[639] Relational Algebra

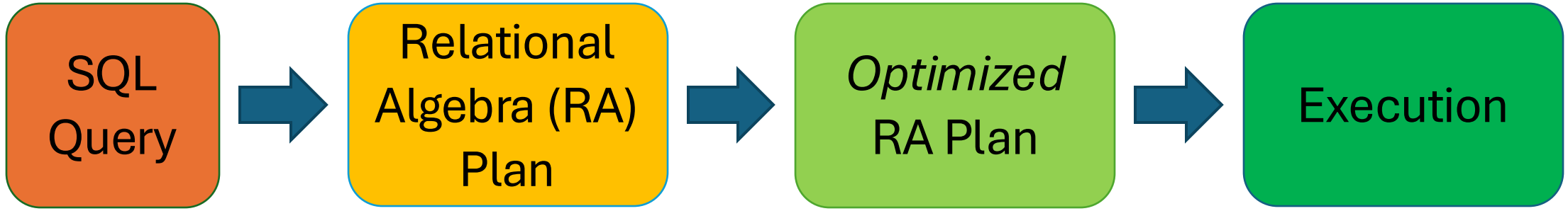
Meenakshi Syamkumar

Learning Objectives

Recognize	basic Relational Algebra operators
Recognize	derived Relational Algebra operators
Convert	SQL queries to RA expression and vice versa

RDBMS Architecture

How does a SQL engine work ?



Declarative query (from user)

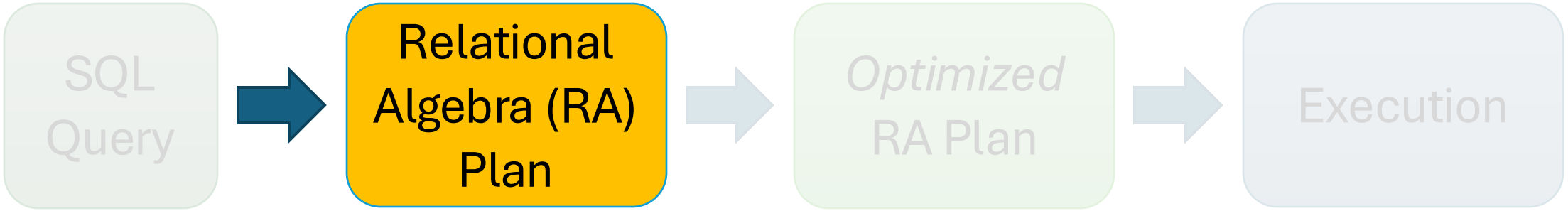
Translate to relational algebra expression

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS Architecture

How does a SQL engine work ?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

Relational Algebra: basic operators

Relational Algebra (RA)

- Basic or primitive operators:
  - Selection: σ
  - Projection: Π
  - Cartesian Product: $\times$
  - Union ($\cup$)
  - Set Difference: -
- Extended or derived or auxiliary operators:
  - Intersection ($\cap$)
  - Renaming: ρ
  - and other operators like joins

Selection (σ) aka *WHERE* clause

- Returns all tuples (data rows) which satisfy a condition
- Notation: $\sigma_c(R)$
- Other examples:
  - $\sigma_{Salary > 40000}$ (Employee)
  - $\sigma_{name = \text{“Smith”}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Students(sid, name, gpa)

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$$\sigma_{gpa > 3.5}(Students)$$

Projection (Π) aka *SELECT* clause

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Another example - project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

Professors(fid, name, cid, did)

SQL:

```
SELECT DISTINCT
  name,
  did
FROM Professors;
```

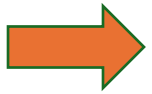
RA:

$$\Pi_{name,did}(Professors)$$

RA Operators are Compositional

Students(sid, name, gpa)

```
SELECT DISTINCT
  name,
  gpa
FROM Students
WHERE gpa > 3.5;
```



$$\Pi_{name,gpa}(\sigma_{gpa>3.5}(Students))$$

$$\sigma_{gpa>3.5}(\Pi_{name,gpa}(Students))$$

How do we represent this query in RA?

Are these logically equivalent?

No, selection (aka WHERE) occurs first!

Cross-Product (×) aka Cartesian Product

- Each tuple in R1 with each tuple in R2 (all possible combinations)
- Notation: R1 × R2
- Another example:
  - Employee × Dependents
- Rarely used in practice
  - joins are more prevalent

Students(sid, name, gpa)
Enrolled(sid, cid, grade)

SQL:

```
SELECT *
FROM Students, Enrolled;
```

RA:

$$Students \times Enrolled$$

$Students \times Enrolled$

sid	name	gpa
101	Alice	3.2
123	Bob	3.8

Students

$\times$

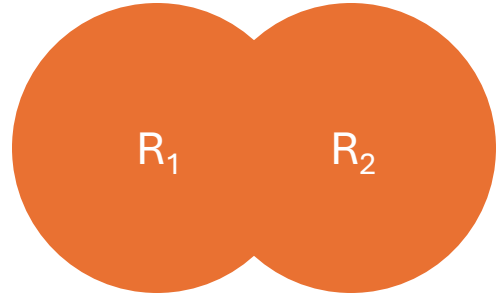
sid	cid	grade
123	544	A
101	639	A

Enrolled

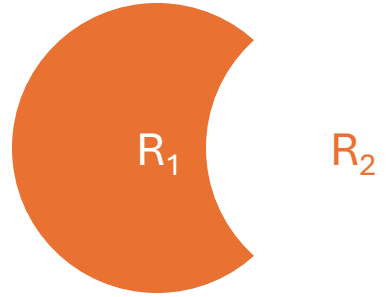
sid	name	gpa	sid	cid	grade
101	Alice	3.2	123	544	A
123	Bob	3.8	123	544	A
101	Alice	3.2	101	639	A
123	Bob	3.8	101	639	A

