Lecture 3: Relational Model

CS348 Spring 2025:
Introduction to Database Management

Instructor: **Xiao Hu**

Sections: 001, 002, 003

Announcements

- Assignment 1 released on Learn
  - Due May 29
- Milestone 0 (Team up) for project
  - Due May 22
  - See Piazza: how to enroll in a project group on LEARN
- Supplementary materials for the course project released on Learn
  - DB2 Sample Application
  - MySQL Sample Application

Outline

- More examples of relational algebra query
- Monotone operators
- Relational calculus (optional)

(Recap) 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 unique name and a domain (or type)
  - The domains are required to be **atomic**

- Each relation contains a set of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed

☞Simplicity is a virtue!

(Recap) Integrity constraints

- Key (Candidate key)
  - A set of attributes that uniquely identify a row, and that is also minimal
  - A key can contain multiple attributes
  - A relation can have multiple keys
- Primary key
  - a designated candidate key in the schema declaration
  - underline all attributes
- Foreign key
  - primary key of one relation appearing as an attribute of another relation

User

<u>uid</u>	name	age	pop
857	Lisa	8	0.7
...	...	...	...

Group

<u>gid</u>	name
dps	Dead Putting Society
...	...

Member

<u>uid</u>	<u>gid</u>
857	dps
...	...

(Recap) RA operators

Core Operators

1. Selection: $\sigma_p R$
2. Projection: $\pi_L R$
3. Cross product: $R \times S$
4. Union: $R \cup S$
5. Difference: $R - S$
6. Renaming: $\rho_{S(A_1 \to A_1', A_2 \to A_2', \ldots)} R$

Derived Operators

1. Join: $R \bowtie_p S$
2. Natural join: $R \bowtie S$
3. Intersection: $R \cap S$

(Recap) Example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

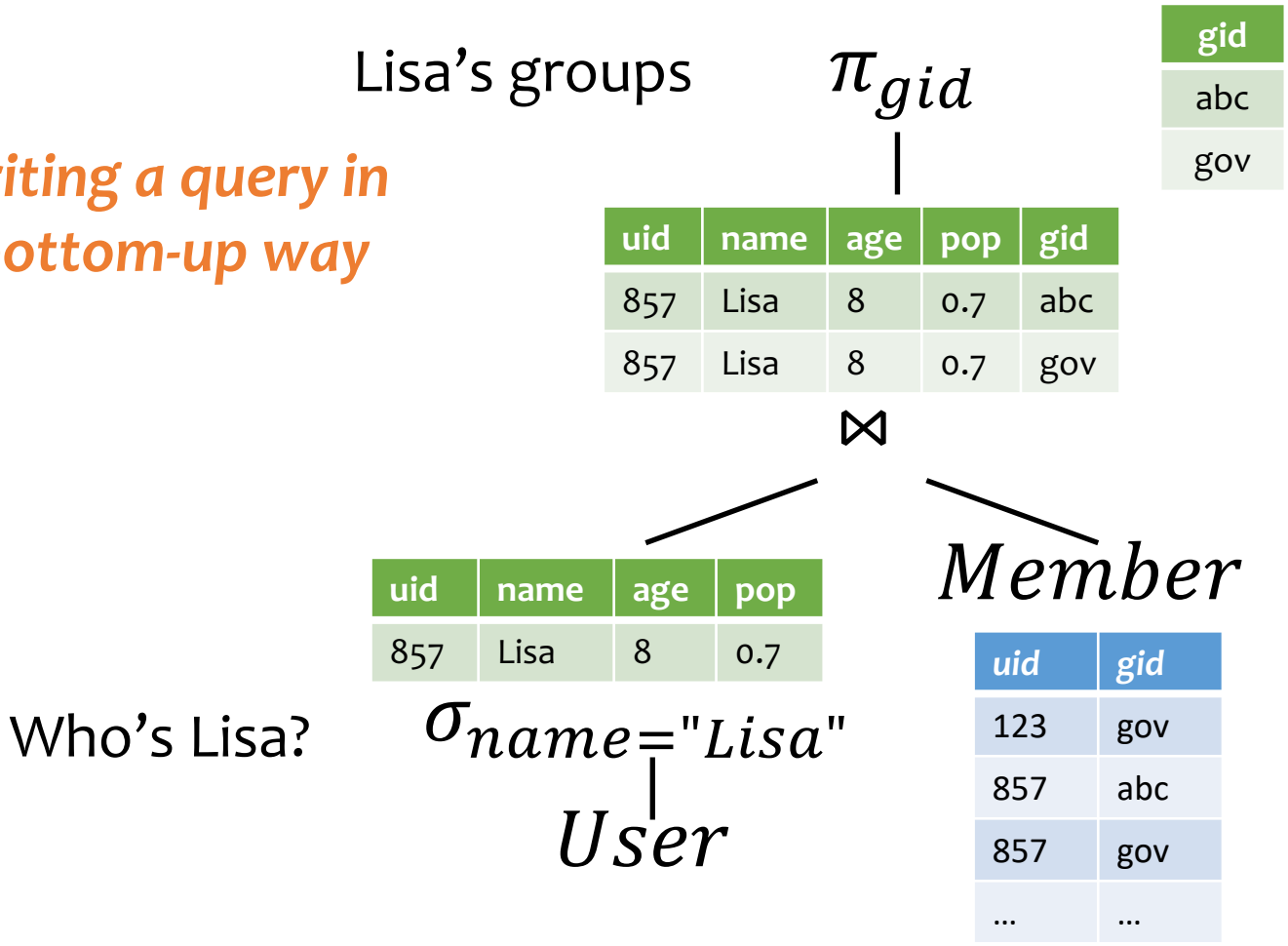
- All groups (ids) that Lisa (a user's name) belongs to

(Recap) Example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- All groups (ids) that Lisa (a user's name) belongs to

