CSE 412 Database Management

Dr. Bharatesh Chakravarthi (BC)

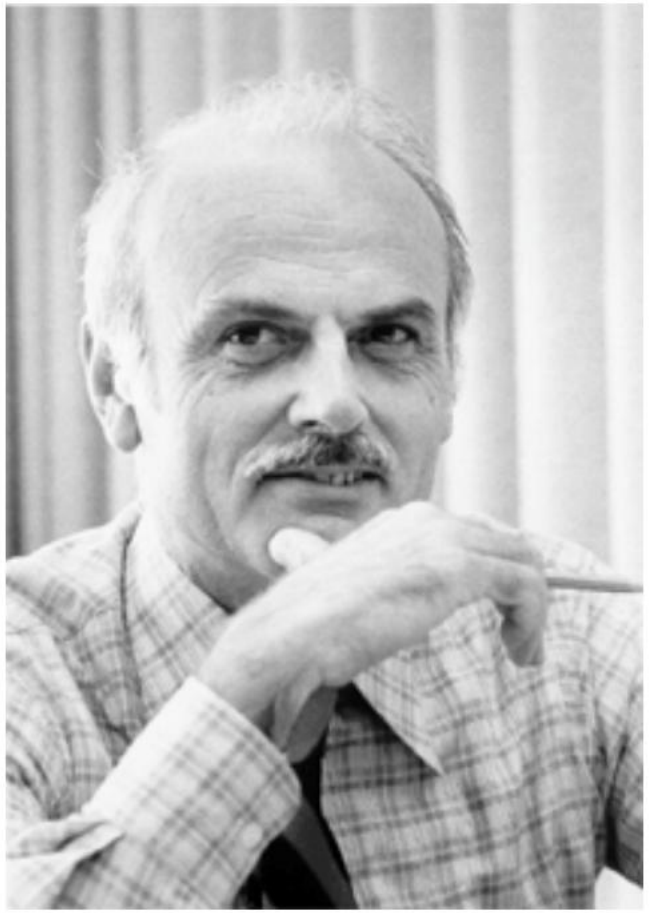
Assistant Teaching Professor

School of Computing and Augmented Intelligence

Arizona State University

Formal query languages

- How do we collect information?
  - Eg., find ssn's of students in CSE 412
- (Let's say: everything is a set!)
- One solution: Rel. algebra, ie., set operators
  - Q1: Which ones??
  - Q2: What is a minimal set of operators?



Edgar F. Codd proposed relational model and Relational algebra in 1970s

Relational Operators

Fundamental Operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

Derived Operators

- Intersection: $R \cap S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

Relational Operators

Fundamental Operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

Derived Operators

- Intersection: $R \cap S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

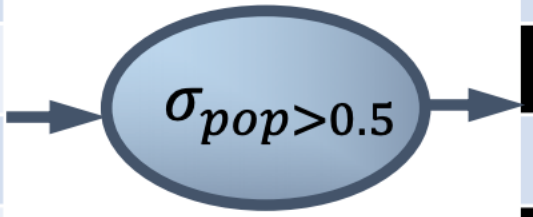
Compose operators to make complex queries.

Selection Example

- Users with popularity higher than 0.5

$$\sigma_{pop>0.5} User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…



uid	name	age	pop
142	Bart	10	0.9
857	Lisa	8	0.7
…	…	…	…

Selection Example

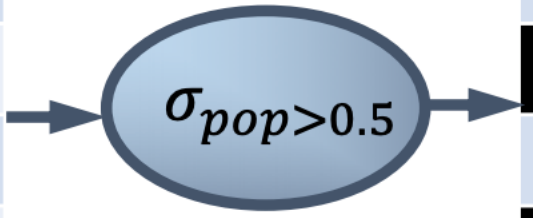
- Users with popularity higher than 0.5

$$\sigma_{pop>0.5} User$$

Corresponding SQL:

Select * FROM User
WHERE pop > 0.5

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…



uid	name	age	pop
142	Bart	10	0.9
857	Lisa	8	0.7
…	…	…	…

Selection

- Input: a table R
- Notation: $\sigma_p R$
  - *P is called as 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

