Advanced Database Systems

Spring 2024

Lecture #03:

Relational Algebra

R&G: Chapters 4.1 & 4.2

Query Execution Overview

SQL Query

```
SELECT S.name
  FROM Student S, Enrolled E
 WHERE S.sid = E.sid
   AND E.cid = 'INF-11199'
```

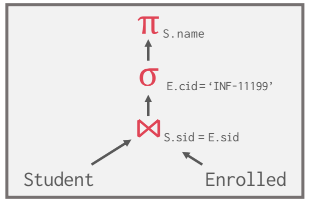
Query Parser & Optimiser

Relational Algebra

$\pi_{S.name}(\sigma_{E.cid='INF-11199'}(\text{Student} \bowtie_{S.sid=E.sid} \text{Enrolled}))$

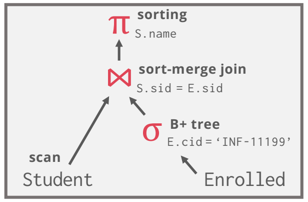
Equivalent to...

Logical Query Plan



But actually will produce plan with operator code

Optimised Physical Query Plan



Query Execution Overview

SQL Query

```
SELECT S.name
  FROM Student S, Enrolled E
 WHERE S.sid = E.sid
   AND E.cid = 'INF-11199'
```

Declarative description of computation

- Say **what** you want, not **how** to get it
- Enables system to optimize the **how**

Foundation in formal query languages

- Relational Calculus

Relational Algebra

$\pi_{S.name}(\sigma_{E.cid='INF-11199'}(\text{Student} \bowtie_{S.sid=E.sid} \text{Enrolled}))$

Operational description of computation

- Systems execute RA query plans

Relational Query Languages

Relational Calculus (basis for SQL)

- Describe the result of computation
- Based on first order logic
- Tuple Relational Calculus (**TRC**)

$\{ S \mid S \in \text{Student} \;\; \exists E \in \text{Enrolled} \;\; (S.sid = E.sid \wedge E.cid = \text{'INF-11199'}) \}$

Relational Algebra

- Algebra on sets
- Operational description of transformations

Are these two equivalent?

Can we go from one to the other?

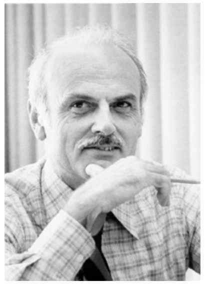
Codd's Theorem

Established equivalence in expressivity between:

- Relational Calculus
- Relational Algebra

Why an important result?

- Connects declarative representation of queries with operational description
- Constructive: we can compile SQL into relational algebra



Edgar F. "Ted" Codd
(1923 - 2003)
Turing Award 1981

Relational Algebra

Algebra of operators on relation instances

$$\pi_{S.name}(\sigma_{E.cid='INF-11199'}(S \bowtie_{S.sid=E.sid} E))$$

Closed: result is also a relation instance

- Enables rich composition!

Typed: input schema determines output schema

- Can statically check whether queries are legal

σ	Selection
π	Projection
ρ	Renaming
∪	Union
–	Set Difference
×	Cross Product
∩	Intersection
⋈	Join

Relational Algebra and Sets

Pure relational algebra has set semantics

- No duplicate tuples in a relation instance
- But can also be defined over bags (multisets)

SQL has multiset (bag) semantics

- We will switch to multiset in the system discussion

Selection

Syntax: $\sigma_{predicate}(R)$

Select a subset of rows (horizontal) that satisfy a selection predicate

- Can combine predicates using conjunctions / disjunctions
- Output schema same as input
- Duplicate elimination? Not needed

R(aid, bid)

aid	bid
a1	101
a2	102
a2	103
a3	104

$\sigma_{aid='a2'}(R)$

aid	bid
a2	102
a2	103

$\sigma_{aid='a2' \wedge bid > 102}(R)$

aid	bid
a2	103

Projection

Syntax: $\pi_{A1, A2, ..., An}$ (R)

Selects a subset of columns (vertical)

- Schema determined by schema of attribute list

Set semantics → results in fewer rows

- Real systems don't automatically remove duplicates
- Why?
  - 1) Semantics: keep duplicates for aggregates
  - 2) Performance: deduplication not free

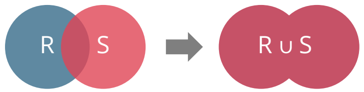
R(aid, bid)

aid	bid
a1	101
a2	102
a2	103
a3	104

π_{aid} (R)

aid
a1
a2
a3

Union



Syntax: R ∪ S

Two input relations must be **compatible**

- Same number of fields
- Fields in corresponding positions have same **type**

Duplicate elimination in practice (SQL)?