Writing a query in a bottom-up way



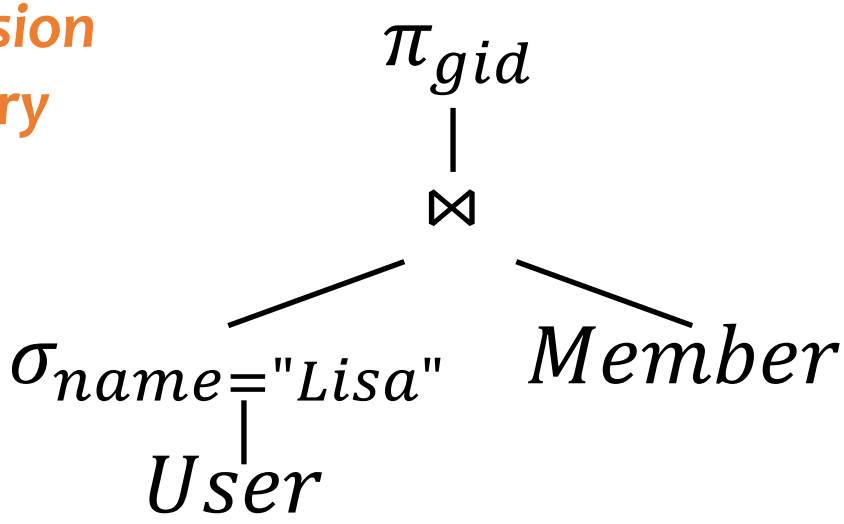
(Recap) Example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- All groups (ids) that Lisa (a user's name) belongs to

$$\pi_{gid}((\sigma_{name="Lisa"}\ User) \bowtie Member)$$

Putting an expression tree back to a query

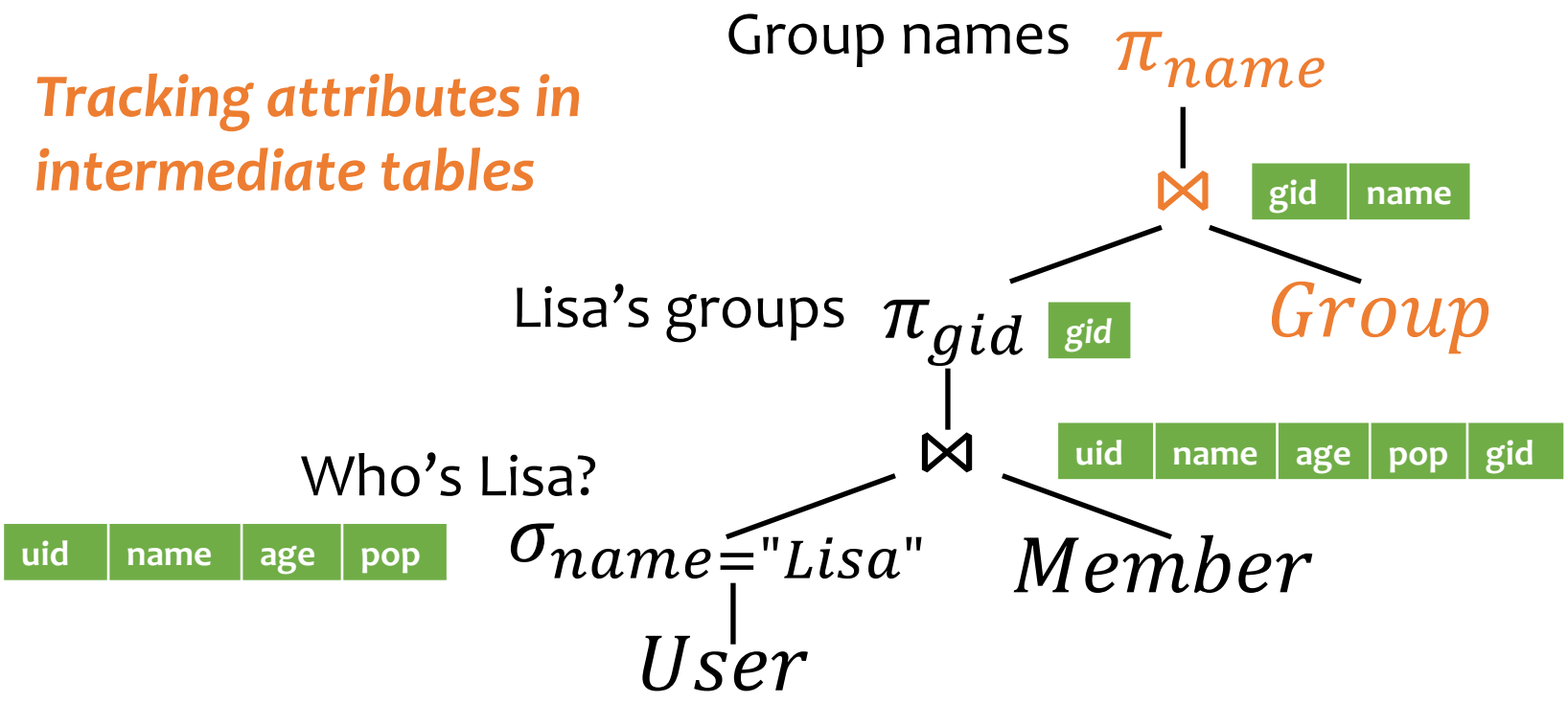


More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- All groups ~~(ids)~~ that Lisa (a user's name) belongs to names?

Tracking attributes in intermediate tables

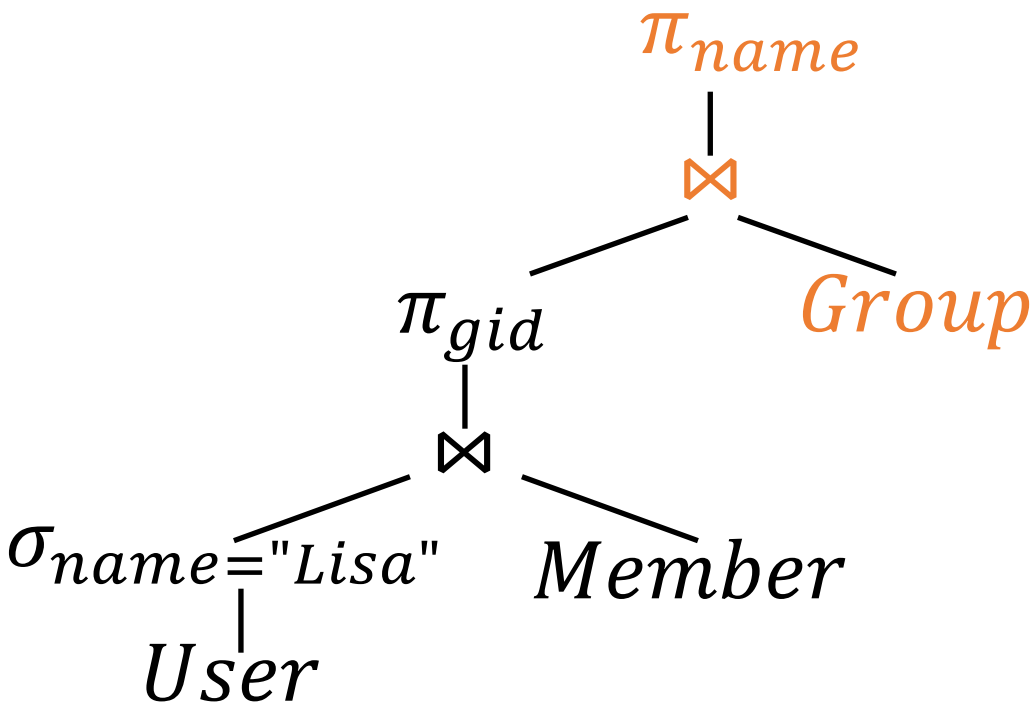


More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- All group names that Lisa belongs to

