Relational Model and Algebra

Introduction to Databases

CompSci 316 Fall 2020

DUKE
COMPUTER SCIENCE

Announcements (Thu. Aug. 20)

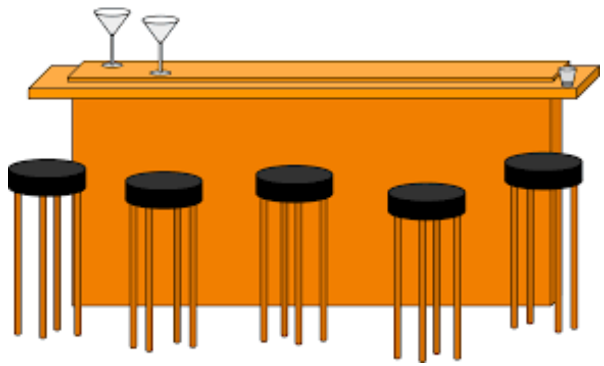
- Project details posted on Sakai
  - Read it carefully!
  - Think about fixed vs. open project (some project videos from last semester will be available on sakai soon – keep them private)
  - Roster for discussion sessions available on sakai (teammates have to be from the same discussion session)
  - You do not have to form your teams or decide fixed/open projects right now. Names of team members and project choices are due on 9/8, so you will have some time (and the class/discussion sections are still in flux)
- Survey has been sent – Due by tomorrow 08/21 night EDT
  - To know about your time zones, expectations, available resources, project / team-member preference etc.
  - Please respond on time – there is a 2% weight for communication!
- Monday's discussion sessions: Installation and practice SQL
  - Emails coming soon

Today's plan

- Revisit relational model
- Simple SQL queries and its semantic
- Start relational algebra

The famous "Beers" database

Bar
Each has an address



Drinker **Frequents** Bars
"X" times a week

Bar **Serves** Beer
At price "Y"

Drinker **Likes** Beer

Drinker
Each has an address

Beer
Each has a brewer

See online database for more tuples

"Beers" as a Relational Database

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Beer

Name	brewer
Budweiser	Anheuser-Busch Inc.
Corona	Grupo Modelo
Dixie	Dixie Brewing

Drinker

name	address
Amy	100 W. Main Street
Ben	101 W. Main Street
Dan	300 N. Duke Street

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Frequents

drinker	beer
Amy	Corona
Dan	Budweiser
Dan	Corona
Ben	Budweiser

Likes

What is an example of a

- Relation
- Attribute
- Tuple
- Schema
- Instance

What is

- Set semantic
  - in relational model
- Bag semantic
  - In SQL (why)

See online database for more tuples

"Beers" as a Relational Database

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Beer

Name	brewer
Budweiser	Anheuser-Busch Inc.
Corona	Grupo Modelo
Dixie	Dixie Brewing

Drinker

name	address
Amy	100 W. Main Street
Ben	101 W. Main Street
Dan	300 N. Duke Street

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Likes

drinker	beer
Amy	Corona
Dan	Budweiser
Dan	Corona
Ben	Budweiser

What is an example of a

- Relation
- Attribute
- Tuple
- Schema
- Instance

What is

- Set semantic
  - in relational model
- Bag semantic
  - In SQL (why)

- Set semantic
  - No duplicates, Order of tuples does not matter
- Bag semantic
  - Duplicates allowed, for efficiency and flexibility
  - Do not want duplicates? Use SELECT DISTINCT …

Basic queries: SFW statement

- SELECT $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition$

In HW1, you can only use SFW

- SELECT, FROM, WHERE are often referred to as SELECT, FROM, WHERE "clauses"
- Each query must have a SELECT and a FROM
- WHERE is optional

Example: reading a table

- SELECT *
  FROM Serves

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

- Single-table query
- * is a shorthand for "all columns"

Example: ORDER BY

- SELECT *
  FROM Serves
  ORDER BY beer

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

- Equivalent to "ORDER BY beer asc" (asc is default option)
- For descending order, use "desc"
- Can combine multiple orders
- What does this return?
  - ORDER BY beer asc, price desc

Example: some columns and DISTINCT

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

- SELECT beer
  FROM Serves

Returns a bag

- Only want unique values? Use DISTINCT

- SELECT DISTINCT beer
  FROM Serves

Returns a set

Example: selecting few rows

- SELECT beer AS mybeer
  FROM Serves
  WHERE price < 2.75

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

- SELECT S.beer
  FROM Serves S
  WHERE bar = ‘The Edge’

What does these return?

