



International University, VNU-HCMC



School of Computer Science and Engineering

Lecture 4: Relational Model and Algebra

Instructor: Nguyen Thi Thuy Loan
nttloan@hcmiu.edu.vn, nthithuyloan@gmail.com
<https://nttloan.wordpress.com/>



International University, VNU-HCMC

Purpose of the Lecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- Introduce core concepts of the Relational Model.
- Explain its role as the foundation of database systems.
- Present the primary Relational Algebra operations.

Duke CS, Fall 2024

2



Warm-up Question

- Why do modern database systems rely on the relational model, and how can we formally describe queries on data?



Outlines

- Relational model
- Relational algebra



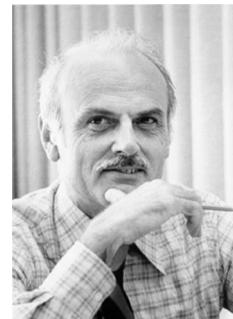
Acknowledgement

- The following slides are referenced from Dr. Sudeepa Roy, Duke University.



Edgar F. Codd (1923-2003)

- Served as a pilot in the Royal Air Force during WW2
- Invented the Relational Model and Algebra at IBM (1970)
- Awarded the Turing Award in 1981 for his pioneering work
- His ideas led to the development of RDBMS (Relational Database Management Systems)



http://en.wikipedia.org/wiki/File:Edgar_F_Codd.jpg



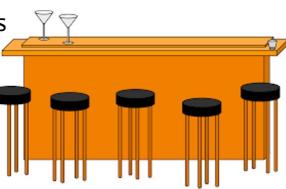
The famous “Beers” database

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Drinkers **Frequent** Bars
“X” times a week

Drinkers
Each has an address

(Later in ER diagram – how to design a relational database)



Bars
Each has an address

Bars **Serve** Beers
At price “Y”



Beers
Each has a brewer

Duke CS, Fall 2024

7



“Beers” as a Relational Database

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

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

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

Duke CS, Fall 2024



Relational data model

- A database is a collection of **relations** (or **tables**)
- Each relation has a set of **attributes** (or **columns**)
- Each attribute has a name and a **domain** (or **type**)
 - No set-valued attributes allowed

Serves

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

Duke CS, Fall 2024

9



Relational data model

- Each relation is a set of **tuples** (or **rows**)
 - Each tuple has a value for each attribute of the relation
 - No duplicate tuples (same values across all attributes)
 - Row order doesn't matter (even though output is always in some order)
- In practice, SQL supports “bags” (allows duplicates)
- Why? Simplicity is a virtue!

Serves

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

Duke CS, Fall 2024

10



Schema vs. instance

Serves

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

Beer

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

Frequents

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

- Ordering of rows doesn't matter (even though output is always in some order)

Duke CS, Fall 2024

11



Schema vs. instance

- Schema (metadata)
 - Specifies the logical structure of data
 - Is defined at setup time
 - Rarely changes
- Instance
 - Represents the data content
 - Changes rapidly, but always conforms to the schema
- Compare types vs. collections of objects of these types in a programming language

Duke CS, Fall 2024

12



Example

- Schema (metadata)
 - Beer (name string, brewer string)
 - Serves (bar string, beer string, price float)
 - Frequents (drinker string, bar string, times_a_week int)
- Instance
 - Beer {<Budweiser, Anheuser-Busch Inc.>, <corona, Grupo Modelo>, ... }
 - Serves {<the Edge, Budweiser, 2.50>, <The Edge, Corona, 3.0>, ... }
 - Frequents {<Ben, Satisfaction,2>, <Dan, The Edge, 1>, ... }

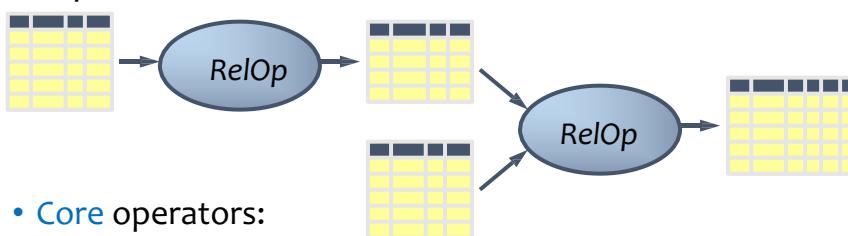
Duke CS, Fall 2024

13



Relational algebra

- A language for querying relational databases on “operators”



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

Duke CS, Fall 2024

14



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!)