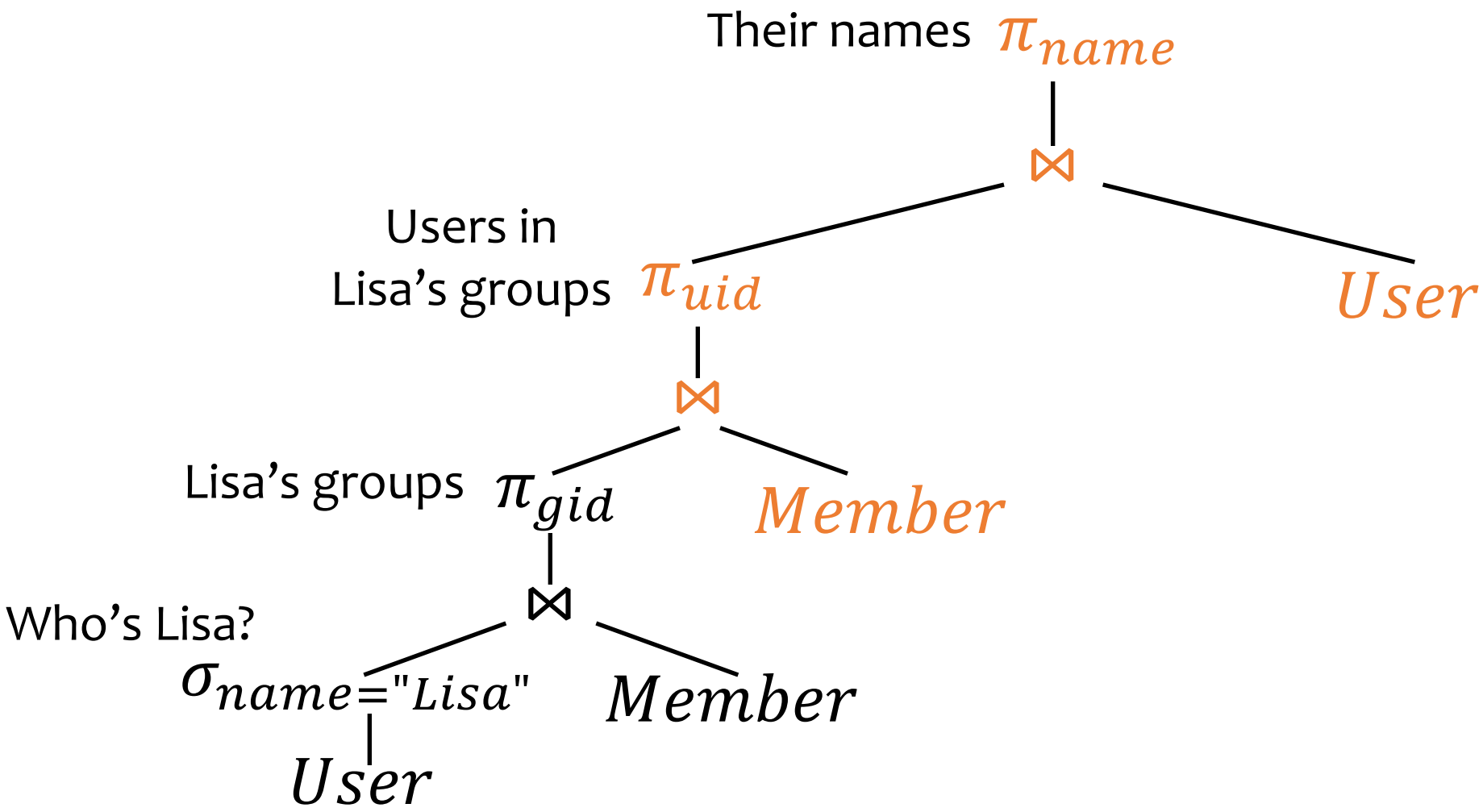
$$\pi_{name}\left(\pi_{gid}\left((\sigma_{name="Lisa"}User) \bowtie Member\right) \bowtie Group\right)$$