More on Selection

- Selection condition can include any column of R,
  - constants, comparison (= , ≤, etc.) and
  - Boolean connectives (∧: and, ∨: or, ¬: not)
- Example: users with popularity at least 0.9 **and** age under 10 ***or*** above 12

$$\sigma_{pop \geq 0.9 \wedge (age < 10 \vee age > 12)} User$$

More on Selection

- Selection condition can include any column of R,
  - constants, comparison (= , ≤, etc.) and
  - Boolean connectives (∧: and, ∨: or, ¬: not)
- Example: users with popularity at least 0.9 and age under 10 or above 12

$$\sigma_{pop \geq 0.9 \wedge (age < 10 \vee age > 12)} User$$

Corresponding SQL:

Select * FROM User WHERE pop > 0.5 AND (age < 10 OR age > 12)

More on Selection

- You must be able to evaluate the condition over each single row of the input table!
- Example: the most popular user

$$\sigma_{pop \geq every\ pop\ in\ User} User$$

WRONG!

Correct Form in SQL could be:
SELECT *
FROM User
WHERE pop = (SELECT MAX(pop) FROM User)

Relational Operators

Fundamental Operators

- ~~Selection:~~ $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

Derived Operators

- Intersection: $R \cap S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

Compose operators to make complex queries.

Projection Examples

- IDs and names of all users

$$\pi_{uid,name}\ User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

uid	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

Projection Examples

Corresponding SQL:

Select UID NAME
FROM User

- IDs and names of all users

$$\pi_{uid,name}\ User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

uid	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

More on projection

- Duplicate output rows are removed (by definition)
- Example: user ages

$$\pi_{age}\, User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

age
10
8
…

More on projection

Corresponding SQL:

Select DISTINCT AGE FROM User

- Duplicate output rows are removed (by definition)
- Example: user ages

$$\pi_{age}\, User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

age
10
8
…

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

Test

- What does this mean:

$$\sigma_{pop > 0.5}(\pi_{name,\ age})User$$

User

uid	*name*	*age*	*pop*
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Test

- How to translate this to SQL?

$$\sigma_{pop > 0.5}(\pi_{name,\ age})User$$

User

uid	*name*	*age*	*pop*
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Test

- How to translate this to SQL?

$$\sigma_{pop > 0.5}(\pi_{name,\ age})User$$

User

uid	*name*	*age*	*pop*
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

```
SELECT Name, Age
FROM User
WHERE pop > 0.5
```

Relational Operators

Fundamental Operators

- ~~Selection:~~ $\sigma_p R$
- ~~Projection:~~ $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

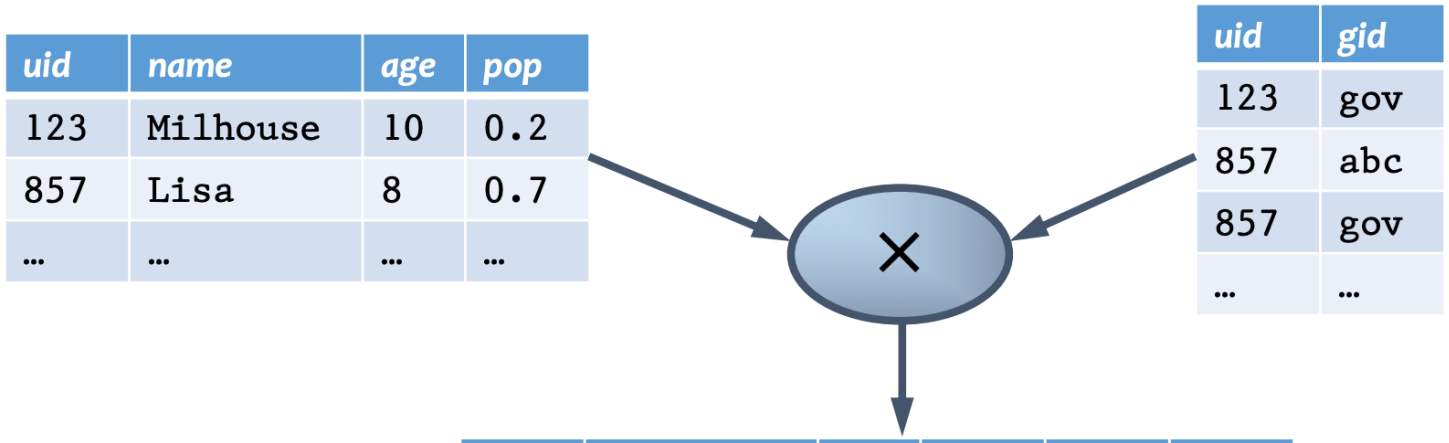
Derived Operators

- Intersection: $R \cap S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

Compose operators to make complex queries.