Selection 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

$\sigma_{\text{price} < 2.75}$ Serves		
bar	beer	price
The Edge	Budweiser	2.50
Satisfaction	Budweiser	2.25



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 \geq

2.75

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

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

Duke CS, Fall 2024

17



More on selection

- You must be able to evaluate the condition for each row of the input table.

– Example: the most expensive beer at any bar

$$\sigma_{\text{price} \geq \text{every price in Servers}}^{\text{User}}$$

WRONG!

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

Duke CS, Fall 2024

18



Projection

- 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!)



Projection

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 $\pi_{beer} Serves$?



More on Projection

- Duplicate output rows are removed (by definition)
- Example: beer on servers

Serves

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

π_{beer} Serves

beer
Budweiser
Corona



Cross product

- Input: two tables R and S
- Notation: $R \times S$
- 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)



Cross product

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

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

Bar x Frequents

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

Duke CS, Fall 2024

23



Cross product

hD

- Ordering of columns is unimportant as far as contents are concerned.



Assoc. Prof. Nguyen Thi Thuy Loan, PhD

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

||

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

- So cross product is commutative, i.e., for any R and S, $R \times S = S \times R$ (up to the ordering of columns)

Duke CS, Fall 2024

24



Derived operator: join

- (Also known as “**theta-join**”: most general joins)
- Input: two tables R and S
 - Notation: $R \bowtie_p S$
 - p is called a join condition (or predicate)
 - 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 equality ($A = 5 \wedge B = 7$): **equijoin**



Join example

- Extend **Frequents** relation with addresses of the bars

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

Ambiguous attribute? Prefix a column reference with table name and “.” to disambiguate identically named columns from different tables. Ex. Use Bar.name

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	4
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

$= \pi_? (Serves \bowtie ? Likes)$

$= \pi_{\text{bar, beer, price, drinker}} (Serves \bowtie_{\text{serves.beer} = \text{Likes.beer}} Likes)$

Serves

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

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

- 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)

Important for set operations:
Union Compatibility

Example on board



Example

Student_A	
StudentID	Name
3	Chi
4	Dũng
5	Hạnh
6	Khoa

Student_B	
StudentID	Name
1	An
2	Bình
3	Chi
4	Dũng



Derived operator: intersection

- 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 $R \bowtie S$

Important for set operations:
Union Compatibility



Example

Student_A	
StudentID	Name
3	Chi
4	Dũng
5	Hạnh
6	Khoa

Student_B	
StudentID	Name
1	An
2	Bình
3	Chi
4	Dũng



Difference

- 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
- Important for set operations:
Union Compatibility

Example on board



Example

Student_A	
StudentID	Name
3	Chi
4	Dũng
5	Hạnh
6	Khoa

Student_B	
StudentID	Name
1	An
2	Bình
3	Chi
4	Dũng



Renaming

- Input: a table R
- Notation: $\rho_S R$, $\rho_{(A_1, A_2, \dots)}^R$, or $\rho S_{(A_1, A_2, \dots)} R$
- Purpose: “rename” a relation and/or its attributes
- Output: a new relation with the same rows as R , but a new name and/or new column names
- Example:
 - $\rho(\text{Tempsids}, \text{Student})$
 - $\rho(S(\text{SID}, \text{SName}), \text{Student})$



Renaming

- Why use it?
 - Avoid confusion when column names are identical
 - Create consistent attribute names for natural joins
- Notes:
 - Renaming does not modify the original database
 - Think of it as creating a temporary copy of R with new names



Renaming example

- 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!

$$\pi_{\text{drinker}} \left(\text{Frequents} \bowtie \begin{array}{l} \text{Bar= 'The Edge} \wedge \\ \text{Bar = 'Satisfaction'} \wedge \\ \text{drinker = drinker} \end{array} \right)$$

$$\pi_{\text{uid}_1} \left(\begin{array}{l} \bowtie \rho_{(d1, b1, t1)} \text{Frequents} \\ \bowtie \text{b1 = 'The Edge' } \wedge \text{b2 = Satisfaction' } \wedge \text{d1=d2} \\ \rho_{(d2, b2, t2)} \text{Frequents} \end{array} \right)$$

Rename!



Exercise

Given a database

- Student(SID, Sname, age, add, phone, ID_card)
- Course(CID, Cname, credit)
- Enrolled (SID, CID, grade)

- Find all the students (SID, Sname) who enroll in the PDM course?
- Find the students (SID, Sname) who enroll in both the OOP and PDM courses?
- Find the course name that didn't have any students to enroll?



Expression tree notation

- Find addresses of all bars that 'Dan' frequents

What if you move σ to the top?

Still correct?

More or less efficient?