More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

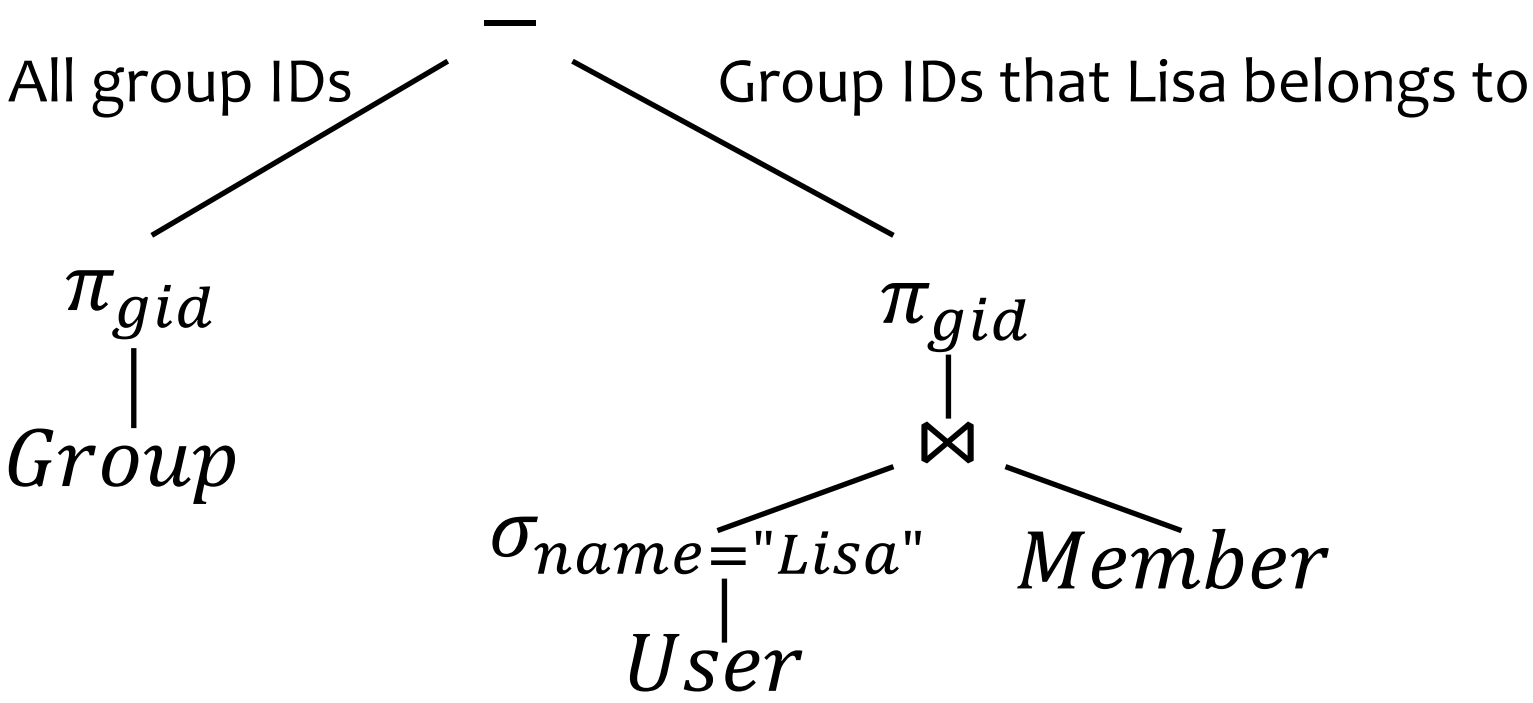
- Names of users in Lisa's groups



More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- IDs of groups that Lisa doesn't belong to



$$\left(\pi_{gid} Group\right) - \left(\pi_{gid}\left(\left(\sigma_{name="Lisa"} User\right) \bowtie Member\right)\right)$$

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)ç

- IDs of users who belong to at least two groups

WRONG!

$$\pi_{uid}\left(Member \bowtie_{\substack{Member.uid=Member.uid \wedge \\ Member.gid \neq Member.gid}} Member\right)$$

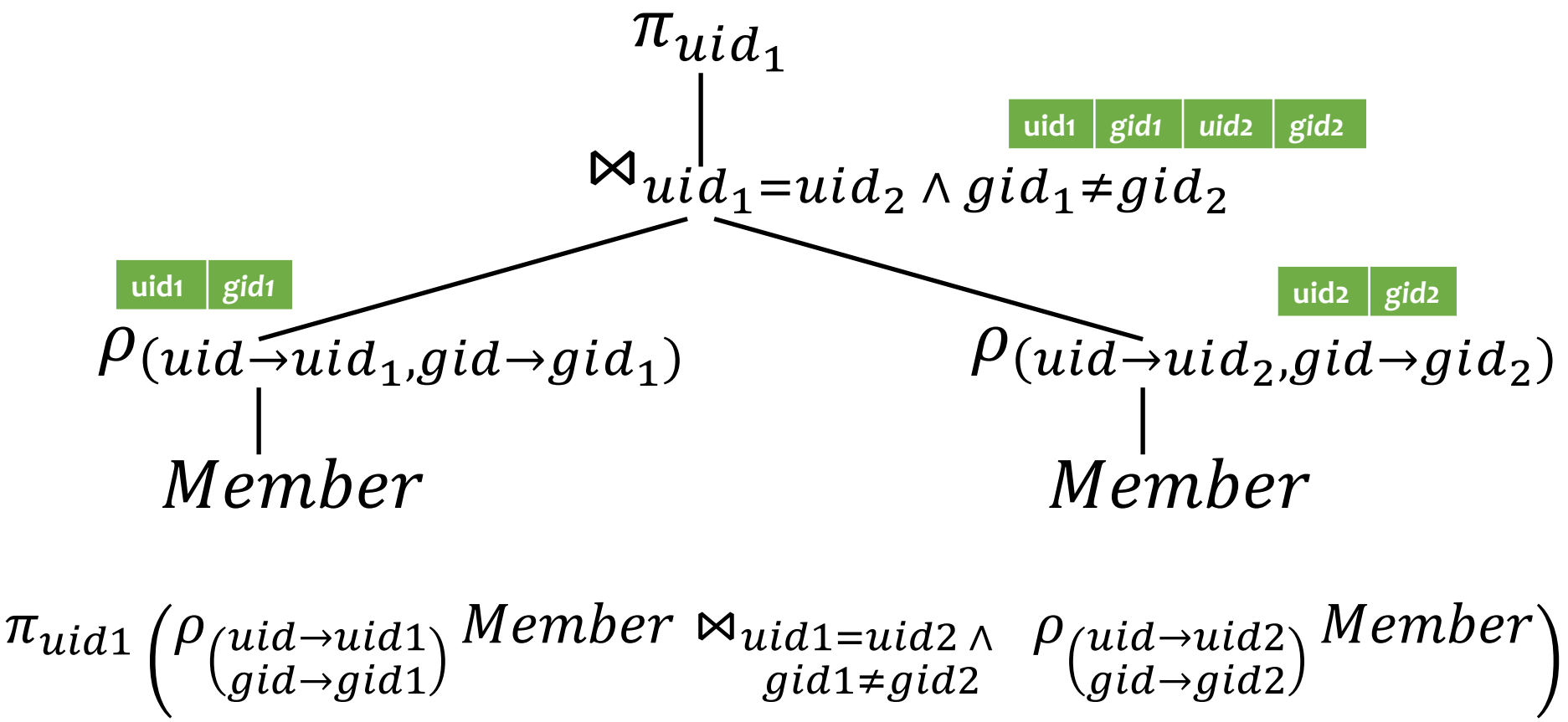
- Because it cannot distinguish two Member tables

☞Renaming can be used to avoid confusion caused by identical column name!