- SELECT list can contain expressions
  Can also use built-in functions such as SUBSTR, ABS, etc.
- NOT EQUAL TO: Use <>
- LIKE matches a string against a pattern
  % matches any sequence of zero or more characters

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

- Which tables do we need?

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Beer

Name	brewer
Budweiser	Anheuser-Busch Inc.
Corona	Grupo Modelo
Dixie	Dixie Brewing

Drinker

name	address
Amy	100 W. Main Street
Ben	101 W. Main Street
Dan	300 N. Duke Street

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Frequents

drinker	beer
Amy	Corona
Dan	Budweiser
Dan	Corona
Ben	Budweiser

Likes

Which tables do we need?

How do we combine them?

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

  - SELECT B.address
    FROM Bar B, Frequents F
    WHERE B.name = F.bar
    AND F.drinker = ‘Dan’

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Frequents

Semantics of SFW

- SELECT $E_1, E_2, \ldots, E_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition$

- For each t_1 in R_1:
  For each t_2 in R_2: … …
  For each t_m in R_m:

1. Apply "FROM"
Form "cross-product" of R1, .., Rm

If $condition$ is true over $t_1, t_2, \ldots, t_m$:

2. Apply "WHERE"
Only consider satisfying rows

Compute and output $E_1, E_2, \ldots, E_n$ as a row

3. Apply "SELECT"
Output the desired columns

Step 1: Illustration of Semantics of SFW

- NOTE: This is "NOT HOW" the DBMS outputs the result, but "WHAT" it outputs!

- SELECT B.address
  FROM Bar B, Frequents F
  WHERE B.name = F.bar
  AND F.drinker = 'Dan'

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Form a "Cross product" of two relations

name	address	drinker	bar	times_a_week
The Edge	108 Morris Street	Ben	Satisfaction	2
The Edge	108 Morris Street	Dan	The Edge	1
The Edge	108 Morris Street	Dan	Satisfaction	2
Satisfaction	905 W. Main Street	Ben	Satisfaction	2
Satisfaction	905 W. Main Street	Dan	The Edge	1
Satisfaction	905 W. Main Street	Dan	Satisfaction	2

Step 2: Illustration of Semantics of SFW

- NOTE: This is "NOT HOW" the DBMS outputs the result, but "WHAT" it outputs!

Discard rows that do not satisfy WHERE condition

- SELECT B.address
  FROM Bar B, Frequents F
  WHERE B.name = F.bar
  AND F.drinker = 'Dan'

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

name	address	drinker	bar	times_a_week
~~The Edge~~	~~108 Morris Street~~	~~Ben~~	~~Satisfaction~~	~~2~~
The Edge	108 Morris Street	Dan	The Edge	1
~~The Edge~~	~~108 Morris Street~~	~~Dan~~	~~Satisfaction~~	~~2~~
~~Satisfaction~~	~~905 W. Main Street~~	~~Ben~~	~~Satisfaction~~	~~2~~
~~Satisfaction~~	~~905 W. Main Street~~	~~Dan~~	~~The Edge~~	~~1~~
Satisfaction	905 W. Main Street	Dan	Satisfaction	2

Step 3: Illustration of Semantics of SFW

- NOTE: This is "NOT HOW" the DBMS outputs the result, but "WHAT" it outputs!