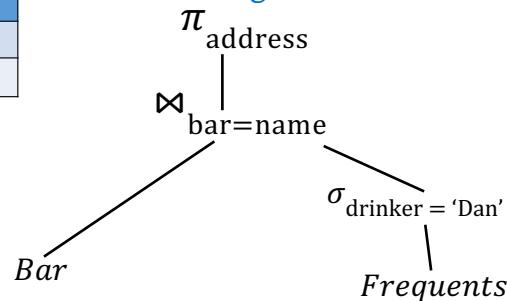
Bar

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

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



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: $\rho_{S(A_1, A_2, \dots)} R$
 - 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, ...



Exercise

$\text{Frequents}(\text{drinker}, \text{bar}, \text{times_of_week})$
 $\text{Bar}(\text{name}, \text{address})$
 $\text{Drinker}(\text{name}, \text{address})$

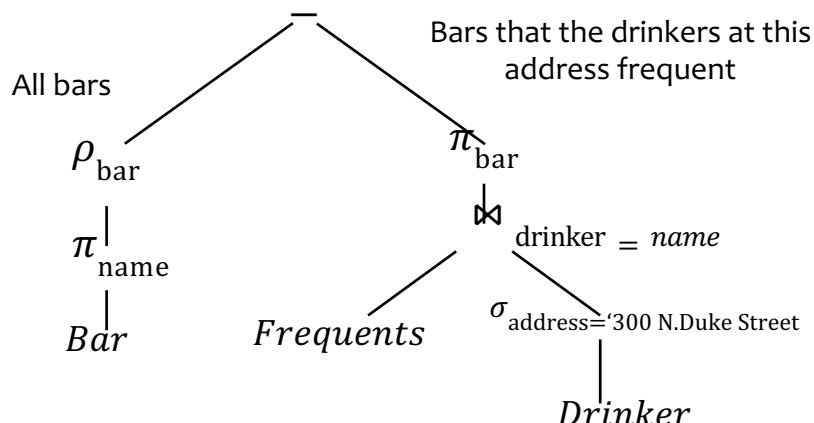
- Find the bars that are not frequented by drinkers who live at the address “300 N. Duke Street”.



Exercise

$\text{Frequents}(\text{drinker}, \text{bar}, \text{times_of_week})$
 $\text{Bar}(\text{name}, \text{address})$
 $\text{Drinker}(\text{name}, \text{address})$

- Find the bars that are not frequented by drinkers who live at the address “300 N. Duke Street”.





A trickier Exercise

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

41

- For each bar, find the drinkers who frequent it the maximum number of times per week.



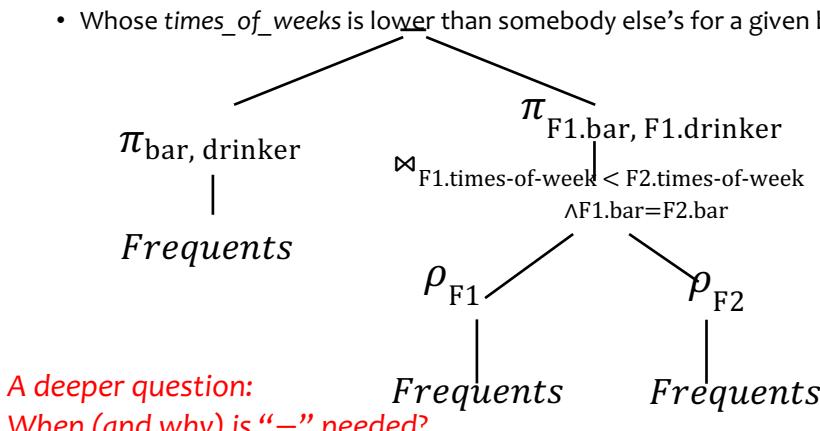
A trickier Exercise

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

42

- For each bar, find the drinkers who frequent it the maximum number of times per 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





Expressions in a Single Assignment

- Example: the theta-join $R3 = R1 \bowtie_C R2$ can be written: $R3 := \sigma_C (R1 \times R2)$
- Precedence of relational operators:
 1. $[\sigma, \pi, \rho]$ (highest)
 2. $[X, \bowtie]$
 3. \cap
 4. $[\cup, -]$



Example

Find the names of sailors who have reserved boat number 103.

- Sailors(sid, sname, rating, age)
- Boats(bid, bname, color)
- Reserves(sid, bid, day)



Find sailors who've reserved a red or a green boat.