- UNION – eliminates duplicates
- UNION ALL – keeps duplicates

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

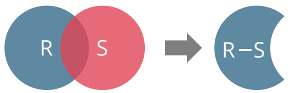
S(aid, bid)

aid	bid
a3	103
a4	104
a5	105

R U S

aid	bid
a1	101
a2	102
a3	103
a4	104
a5	105

Set Difference



Syntax: R − S

Same as with union, both input relations must be **compatible**

- Duplicate elimination? Not needed

SQL expression: EXCEPT

- EXCEPT vs EXCEPT ALL

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(aid, bid)

aid	bid
a3	103
a4	104
a5	105

R – S

aid	bid
a1	101
a2	102

S – R

aid	bid
a4	103
a5	105

Cross Product

Syntax: R × S

Each row of R paired with each row of S

- How many rows in result? |R|*|S|
- Schema compatibility? Not needed
- Duplicates? None generated

R × S has two bid attributes

- Not allowed, leave them unnamed
- Identify attributes by position

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(bid, cid)

bid	cid
b3	23
b4	24

R × S

aid	(bid)	(bid)	cid
a1	101	b3	23
a1	101	b4	24
a2	102	b3	23
a2	102	b4	24
a3	103	b3	23
a3	103	b4	24

Renaming (ρ = "rho")

Renames relations and their attributes

- Note that relational algebra doesn't require names
- We could just use positional arguments

ρ(Temp(2 → bid1, 3 → bid2), R × S)

Output Relation Name — *Renaming List position → new name* — *Input Expression*

R × S

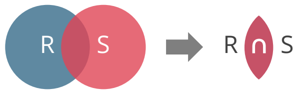
aid	(bid)	(bid)	cid
a1	101	b3	23
a1	101	b4	24
a2	102	b3	23
a2	102	b4	24

Temp

aid	bid1	bid2	cid
a1	101	b3	23
a1	101	b4	24
a2	102	b3	23
a2	102	b4	24

Compound Operator: Intersection

Syntax: R ∩ S



Same as with union, both input relations must be **compatible**

SQL expression: INTERSECT

Equivalent to: R – (R – S)

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(aid, bid)

aid	bid
a3	103
a4	104
a5	105

R ∩ S

aid	bid
a3	103

Compound Operator: Join

Joins are compound operators (like intersection):

- Generally, $\sigma_\theta(R \times S)$

Hierarchy of common kinds:

- **Theta Join** ($\bowtie_\theta$): join on logical expression θ
  - **Equi-Join**: theta join with theta being a conjunction of equalities
    - **Natural Join** ($\bowtie$): equi-join on all matching column names

Note: we will need to learn a good join algorithm

Avoid cross-product if we can!

Theta Join Example

Student

sid	name	age
12344	Jones	18
12355	Smith	23
12366	Gold	21

Enrolled

sid	cid	grade
12344	INF-10080	65
12355	INF-11199	72

Student $\bowtie_{sid=sid}$ Enrolled

(sid)	name	age	(sid)	cid	grade
12344	Jones	18	12344	INF-10080	65
12355	Smith	23	12355	INF-11199	72

Note that output needs a rename operator!

Theta Join Example 2

Example: Get senior students for each student

Student $\bowtie_{field3 < field6}$ Student (i.e., age < age2)

R $\bowtie_\theta$ S = σ_θ(R × S)

Student $\bowtie_{field3 < field6}$ Student

(sid)	(name)	(age)	(sid)	(name)	(age)
12344	Jones	18	12355	Smith	23
12344	Jones	18	12366	Gold	21
12366	Gold	21	12355	Smith	23

Output schema same as that of cross product

Student

sid	name	age
12344	Jones	18
12355	Smith	23
12366	Gold	21

Natural Join

Syntax: R ⋈ S

Special case of equi-join in which equalities are specified for all matching fields and duplicate fields are projected away

R ⋈ S = $\pi_{\text{unique fld.}}\ \sigma_{\text{eq.matching fld.}}$ (R × S)

Compute R × S

Select rows where fields appearing in both relations have equal values

Project onto the set of all unique fields

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(bid, cid)

bid	cid
101	c3
101	c4
105	c5

R ⋈ S

aid	bid	cid
a1	101	c3
a1	101	c4

Extra Operators

Group By / Aggregation (γ)

$\gamma_{\text{dept, AVG(age)}}$ (Student)

$\gamma_{\text{dept, AVG(age), COUNT(*) > 2}}$ (Student) with selection (HAVING clause)

Duplicate Elimination (δ)

only under multiset (bag) interpretation of relational algebra

Assignment (R ← S)

Sorting (τ)

Division (R ÷ S)

Summary

Relational Algebra

A small set of operators mapping relations to relations

Operational, in the sense that you specify the explicit order of operations

A closed set of operators! Mix and match

Basic operators: σ, π, ρ, ∪, −, ×

Important compound operators: ∩, ⋈