Output the "address" output of rows that survived

- SELECT B.address
  FROM Bar B, Frequents F
  WHERE B.name = F.bar
  AND F.drinker = 'Dan'

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

name	address	drinker	bar	times_a_week
~~The Edge~~	~~108 Morris Street~~	~~Ben~~	~~Satisfaction~~	~~2~~
The Edge	108 Morris Street	Dan	The Edge	1
~~The Edge~~	~~108 Morris Street~~	~~Dan~~	~~Satisfaction~~	~~2~~
~~Satisfaction~~	~~905 W. Main Street~~	~~Ben~~	~~Satisfaction~~	~~2~~
~~Satisfaction~~	~~905 W. Main Street~~	~~Dan~~	~~The Edge~~	~~1~~
Satisfaction	905 W. Main Street	Dan	Satisfaction	2

Final output: Illustration of Semantics of SFW

- NOTE: This is "NOT HOW" the DBMS outputs the result, but "WHAT" it outputs!

Output the "address" output of rows that survived

- SELECT B.address
  FROM Bar B, Frequents F
  WHERE B.name = F.bar
  AND F.drinker = 'Dan'

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

address
108 Morris Street
905 W. Main Street

SQL vs. C++, Java, Python…

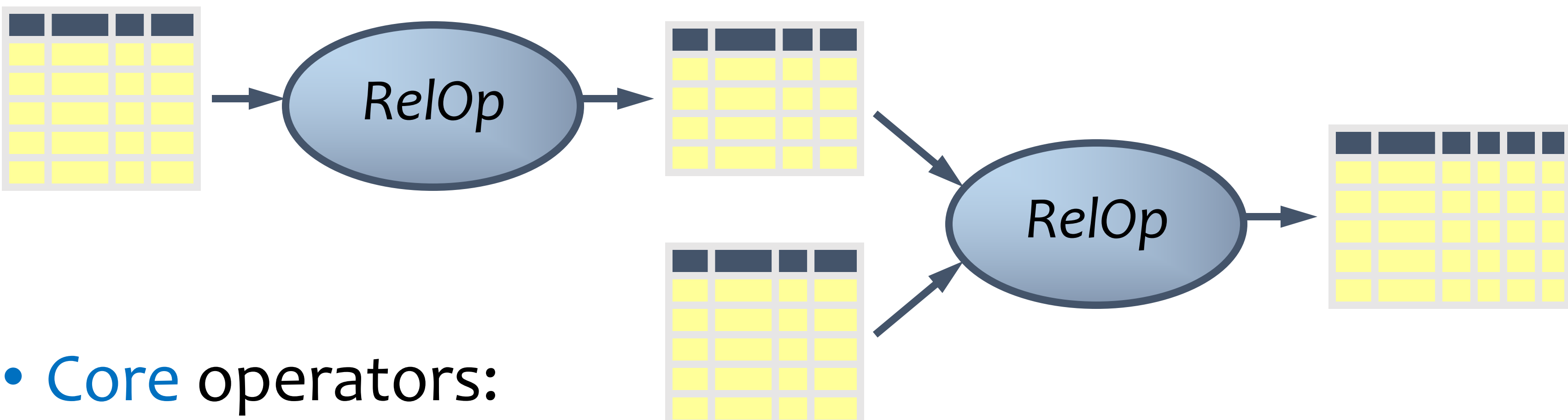
SQL vs. C++, Java, Python…

SQL is declarative

- Programmer specifies what answers a query should return,
- but not how the query is executed
- DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.
- Not a "Procedural" or "Operational" language like C++, Java, Python
- There are several ways to write a query, but equivalent queries always provide the same (equivalent) results
- SQL (+ its execution and optimizations) is based on a strong foundation of "Relational Algebra"

Relational algebra

A language for querying relational data based on "operators"



- Core operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries

Selection

- Input: a table R
- Notation: $\sigma_p R$
  - p is called a selection condition (or predicate)
- Purpose: filter rows according to some criteria
- Output: same columns as R, but only rows of R that satisfy p (set!)

Example: Find beers with price < 2.75

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

$\sigma_{price<2.75}$ **Serves**

bar	beer	price
The Edge	Budweiser	2.50
[redacted]		
Satisfaction	Budweiser	2.25

No actual deletion!

Equivalent SQL query?

More on selection

- Selection condition can include any column of R, constants, comparison ($=$, $\leq$, etc.) and Boolean connectives ($\wedge$: and, $\vee$: or, $\neg$: not)
  - Example: Serves tuples for "The Edge" or price >= 2.75

$$\sigma_{bar='The\ Edge' \vee price \geq 2.75}\ Serves$$

- You must be able to evaluate the condition over each single row of the input table!
  - Example: the most expensive beer at any bar

$$\sigma_{price \geq every\ price\ in\ Serves}\ User$$

WRONG!

