples in R_1 and R_2 that agree on the attributes:
  - $\{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\}$, the join attributes.
- Derived operator:
  - Q: How to derive it in terms of $R_1 \times R_2$ and σ?

- Example: **Employee** $\bowtie$ **Dependents**

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ Dependents =

$\Pi_{\text{Name, SSN, Dname}}(\sigma_{\text{SSN=SSN2}}(\text{Employee x } \rho_{\text{SSN2, Dname}}(\text{Dependents}))$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Natural Join

- Given $R(A, B, C, D), S(A, C, E)$, what is the schema of $R \bowtie S$?
- Given $R(A, B, C), S(D, E)$, what is the schema of $R \bowtie S$?
- Given $R(A, B), S(A, B)$, what is the schema of $R \bowtie S$?

Equi-join

- Most frequently used in practice:

$$R_1 \bowtie_{A=B} R_2$$

- Natural join is a particular case of equi-join.
- A lot of research on how to do it efficiently.

Summary of Relational Algebra

- Basic primitives:
  $E ::=$
    - R
    - $\sigma_c(E)$
    - $\pi_{A_1,,\ldots,A_n}(E)$
    - $E_1 \times E_2$
    - $E_1 \cup E_2$
    - $E_1 - E_2$
    - $\rho_{S(A_1,\ldots,A_n)}(E)$
- Derived:
    - $E_1 \bowtie E_2$
    - $E_1 \bowtie_c E_2$
    - $E_1 \cap E_2$

Behind the Scene: Other query languages?

Find the manufacturer of the beers that people in Champaign like.

Relational algebra:

- Join Drinkers(bname, city) with Likes(drinker, beer).
- Join that with Beers(bname, manf).
- Restrict to tuples to Drinker.city = "Champaign, IL".
- Project over Beers.manf.

Relational calculus:

- Return Beers.manf such that there exists Likes.beer = Beers.name and Likes.drinker = Drinkers.name and Drinkers.city = "Champaign, IL".

Do you see-- So what's the difference?

Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.

- $X_1 := Drinkers(dname, city) \bowtie_{dname=drinker} Likes(drinker, beer)$
- $X_2 := X_1 \bowtie_{beer=bname} Beers(bname, manf)$
- $Answer(company) \coloneqq \pi_{manf}(\sigma_{city="champaign"}\, X_2)$

Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example

- Given Bars(name, addr), Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

Name: ______ NetID: ______

1. What do you like?

2. What do you dislike?

way to go

3. On a scale of 1 ... 5

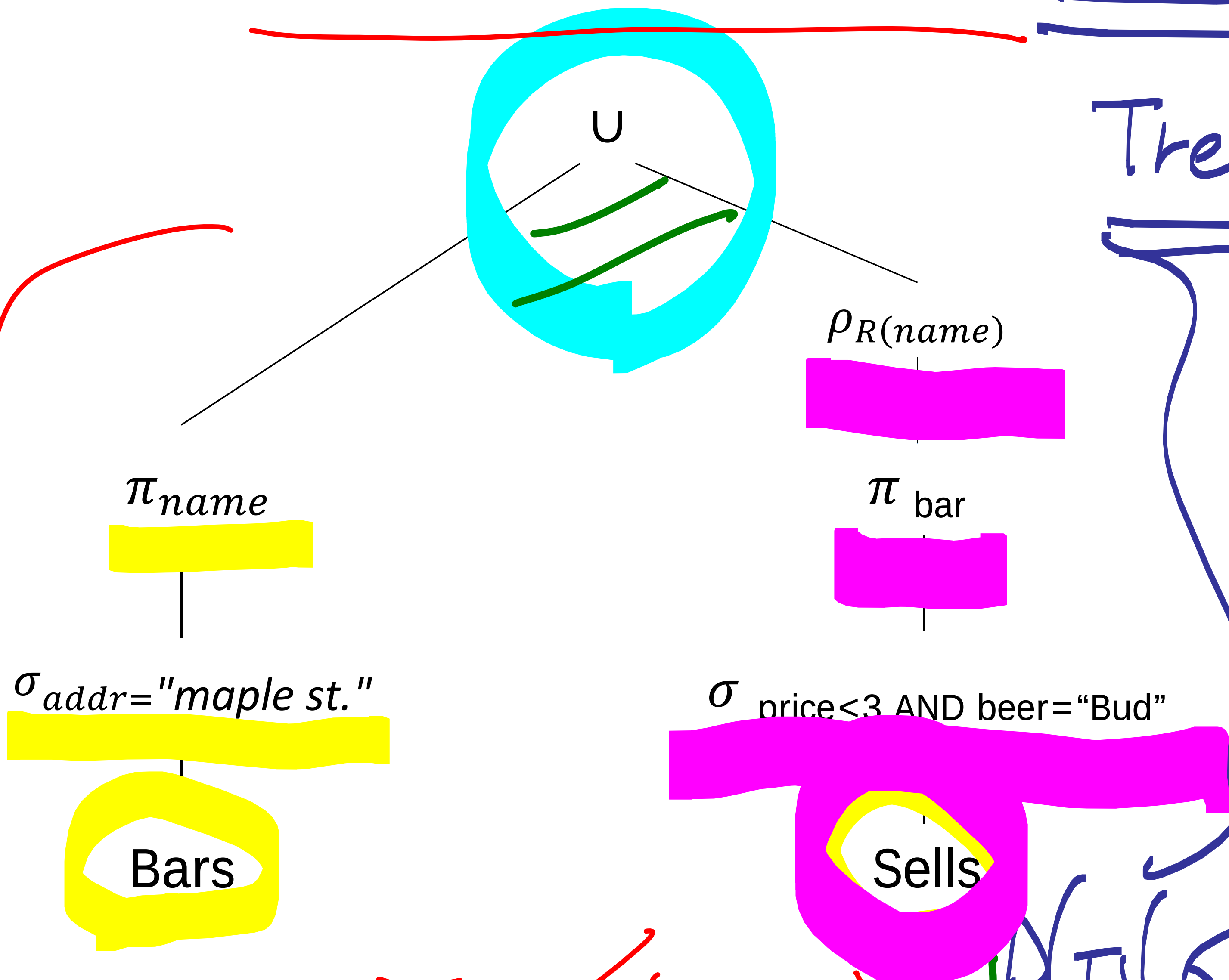
1	2	3	4	5
o	o	o	o	o
poor	bad	Avg	good	Excellent

Don't do that

As a Tree: Parse Tree

Query Opt.

Tree $\rightarrow$ Tree



$\pi_{name}\sigma(Bars) \cup \rho(\pi(\sigma(Sells)))$

Q: How to do this?

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.

Operations on Bags (and why we care)

- Union: {a,b,b,c} U {a,b,b,b,e,f,f} = {a,a,b,b,b,b,b,c,e,f,f}
  - *add* the number of occurrences
- Difference: {a,b,b,b,c,c} – {b,c,c,c,d} = {a,b,b}
  - subtract the number of occurrences
- Intersection: {a,b,b,b,c,c}   {b,b,c,c,c,c,d} = {b,b,c,c}
  - minimum of the two numbers of occurrences
- Selection: preserve the number of occurrences
- Projection: preserve the number of occurrences (no duplicate elimination)
- Cartesian product, join: no duplicate elimination
- → Read book for detail.

- But, why do we care?