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)ç

- IDs of users who belong to at least two groups

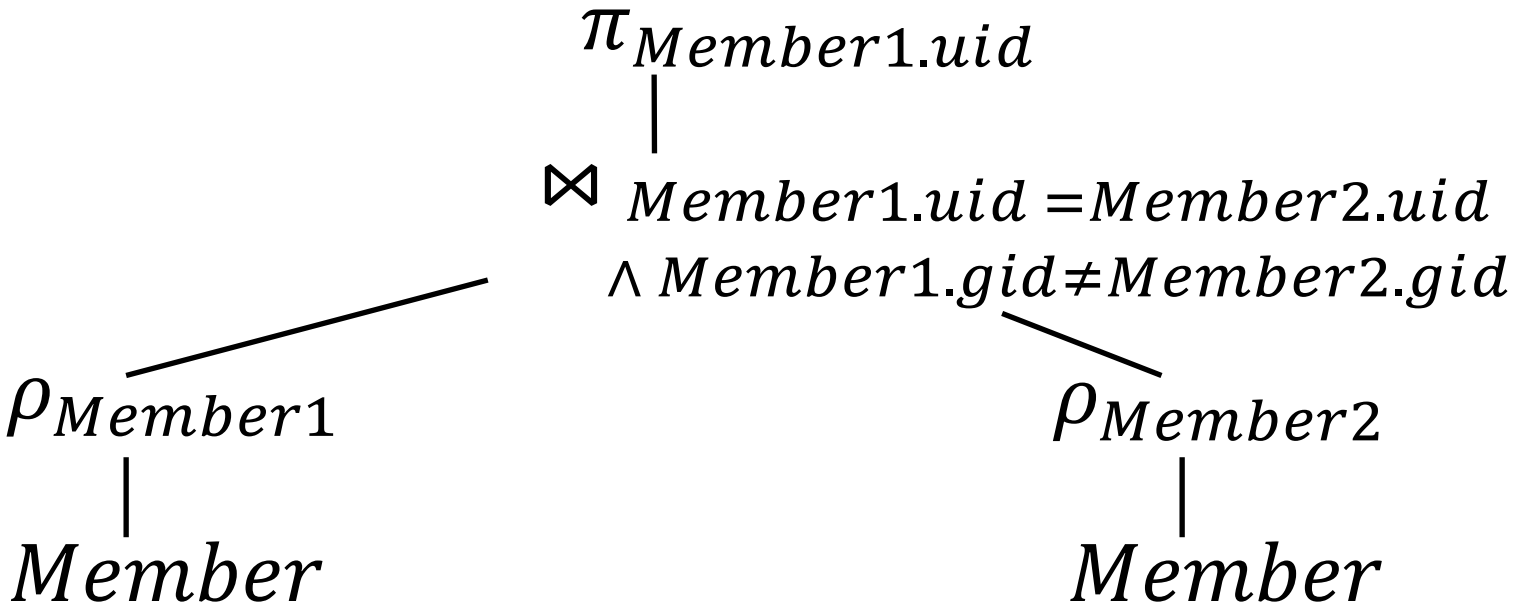


$$\pi_{uid1}\left(\rho_{\substack{uid\to uid1\\ gid\to gid1}} Member \bowtie_{\substack{uid1=uid2 \wedge\\ gid1\neq gid2}} \rho_{\substack{uid\to uid2\\ gid\to gid2}} Member\right)$$

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)ç

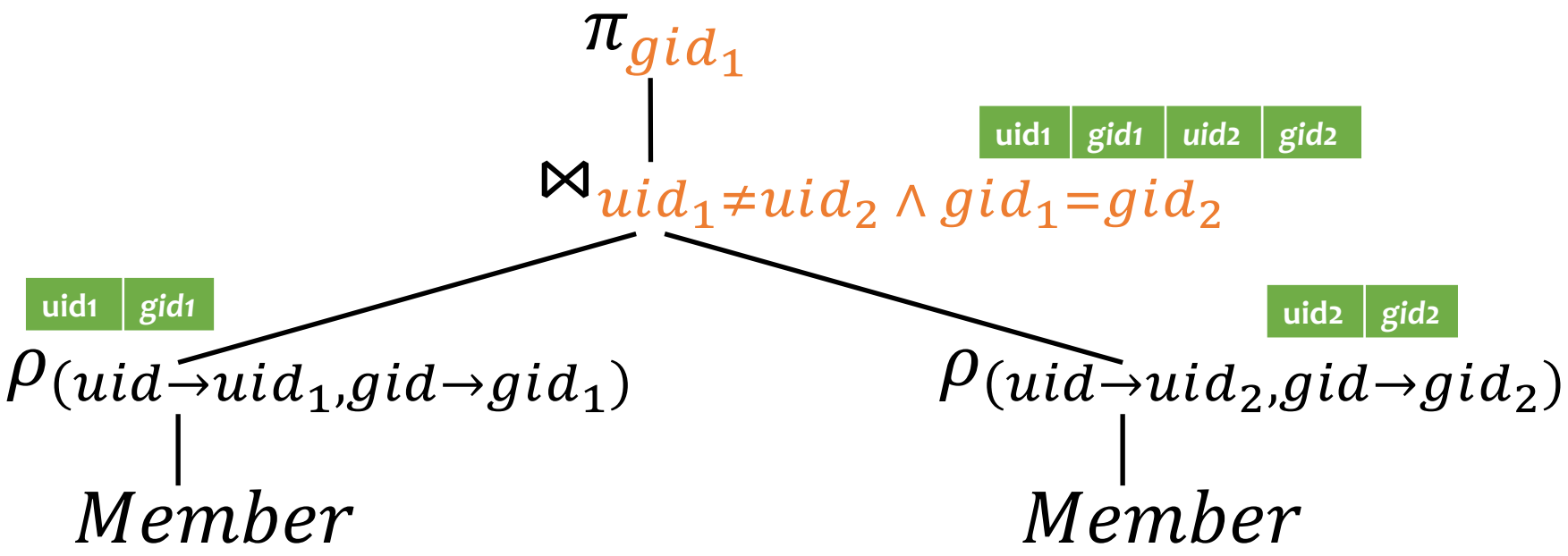
- IDs of users who belong to at least two groups



More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)ç

- IDs of groups that have at least two users



Post-lecture

- Write a relational algebra query only using:
  - Input tables
  - Relational operators (core & derived)
  - Constants in natural language queries (e.g. "$Lisa$")

- Should not depend on the database content

Take home exercise

- IDs of users who belong to at least three groups

- IDs of users who belong to exactly two groups

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Who are the most popular users?