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

Projection

End of lecture 08/20

- Input: a table R
- Notation: $\pi_L R$
  - L is a list of columns in R
- Purpose: output chosen columns
- Output: same rows, but only the columns in L $(set!)$

Example: Find all the prices for each beer

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

$\pi_{beer,price}$ **Serves**

beer	price
Budweiser	2.50
Corona	3.00
Budweiser	2.25

Output of π_{beer}Serves?

Cross product

- Input: two tables R and S
- Natation: $R \times S$
- Purpose: pairs rows from two tables
- Output: for each row r in R and each s in S, output a row rs (concatenation of r and s)

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

name	address	drinker	bar	times_a_week
The Edge	108 Morris Street	Ben	Satisfaction	2
The Edge	108 Morris Street	Dan	The Edge	1
The Edge	108 Morris Street	Dan	Satisfaction	2
Satisfaction	905 W. Main Street	Ben	Satisfaction	2
Satisfaction	905 W. Main Street	Dan	The Edge	1
Satisfaction	905 W. Main Street	Dan	Satisfaction	2

Bar x Frequents

Note: ordering of columns does not matter, so R X S = S X R (commutative)

Derived operator: join

(A.k.a. "theta-join": most general joins)

- Input: two tables R and S
- Notation: $R \bowtie_p S$
  - p is called a join condition (or predicate)

One of the most important operations!

- Purpose: relate rows from two tables according to some criteria

- Output: for each row r in R and each row s in S, output a row rs if r and s satisfy p

- Shorthand for $\sigma_p(R \times S)$

Predicate p only has conjunctions of equality

e.g., (A1 = A2) ∧ (B1 = B2) ∧ (C1 = C2) : equijoin

Join example

Ambiguous attribute?
Use Bar.name

- Extend Frequents relation with addresses of the bars

$$Frequents \bowtie_{bar=name} Bar$$

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

name	address	drinker	bar	times_a_week
~~The Edge~~	~~108 Morris Street~~	~~Ben~~	~~Satisfaction~~	~~2~~
The Edge	108 Morris Street	Dan	The Edge	1
~~The Edge~~	~~108 Morris Street~~	~~Dan~~	~~Satisfaction~~	~~2~~
Satisfaction	905 W. Main Street	Ben	Satisfaction	2
~~Satisfaction~~	~~905 W. Main Street~~	~~Dan~~	~~The Edge~~	~~1~~
Satisfaction	905 W. Main Street	Dan	Satisfaction	2

Join Types

- Theta Join

- Equi-Join

- Natural Join

- Later, (left/right) outer join, semi-join

Derived operator: natural join

- Input: two tables R and S
- Notation: $R \bowtie S$ (i.e. no subscript)
- Purpose: relate rows from two tables, and
  - Enforce equality between identically named columns
  - Eliminate one copy of identically named columns
- Shorthand for $\pi_L(R \bowtie_p S)$, where
  - p equates each pair of columns common to R and S
  - L is the union of column names from R and S (with duplicate columns removed)

Natural join example

$$Serves \bowtie Likes$$
$$= \pi_{?}(Serves \bowtie_{?} Likes)$$
$$= \pi_{bar,beer,price,drinker}\left(Serves \bowtie_{\substack{Serves.beer= \\ Likes.beer}} Likes\right)$$

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

Likes

drinker	beer
Amy	Corona
Dan	Budweiser
Dan	Corona
Ben	Budweiser

$Serves \bowtie Likes$

bar	beer	price	drinker
The Edge	Budweiser	2.50	Dan
The Edge	Budweiser	2.50	Ben
The Edge	Corona	3.00	Amy
The Edge	Corona	3.00	Dan
…	….	…..	

Natural Join is on beer

Only one column for beer in the output

What happens if the tables have two or more common columns?