Cross product example

$User \times Member$



uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

×

uid	gid
123	gov
857	abc
857	gov
…	…

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Cross product example

$User \times Member$

uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

×

uid	gid
123	gov
857	abc
857	gov
…	…

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Corresponding SQL:

Select *

FROM User, Member

Cross product

- Input: two tables R and S
- Notation: R X S
- **Purpose: pairs of 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)*

A note on column/row ordering

- Ordering of columns/rows is unimportant as far as contents are concerned

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

=

uid	gid	uid	name	age	pop
123	gov	123	Milhouse	10	0.2
857	abc	123	Milhouse	10	0.2
857	gov	123	Milhouse	10	0.2
123	gov	857	Lisa	8	0.7
857	abc	857	Lisa	8	0.7
857	gov	857	Lisa	8	0.7
…	…	…	…	…	…

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

Relational Operators

Fundamental Operators

- ~~Selection:~~ $\sigma_p R$
- ~~Projection:~~ $\pi_L R$
- ~~Cross product:~~ $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

Derived Operators

- Intersection: $R \cap S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

Compose operators to make complex queries.

Test

- What does this mean:

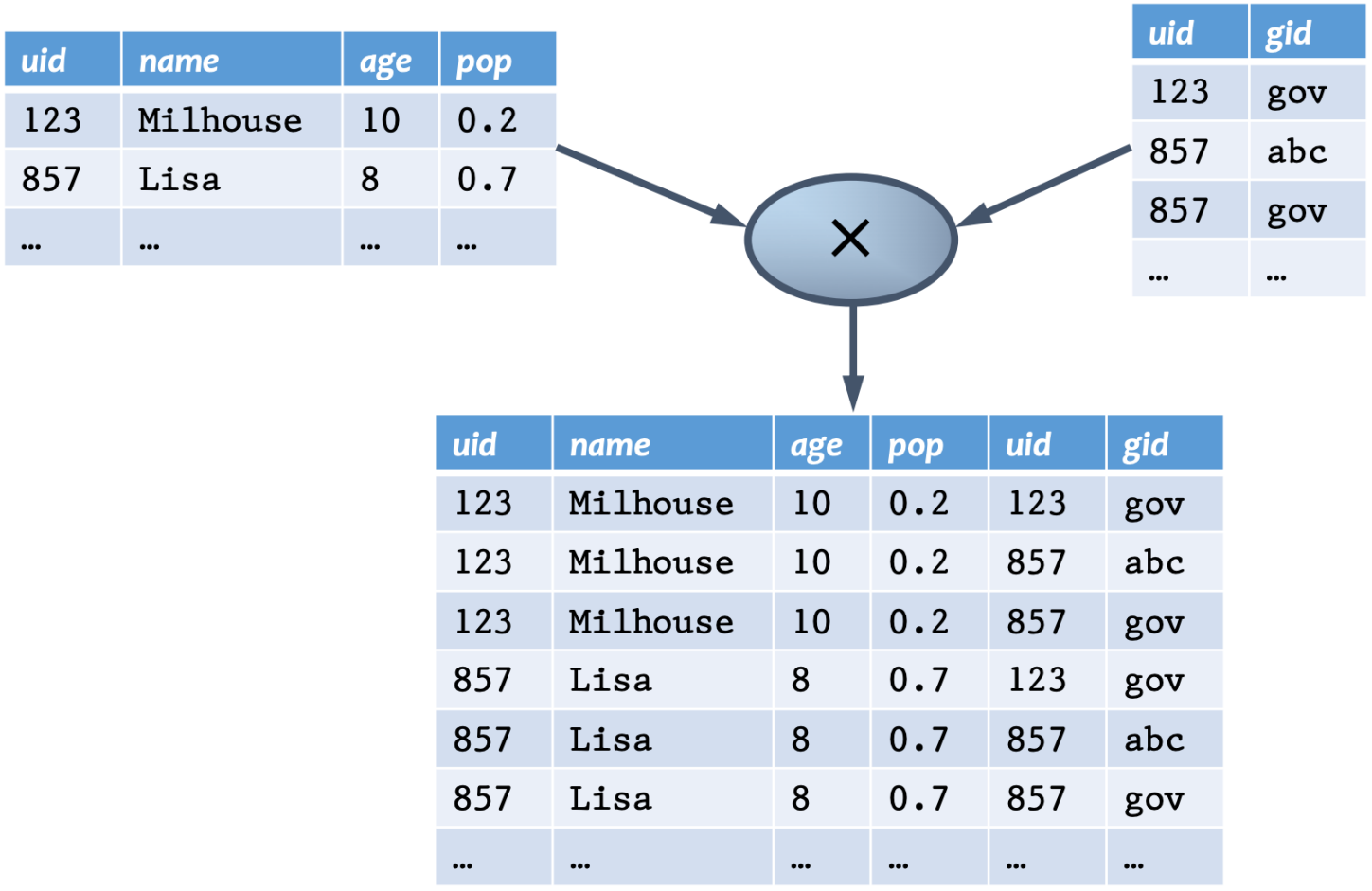
$$\sigma_{User.uid = Member.uid}(User \times Member)$$