$$\sigma_{pop \geq every\ pop\ in\ User}\ User$$ **WRONG!**

- Because it cannot be evaluated over a single row

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

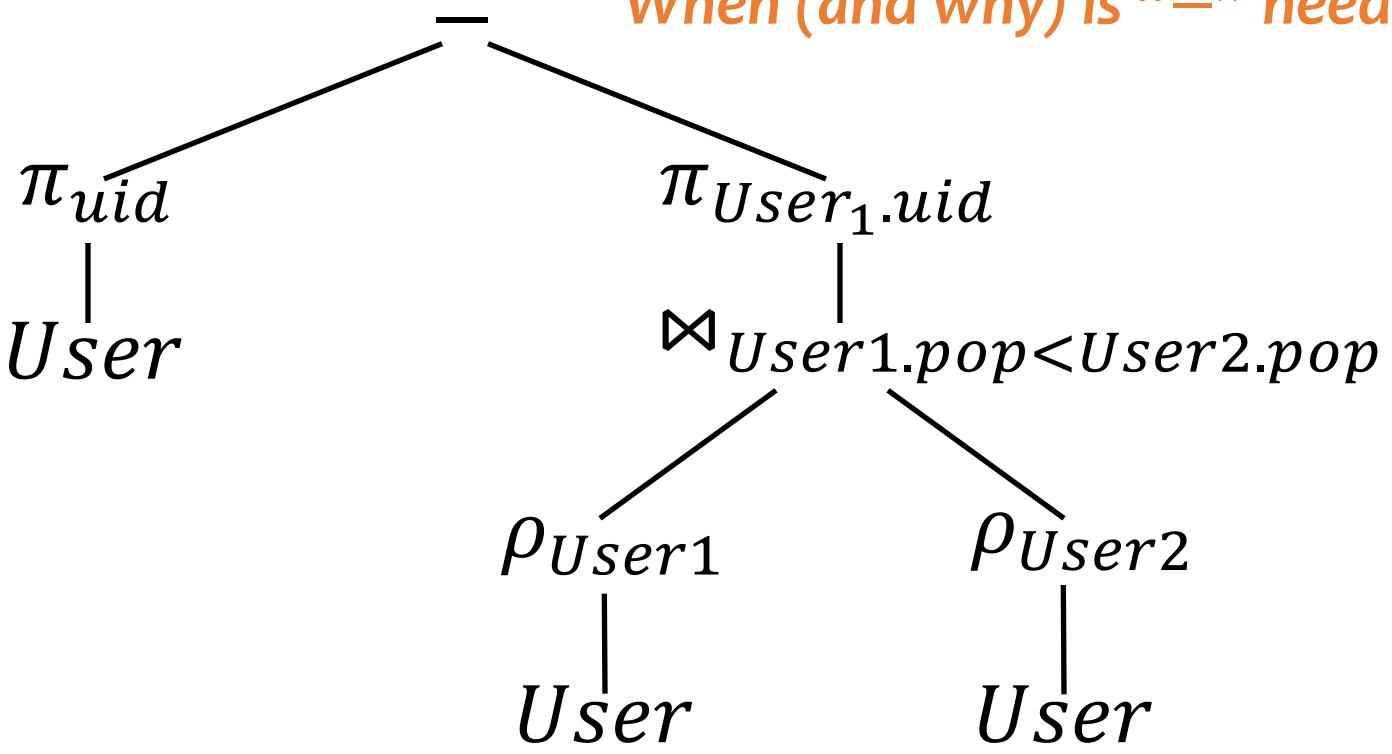
- Who are the most popular users?
  - Who does NOT have the highest *pop* rating?
  - Whose *pop* is lower than somebody else's?

More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Who are the most popular users?
  - Who does NOT have the highest *pop* rating?
  - Whose *pop* is lower than somebody else's?

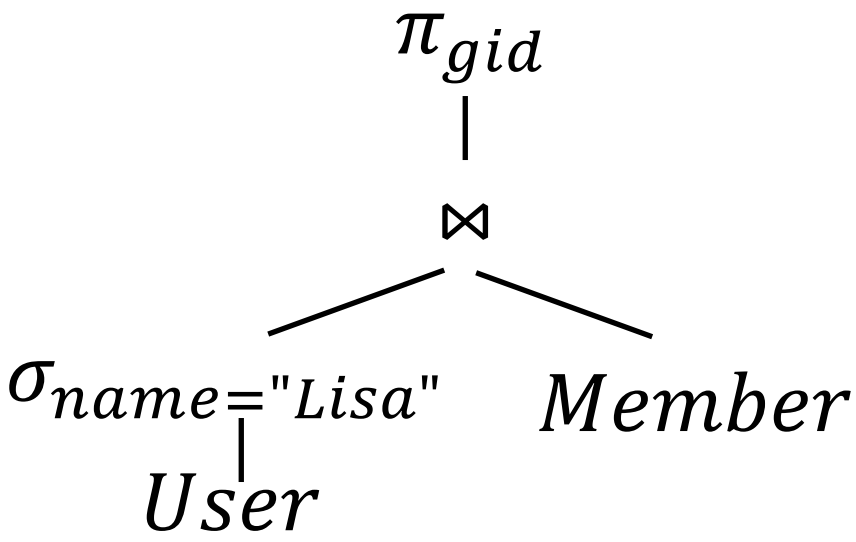
When (and why) is "–" needed?



More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Who joins all the groups that Lisa joins?
  - All groups (ids) that Lisa belongs to Suppose as *S(gid)*

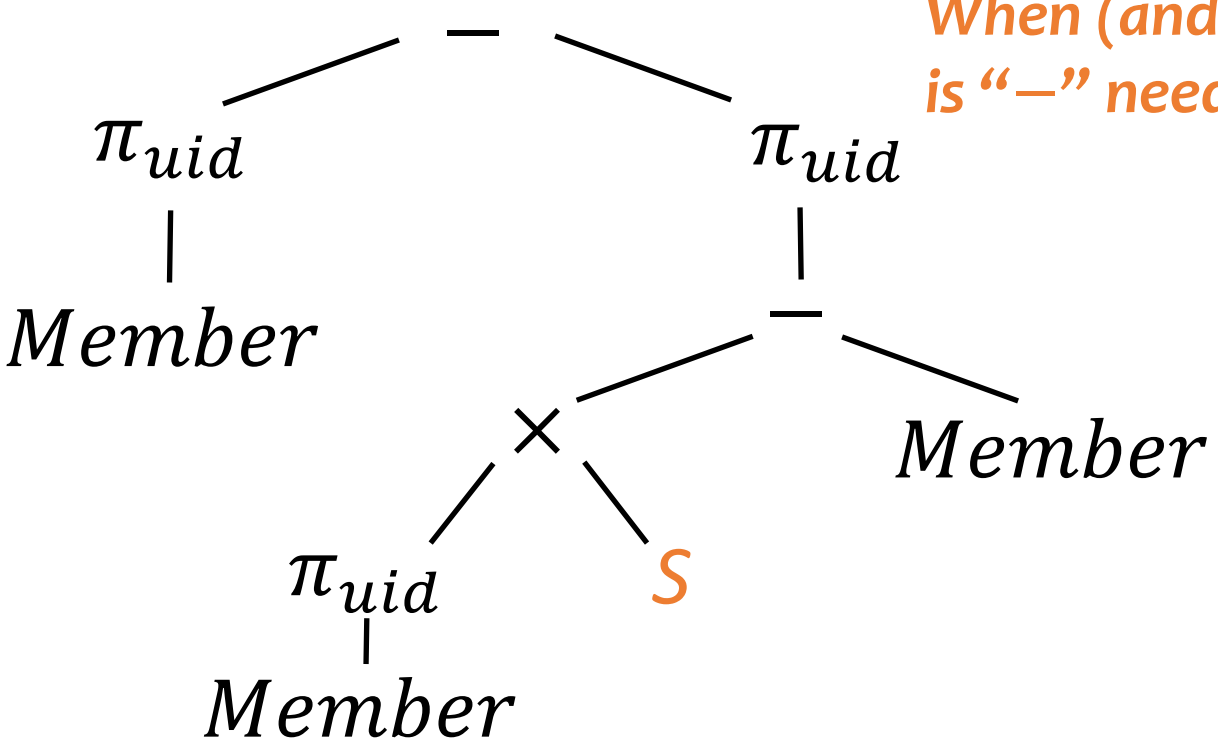


More example

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Who joins all the groups that Lisa joins?
  - All groups (ids) that Lisa belongs to — Suppose as *S(gid)*
  - Who joins all groups in *S*?
  - *Who does not join some group in S?*

When (and why) is "—" needed?