Union

Important for set operations:

Union Compatibility

- Input: two tables R and S
- Notation: $R \cup S$
  - R and S must have identical schema
- Output:
  - Has the same schema as R and S
  - Contains all rows in R and all rows in S (with duplicate rows removed)

Example on board

Difference

Important for set operations:

Union Compatibility

- Input: two tables R and S
- Notation: $R - S$
  - R and S must have identical schema
- Output:
  - Has the same schema as R and S
  - Contains all rows in R that are not in S

Example on board

Derived operator: intersection

Important for set operations:

Union Compatibility

- Input: two tables R and S
- Notation: $R \cap S$
  - R and S must have identical schema
- Output:
  - Has the same schema as R and S
  - Contains all rows that are in both R and S
- How can you write it using other operators?

- Shorthand for $R - (R - S)$
- Also equivalent to $S - (S - R)$
- And to $R \bowtie S$

What if you move σ to the top?
Still correct?
More or less efficient?

Expression tree notation

- Find addresses of all bars that 'Dan' frequents

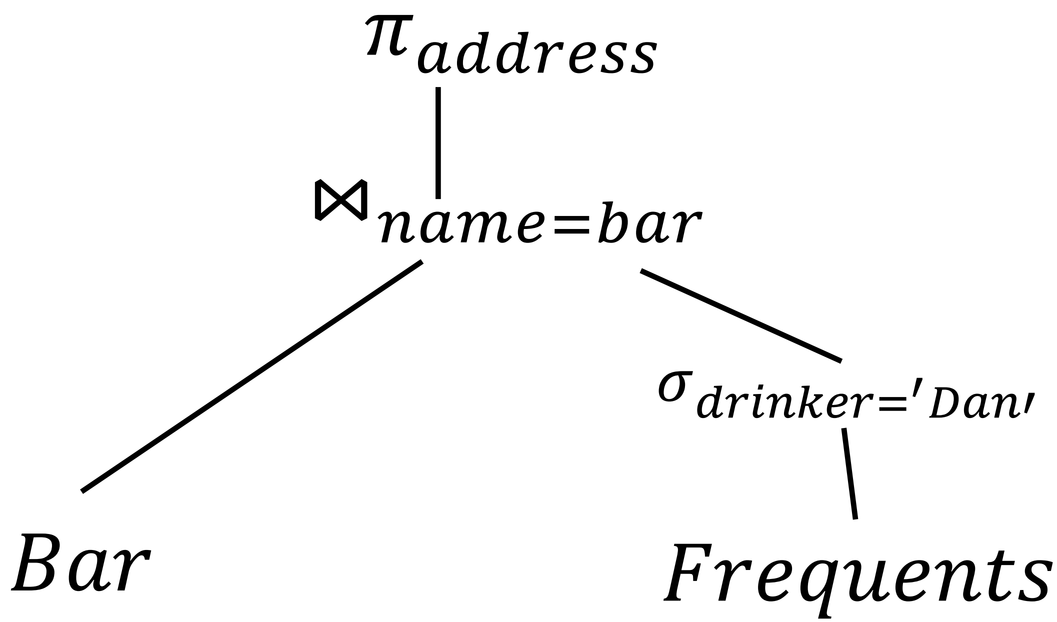
Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Also called logical Plan tree



Equivalent to

$$\pi_{address}\left(Bar \bowtie_{name = bar} \left(\sigma_{drinker='Dan'} Frequents\right)\right)$$

Using the same relation multiple times

- Find drinkers who frequent both “The Edge” and “Satisfaction”

Frequents

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

WRONG!

$$\pi_{drinker}\left(Frequents \bowtie_{\substack{bar='The\ Edge' \wedge \\ bar='Satisfaction' \wedge \\ drinker=drinker}} Frequents\right)$$

Rename!

$$\pi_{d1}\left(\begin{array}{c}\rho_{F1(d1,b1,t1)}Frequents \\ \bowtie_{b1='The\ Edge' \wedge b2='Satisfaction' \wedge d1=d2} \\ \rho_{F2(d1,b1,t1)}Frequents\end{array}\right)$$