Test

- What does this mean:

$$\sigma_{User.uid = Member.uid}(User \times Member)$$

$User \times Member$



uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

uid	gid
123	gov
857	abc
857	gov
…	…

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Test

- What does this mean:

$$\sigma_{User.uid = Member.uid}(User \times Member)$$

$User \times Member$

uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

×

uid	gid
123	gov
857	abc
857	gov
…	…

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

$\sigma_{User.uid = Member.uid}$

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
[redacted]					
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Test

- What does this mean:

$$\sigma_{User.uid = Member.uid}(User \times Member)$$

It returns you the group information for each user

$User \times Member$

uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

×

uid	gid
123	gov
857	abc
857	gov
…	…

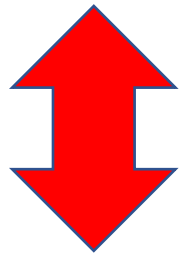
uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

$\sigma_{User.uid = Member.uid}$

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
[redacted]					
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Because this query pattern is so popular, a new operator is derived from cross product (X) and selection (*σ), called join (⋈*)

$$\sigma_{User.uid = Member.uid}(User \times Member)$$



$$User \bowtie_{User.uid = Member.uid} Member$$

Derived operator: Join

- (a.k.a. “theta-join”)
- 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 rows *s* in S, output *rs*, if *r* and *s* satisfy *p*
- *Shorthand for* σ_p (R X S)

Join example

- Info about users, plus IDs of their groups

$$User \bowtie_{User.uid=Member.uid} Member$$

uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

uid	gid
123	gov
857	abc
857	gov
…	…

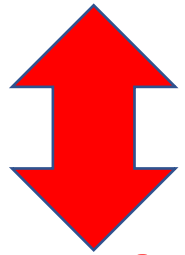
Prefix a column reference with table name and "." to disambiguate identically named columns from different tables

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
[redacted]					
[redacted]					
[redacted]					
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

Corresponding SQL 1:

Select * **FROM** User, Member **WHERE** User.uid = Member.uid

$$\sigma_{User.uid = Member.uid}(User \times Member)$$



$$User \bowtie_{User.uid = Member.uid} Member$$

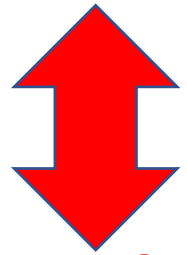
Corresponding SQL 2:

Select * **FROM** User JOIN Member ON User.uid = Member.uid

Corresponding SQL 1:

Select * **FROM** User, Member **WHERE** User.uid = Member.uid

$$\sigma_{User.uid = Member.uid}(User \times Member)$$



$$User \bowtie_{User.uid = Member.uid} Member$$

Both Correct!

Corresponding SQL 2:

Select * **FROM** User JOIN Member ON User.uid = Member.uid

Relational Operators

Fundamental Operators

- ~~Selection:~~ $\sigma_p R$
- ~~Projection:~~ $\pi_L R$
- ~~Cross product:~~ $R \times S$
- Union: $R \cup S$
- Difference: $R - S$

Derived Operators

- Intersection: $R \cap S$
- ~~Join:~~ $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Renaming: $\rho_{S(A_1,A_2,\dots)} R$

Compose operators to make complex queries.