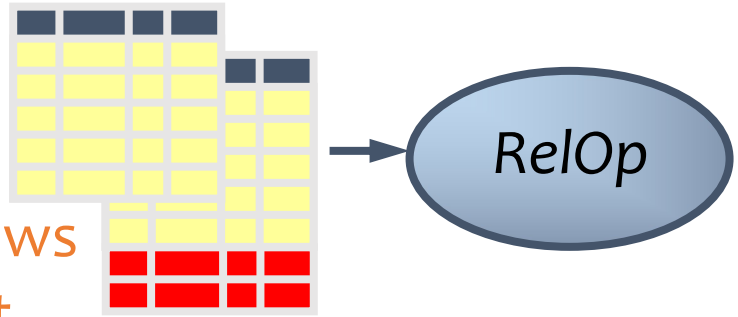


Composite operator: Division

- Input: a table $R(A, B), S(A)$
- Notation: $R \div S$
- Output:
  - A relation defined on B (attributes in R but not in S)
  - A tuple b is an output if
    - for every tuple a in S, (a, b) is a tuple in R
- Shorthand for $\pi_B R - \pi_B((\pi_B R) \times S - R)$
- Example: $Member(uid, gid) \div S(gid)$

$$\pi_{uid} Member - \pi_{uid}((\pi_{uid} Member) \times S - Member)$$

Non-monotone operators



Add more rows to the input...

What happens to the output?

- If some old output rows may become invalid, and need to be removed → the operator is non-monotone

- Example: difference operator $R - S$

uid	gid
123	gov
857	abc

R

—

uid	gid
123	gov
901	edf
857	**abc**

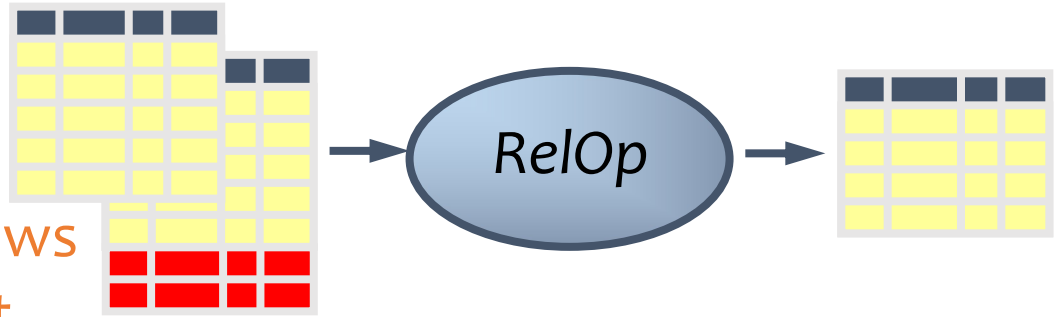
S

→

uid	gid
~~857~~	~~abc~~

This old row becomes invalid because the new row added to S

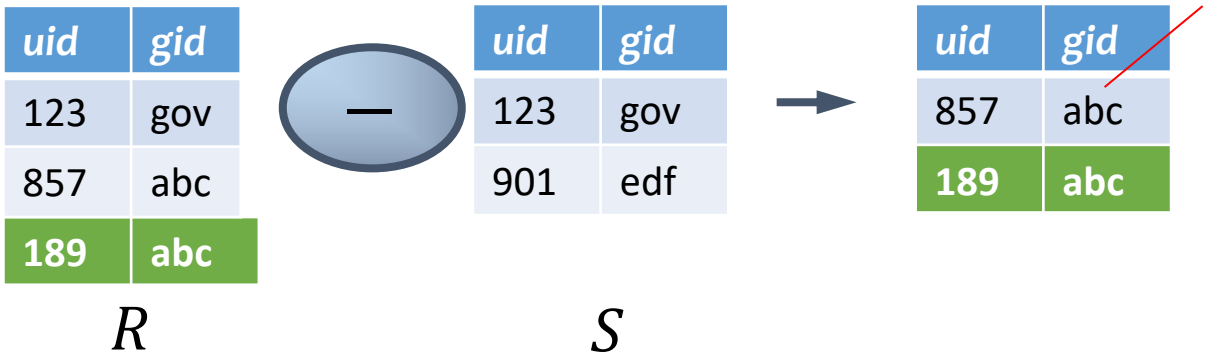
Non-monotone operators



Add more rows to the input...

What happens to the output?

- If some old output rows may become invalid, and need to be removed → the operator is non-monotone
- Otherwise (old output rows always remain "correct") → the operator is monotone



Classification of relational operators

- 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

Composition of operators

- The composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input

- The composition of monotone and non-monotone operators could produce a monotone or non-monotone query
  - $R - (R - S) = R \cap S$ is monotone w.r.t. both R and S
  - $\pi_B R - \pi_B((\pi_B R) \times S - R)$ is non-monotone w.r.t. S and monotone w.r.t. R

Why is "—" needed?

- Is the "highest" query monotone? No!
  - Current highest *pop* is 0.9
  - Add another row with *pop* 0.91
  - Old answer is invalidated

- Is the "division" query monotone? No!
  - Suppose Lisa only joins gov; all users belonging to gov are valid
  - Add another row with Lisa joining dps
  - Users who only joins gov but not dps are invalidated

☞So non-monotone queries must use difference!

Why do we need core operator X?

- Difference
  - The only non-monotone operator
- Projection
  - The only operator that removes columns
- Cross product
  - The only operator that adds columns
- Union
  - The only operator that adds rows (w/o changing schema)
- Selection
  - The only operator that removes rows (w/o changing schema)
- Renaming
  - The only operator that changes the name of columns

Extensions of relational algebra

- Duplicate handling ("bag semantics")
- Grouping and aggregation

☞ All these will come up when we talk about SQL

☞ But for now we will stick to standard relational algebra without these extensions

Relational calculus (Optional)

- Relational Algebra: **procedural** language
  - An algebraic formalism in which queries are expressed by applying a sequence of operations to relations
- Relational Calculus: **declarative** language
  - A logical formalism in which queries are expressed as formulas of first-order logic
- Codd's Theorem: Relational Algebra and Relational Calculus are essentially equivalent in terms of expressive power.