Renaming

- Input: a table R
- Notation: $\rho_S\, R$, $\rho_{(A_1, A_2, \ldots)} R$, or $\rho_{S(A_1, A_2, \ldots)} R$
- Purpose: "rename" a table and/or its columns
- Output: a table with the same rows as R, but called differently
- Used to
  - Avoid confusion caused by identical column names
  - Create identical column names for natural joins
- As with all other relational operators, it doesn't modify the database
  - Think of the renamed table as a copy of the original

Summary of core operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_{S(A_1, A_2, \ldots)} R$
  - Does not really add "processing" power

Summary of derived operators

- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Intersection: $R \cap S$

- Many more
  - Semijoin, anti-semijoin, quotient, …

End of lecture 08/25

Announcements (Tue. Aug. 25)

- Reminder:
- Post all questions about lectures/HW on piazza and answer each other's questions!
- Remember to sign in while watching recordings
  - Everyone: please try for today's lecture by tomorrow (Wed) night

Frequents(drinker, bar, times_of_week)
Bar(name, address)
Drinker(name, address)

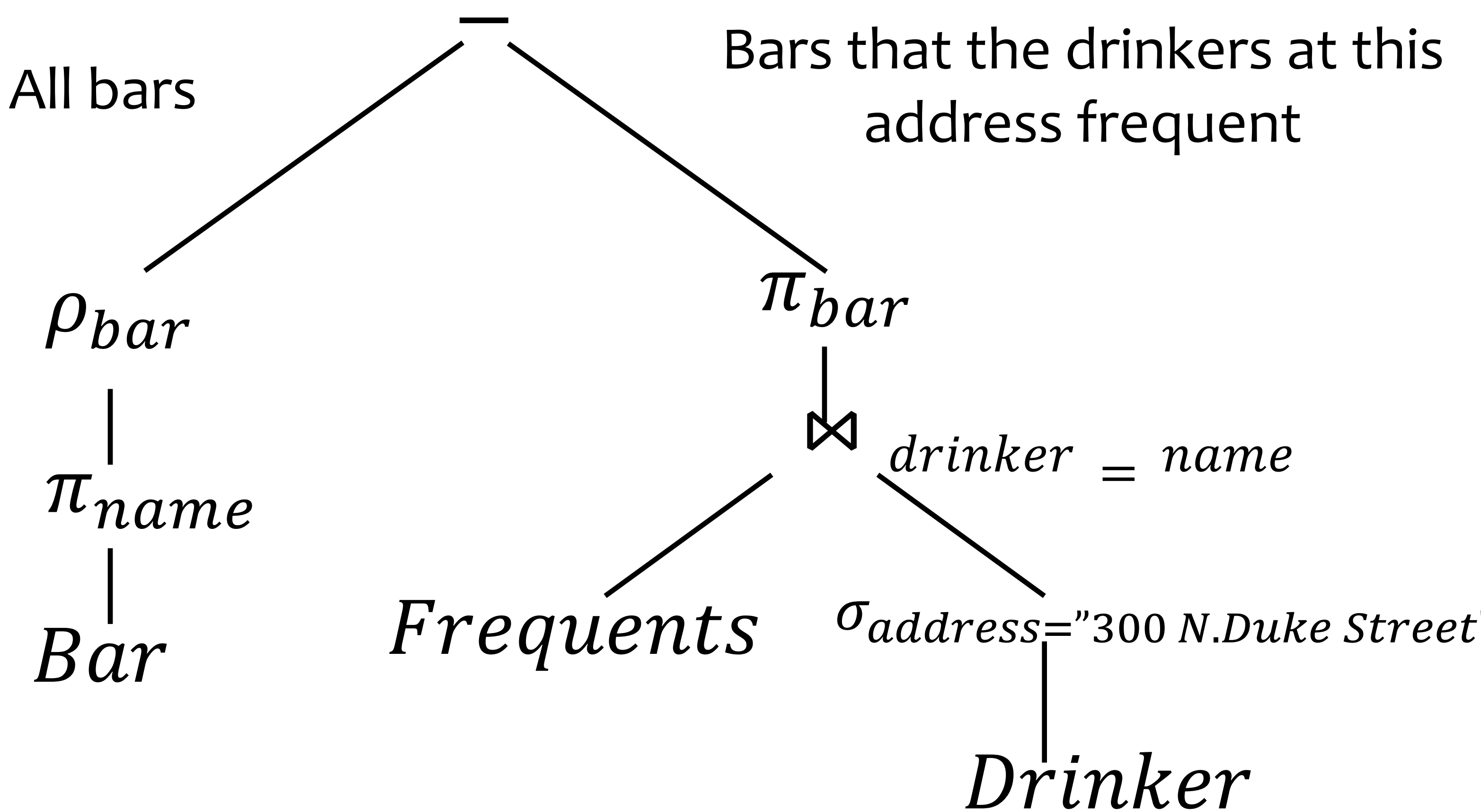
Exercise

- Bars that drinkers in address “300 N. Duke Street” do not frequent

Frequents(drinker, bar, times_of_week)
Bar(name, address)
Drinker(name, address)

Exercise

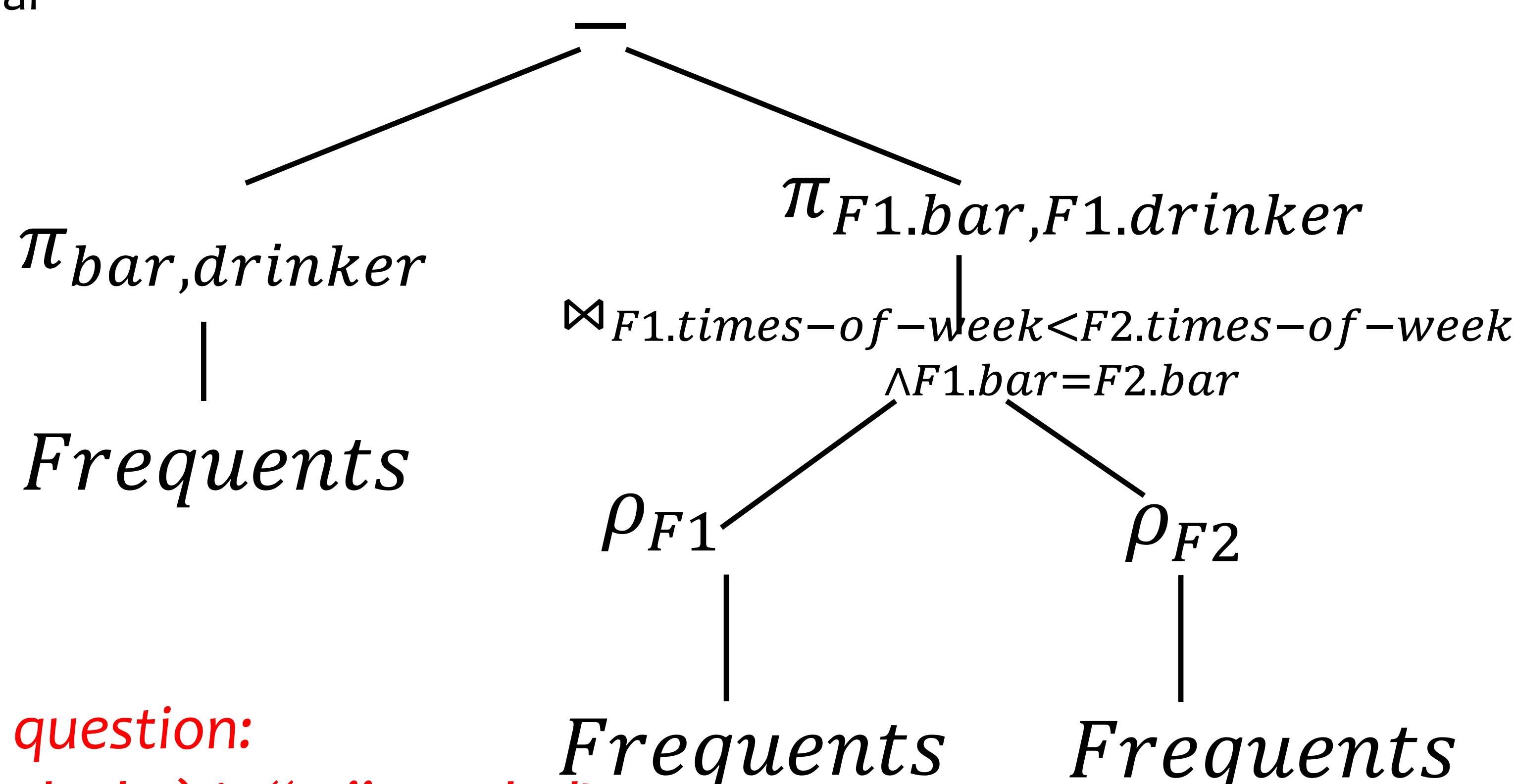
- Bars that drinkers in address "300 N. Duke Street" do not frequent