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Use of rename operation

Reserves(sid, bid, day)

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho (\text{Tempboats}, (\sigma_{\text{color} = 'red' \vee \text{color} = 'green'} \cdot \text{Boats}))$$

$\pi_{\text{sname}}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$
Can also define Tempboats using union. Try the “AND” version yourself



Division

- Not supported as a primitive operator, but useful for expressing queries like:

Find sailors who have reserved all boats.

- Let A have 2 fields, x and y ; B have only field y :

$$A/B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

- i.e., A/B contains all x tuples (sailors) such that for every y tuple (boat) in B , there is an xy tuple in A .
- Or: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B , the x value is in A/B .



Examples of Division A/B

$$\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

sno	pno
s1	p1
s1	p
s1	2
s1	p
s2	3
s2	p
s3	4
s4	p
s4	1

A/
p
2
p

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Duke CS, Fall 2024

51



Expressing A/B Using Basic Operators

- Division is not essential operator; just a useful shorthand.
 - (Also true of joins, but joins are so common that systems implement joins specially)
- Idea: For A/B, find all x values that are not disqualified by any y in B.
 - An x value is disqualified if, when combined with some y from B, the tuple (x,y) does not appear in A.

Disqualified x values:

all disqualified tuples

A/B:

$$\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

Duke CS, Fall 2024

52



Find the name of sailors who've reserved all boats

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

- Uses **division**; schemas of the input relations to/ must be carefully chosen:

$$\rho (Tempsids, (\pi_{sid,bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$\dots / \pi_{bid} (\sigma_{bname = 'Interlake'} Boats)$$

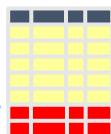
Duke CS, Fall 2024

53



Monotone operators

Add more rows
to the input...



RelOp



What happens
to the output?

- If we add more rows to the input... What happens to the output?
- An operator is monotone if adding rows to the input never removes existing output rows.
- An operator is non-monotone if adding rows to the input may cause some old output rows to disappear.

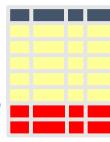
Duke CS, Fall 2024

54



Monotone operators

Add more rows
to the input...



What happens
to the output?

- If old results always remain valid when more data is added, the operator is **monotone**.

Example: Union, Join, Selection.

- Adding data only increases (or keeps) the results. Old answers are never invalidated.
- Formal definition, for a monotone operator op :

$$\text{If } R \subseteq R' \text{ then } op(R) \subseteq op(R') \quad \text{for any } R, R'$$



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”?

Monotone queries:

- If you add more rows, the old results stay correct.
- Example: simple selections or joins.

Is “highest” monotone?

- No!
- Suppose the current highest price = 3.0.
- If we add a new row with price = 3.01,
The old answer (3.0) is wrong.
- So it must use a difference!



Why do we need each core relational operator (X)?

Difference ($R - S$)

- It is the **only non-monotone operator**.
- Adding new rows to the input can remove results (because rows might now be subtracted).
- That's why we can't build it from only monotone operators.



Why do we need each core relational operator (X)?

Projection ($\pi_L R$)

- It is the **only operator that removes columns**.
- Other operators only combine or filter rows, but don't reduce attributes.
- Without projection, we couldn't simplify a relation to fewer attributes.



Why do we need each core relational operator (X)?

Cross product ($R \times S$)

- It is the **only operator that adds new columns**.
- Combining attributes from two relations creates a wider schema.



Why do we need each core relational operator (X)?

Union ($R \cup S$)

- It is the **only operator that allows you to add rows** from another relation.
- It's the building block for combining datasets.



Why do we need each core relational operator (X)?

Selection ($\sigma_p R$)

- It is the only operator that **removes rows** based on a condition.
- It's the core operator for **filtering tuples** in a relation.



Extensions to relational algebra

- Duplicates → Bag algebra (handle repeated tuples)
- Grouping & Aggregation → SUM, COUNT, AVG, etc.
- Extended Projection → Compute new column values



Why is RA a good query language?

- Simple:
 - A few core operators
 - Easy-to-understand semantics
- Declarative?
 - More declarative than older models (e.g., CODASYL)
 - But combining operators can feel somewhat “procedural”



Why is RA a good query language?

- Complete
 - Expressive enough to represent all basic relational queries
 - (Completeness depends on the formal definition used)



Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{<\text{selection condition}>} (R)$
PROJECT	Produces a new relation with only some of the attributes of R and removes duplicate tuples.	$\pi_{<\text{attribute list}>} (R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{(<\text{join attributes } 2>)} R_2$, $(<\text{join attributes } 2>) R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, $\text{OR } R_1 *_{(<\text{join attributes } 2>)} R_2$, $R_2 \text{ OR } R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$



International University, VNU-HCMC

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Thank you for your attention!

Duke CS, Fall 2024

67