Relational calculus (Optional)

- Use *first-order logic (FOL) formulae* to specify properties of the query answer

- Example: Who are the most popular?
  - $\{u.uid \mid u \in User \wedge \neg(\exists u' \in User: u.pop < u'.pop)\}$, or
  - $\{u.uid \mid u \in User \wedge (\forall u' \in User: u.pop \geq u'.pop)\}$

User (<u>*uid* </u>int, *name* string, *age* int, *pop* float)
Group (<u>*gid*</u> string, *name* string)
Member (<u>*uid*</u> int, <u>*gid*</u> string)

Relational calculus (Optional)

- Relational algebra = "safe" relational calculus
  - Every query expressible as a safe relational calculus query is also expressible as a relational algebra query
  - And vice versa

- Example of an "unsafe" relational calculus query
  - $\{u.name \mid \neg(u \in User)\}$ → users not in the User table
  - Cannot evaluate it just by looking at the database

- A query is *safe* if, for all database instances conforming to the schema, the query result can be computed using only constants appearing in the database instance or in the query itself.

Limits of relational algebra (Optional)

- Relational algebra has no recursion
  - Example: given relation *Friend*(*uid1*, *uid2*), who can Bart reach in his social network with any number of hops?
    - Writing this query in r.a. is impossible!
  - So it is not as powerful as general-purpose languages

- But why not?
  - Optimization becomes undecidable

  ☞Simplicity is empowering
  - Besides, you can always implement it at the application level, and recursion is added to SQL nevertheless!

Applications of relational algebra (Optional)

- What are the tables or relations?
- How to express the query in relational algebra?

How to list all triangles in a directed graph?

How to list all distinct pairs of users sharing some common friends?

How to partition users so that no neighbors fall into the same group?

Graph analytics (Optional)

Edge

source	destination
Alice	Bob
Alice	Cathy
Bob	David
David	Alice
...	...

Edge1

A	B
Alice	Bob
Alice	Cathy
Bob	David
David	Alice
...	...

Edge2

B	C
Alice	Bob
Alice	Cathy
Bob	David
David	Alice
...	...

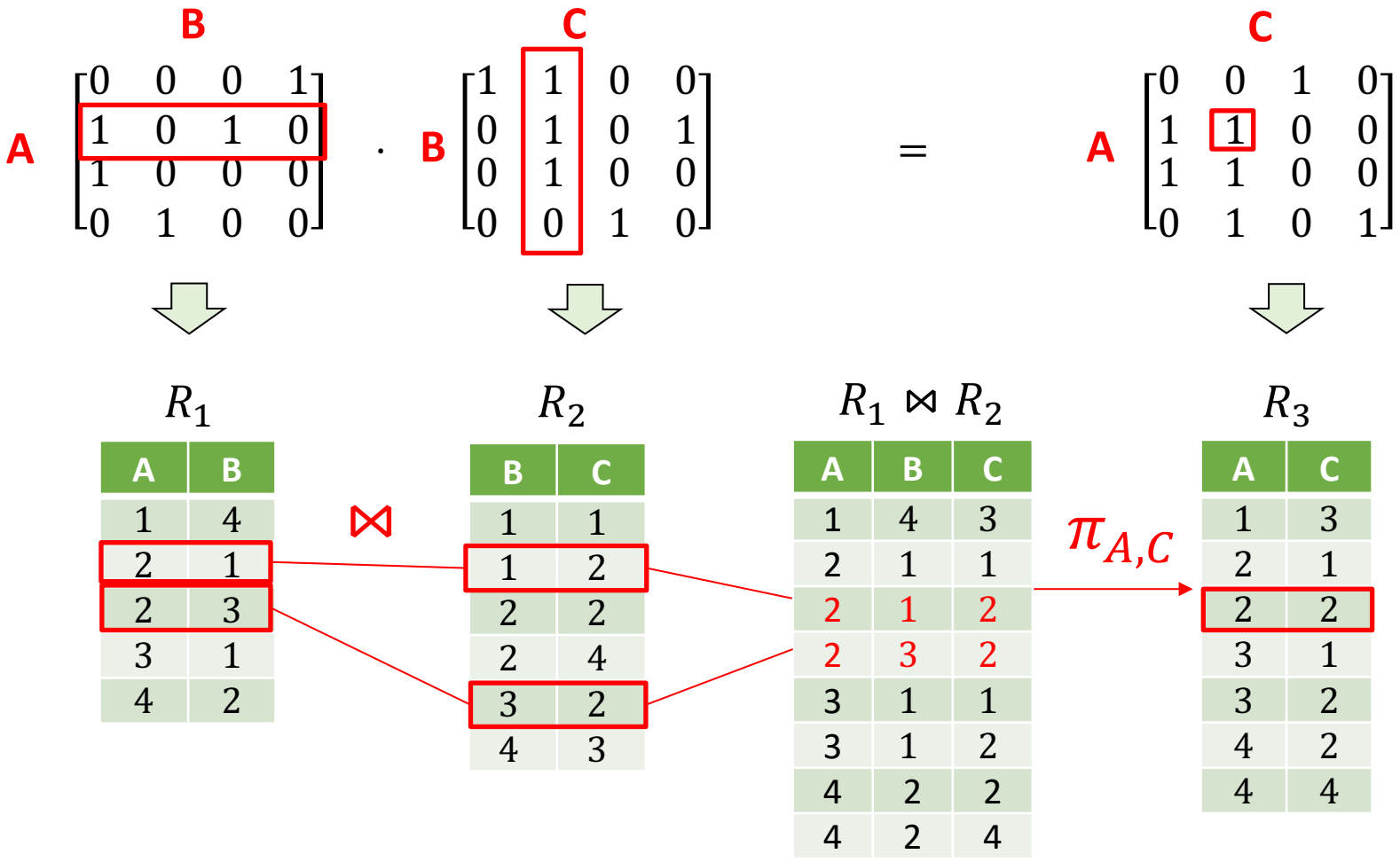
Edge3

C	A
Alice	Bob
Alice	Cathy
Bob	David
David	Alice
...	...

$$Edge1(A,B) \bowtie Edge2(B,C) \bowtie Edge3\ (C,A)$$

A	B	C
Alice	Bob	David
...	...	...

Matrix multiplication (Optional)



A	B
1	4
2	1
2	3
3	1
4	2

B	C
1	1
1	2
2	2
2	4
3	2
4	3

A	B	C
1	4	3
2	1	1
2	1	2
2	3	2
3	1	1
3	1	2
4	2	2
4	2	4

A	C
1	3
2	1
2	2
3	1
3	2
4	2
4	4

Summary

- Part 1: Relational data model
  - Data model
  - Database schema
  - Integrity constraints (keys)
  - Languages
    - relational algebra
    - relational calculus (optional)

- Part 2: Relational algebra – basic language
  - Core & derived operators
  - Write a relational algebra query
  - Applications to other domains (optional)

What's next?

- Lecture 4: SQL