A trickier exercise

Frequents(drinker, bar, times_of_week)
Bar(name, address)
Drinker(name, address)

- For each bar, find the drinkers who frequent it max no. times a week
  - Who do NOT visit a bar max no. of times?
  - Whose *times_of_weeks* is lower than somebody else's for a given bar



A deeper question:
When (and why) is "$-$" needed?

A trickier exercise

Frequents(drinker, bar, times_of_week)
Bar(name, address)
Drinker(name, address)

- For each bar, find the drinkers who frequent it max no. times a week

What if there are different drinkers with the same name in the Frequents table?

Drinker	Bar	Times_of_week
Dan	The Edge	7
Dan	The Edge	5
Joe	The Edge	6

Correct answer: (Dan, The Edge)

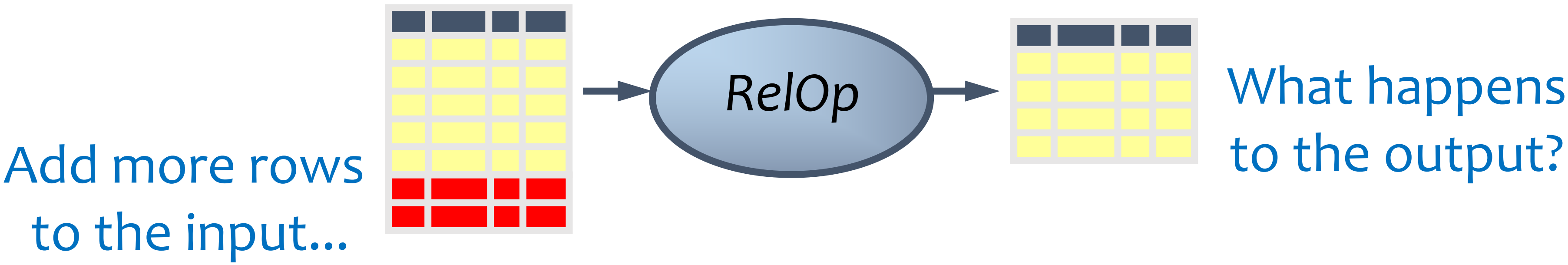
What does the previous query return? Empty set

How to fix the query?

1. Project to (drinker, bar, times_a_week) both sides
2. Take difference –
3. Project to (drinker, bar)

In general, projection before and after difference can give very different results -- check carefully which one is correct!

Monotone operators



- If some old output rows may need to be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows always remain "correct" when more rows are added to the input
- Formally, for a monotone operator op:
  $R \subseteq R'$ implies $op(R) \subseteq op(R')$ for any R, R'

Which operators are non-monotone?

• Selection: $\sigma_p R$	Monotone
• Projection: $\pi_L R$	Monotone
• Cross product: $R \times S$	Monotone
• Join: $R \bowtie_p S$	Monotone
• Natural join: $R \bowtie S$	Monotone
• Union: $R \cup S$	Monotone
• Difference: $R - S$	Monotone w.r.t. R; non-monotone w.r.t S
• Intersection: $R \cap S$	Monotone

Why is "–" needed for "highest"?

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Is the "highest" query monotone?
  - No!
  - Current highest *price* 3.0
  - Add another row with *price 3.01*
  - Old answer is invalidated

☞So it must use difference!

Extensions to relational algebra

- Duplicate handling (“bag algebra”)
- Grouping and aggregation
- “Extension” (or “extended projection”) to allow new column values to be computed

- (Coming later)