Union (∪), Set Difference (–), & Intersection (∩)

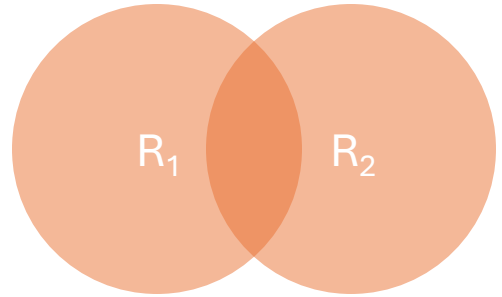
- Union:
  - R1 ∪ R2
  - Example:
    - CS639_students ∪ CS544_students



- Set Difference:
  - R1 – R2
  - Example:
    - CS639_students – CS544_students



- Intersection:
  - R1 ∩ R2 = R1 – (R1 – R2)
  - Example:
    - CS639_students ∩ CS544_students



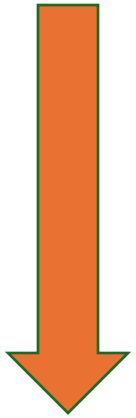
Renaming (ρ)

- Changes the schema, not the instance
- Can be used as a replacement for projection operator (Π) if all columns are renamed
- A 'special' operator - neither basic nor derived
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Note: the above is shorthand for the proper form (since names, not order matters!):
  - $\rho_{A1\rightarrow B1,\ldots,An\rightarrow Bn}(R)$

Students(sid, name, gpa)

SQL:

```
SELECT
  sid AS student_id,
  name AS student_name,
  gpa AS grade_point_avg
FROM Students;
```



RA:

$$\rho_{student\_id,student\_name,grade\_point\_avg}(Students)$$

Relational Algebra: derived operators

Relational Algebra (RA)

- Extended or derived or auxiliary operators:
  - Intersection (∩)
  - Renaming: ρ
  - and other operators like joins:
    - Natural join (⋈)
      - Left Outer Join (⟕)
      - Right Outer Join (⟖)
      - Full Outer Join (⟗)
    - Theta join (⋈_θ)
    - Aggregations (SUM, COUNT, AVG, MIN, and MAX)
    - Grouping (γ)
    - Division (÷)

Natural Join (⋈)

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
  - If R_1 has attribute set A, and R_2 has attribute set B, and they share attributes $A \cap B = C$, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
  - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ renames the shared attributes in one of the relations
    - The selection $\sigma_{C=D}$ checks equality of the shared attributes
    - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

Students(sid, name, gpa)
Enrolled(sid, cid, grade)

SQL:

```
SELECT DISTINCT
  S.sid, name, cid, grade
FROM
  Students S,
NATURAL JOIN Enrolled;
```

RA:

$$Students \bowtie Enrolled$$

Theta Join ($\bowtie_\theta$)

- Notation: $R_1 \bowtie_\theta R_2$
- Example: R ⋈_{R.A = S.B} S
- Unlike natural join, supports different conditions like <, >, <=, >=, =
- Unlike natural join, includes all attributes from both relations

Students(sid, name, gpa)
Enrolled(sid, cid, grade)

SQL:

```
SELECT DISTINCT
  S.sid, name, cid, grade
FROM
  Students S,
JOIN Enrolled E ON S.sid = E.sid;
```

RA:

$$Students \bowtie_{S.sid=E.sid} Enrolled$$

Relational Algebra (RA)

- Extended or derived or auxiliary operators:
  - Intersection (∩)
  - Renaming: ρ
  - and other operators like joins:
    - Natural join (⋈)
      - Left Outer Join (⟕)
      - Right Outer Join (⟖)
      - Full Outer Join (⟗)
    - Theta join (⋈_θ)
    - Aggregations (SUM, COUNT, AVG, MIN, and MAX)
    - Grouping (γ)
    - Division (÷)

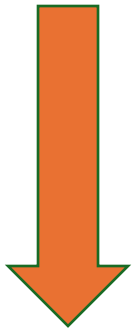
Grouping (γ)

- Notation: γA1,A2,...,An,f_1(A),f_2(B),...(R)
- A1, A2, ..., A_n: attributes by which relation R is grouped
- f_1(A), f_2(B), ...: Aggregation functions applied to columns
- While using count aggregate, RA expression does not contain projection operator (Π)
- For other aggregates, RA expression must explicitly contain the projection operator (Π)

Enrolled(sid, cid, grade)

SQL:

```
SELECT
  COUNT(*)
FROM
  Enrolled
GROUP BY cid;
```



RA:

$\gamma_{cid,COUNT(*)}(Enrolled)$

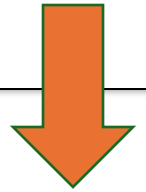
Division (÷)

- Notation: $R_1 \div R_2$
- *Note: we have a new schema for the Students table just for this example
- Example: find all students who have completed their graduation requirements
- Typically associated with “for all” conditions

Students(sid, name, cid, grade)*
GraduationRequirements(cid)

SQL:

```
SELECT
  sid, name
FROM
  Students
GROUP BY sid
HAVING COUNT(DISTINCT cid) =
(SELECT COUNT(*) FROM
GraduationRequirements);
```



RA:

Students ÷ GraduationRequirements