

Relational Algebra

Davood Rafiei

Original material Copyright 2001-2022

(some material from textbooks and other instructors)

Relational Query Languages

- Languages for describing queries on a relational database
- Three variants
 - Relational Algebra
 - Relational Calculus
 - SQL
- Query languages v.s. programming languages
 - QLs not expected to be “Turing complete”.
 - QLs support easy, efficient access to large data sets.

SQL & Relational Algebra

- *Structured Query Language (SQL)*
 - Predominant application-level query language
 - Declarative
- *Relational Algebra*
 - Intermediate language used within DBMS
 - Procedural

Algebra

- Study of **operations** in an abstract level on some **domains**
 - e.g. operations $+$, $-$ and $*$ on **natural numbers**
- As a language, study of syntax and semantics of expressions
 - e.g $2+3$, $(46-3)+3$, $(7*x)+(3*x)$
- Relational algebra
 - Domain: set of all relations
 - Expressions: referred to as *queries*

Operations on Tables

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
1133	Joe	125-34 Ave	biking
2232	Bob	7 Whyte Ave	hockey
5678	John	123-34 Ave	stamps

Person

<i>id</i>	<i>name</i>	<i>office</i>
1133	Joe	333 Ath
5678	John	222 CSC

Employee

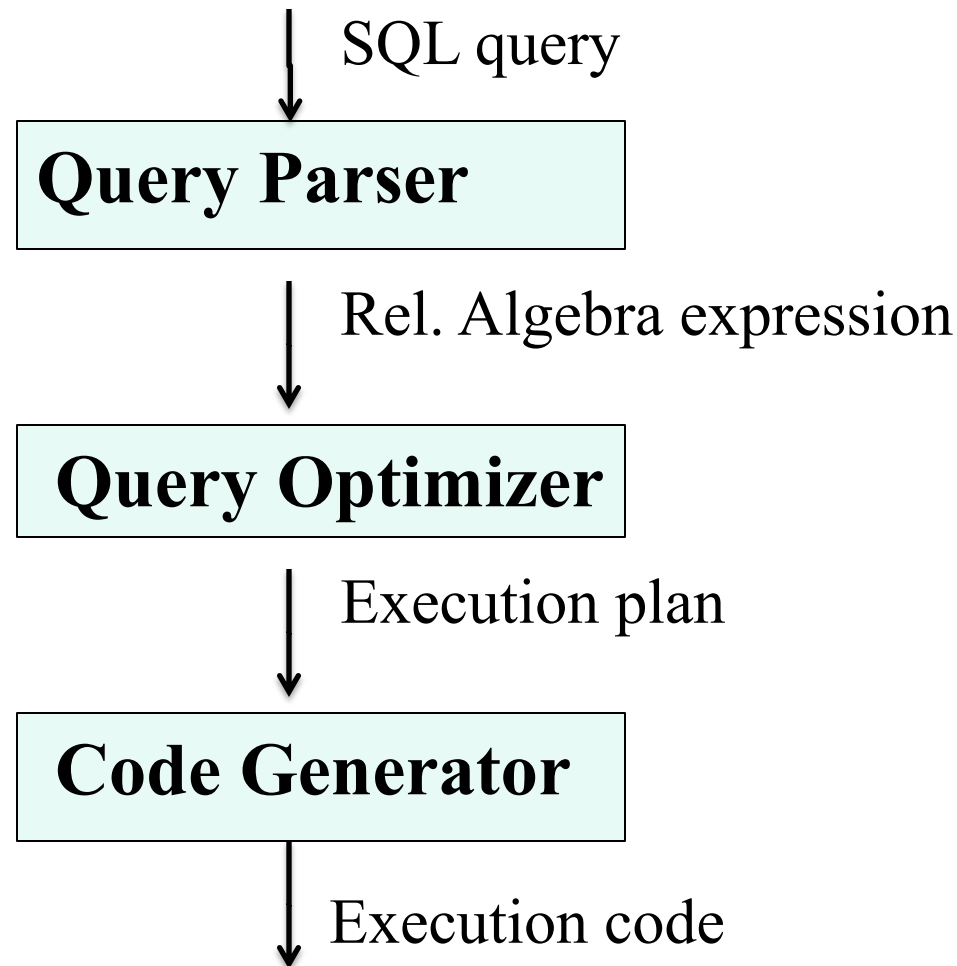
Operations on one table: pick some rows, pick some columns

Operations on two tables: join, ...

Relational Algebra

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, cartesian product, renaming
- *Derived operators*: set intersection, division, join
- *The language is “procedural”*
 - expressions or queries specify a sequence of operations to be performed to obtain the result

Role inside a DBMS



Select Operator

- Select rows that satisfy the condition

$$\sigma_{condition}(relation)$$

- Example:

$$\sigma_{hobby='hockey'}(Person)$$

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
1133	Joe	125-34 Ave	biking
2232	Bob	7 Whyte Ave	hockey
5678	John	123-34 Ave	stamps

Person

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
2232	Bob	7 Whyte Ave	hockey

Selection Condition

- Operators: $<$, \leq , \geq , $>$, $=$, \neq
- Simple selection condition:
 - $\langle attribute \rangle operator \langle constant \rangle$
 - $\langle attribute \rangle operator \langle attribute \rangle$
- $\langle condition \rangle AND \langle condition \rangle$
- $\langle condition \rangle OR \langle condition \rangle$
- NOT $\langle condition \rangle$

More Examples

- $\sigma_{id > 1000 \text{ OR } \textit{hobby} = \text{'hockey'}} (\text{Person})$
- $\sigma_{id > 3000 \text{ AND } id < 3500} (\text{Person})$
- $\sigma_{\text{NOT}(\textit{hobby} = \text{'hockey'})} (\text{Person})$
- $\sigma_{\textit{hobby} \neq \text{'hockey'}} (\text{Person})$

Project Operator

- Project on (or pick) a subset of columns

$$\pi_{attribute\ list}(relation)$$

- Example:

$$\pi_{name,hobby}(\text{Person})$$

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
1133	Joe	125-34 Ave	biking
2232	Bob	7 Whyte Ave	hockey
5678	John	123-34 Ave	stamps

Person

<i>name</i>	<i>hobby</i>
John	hockey
Joe	biking
Bob	hockey
John	stamps

Example

$\pi_{name, address}(\text{Person})$

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
1133	Joe	125-34 Ave	biking
2232	Bob	7 Whyte Ave	hockey
5678	John	123-34 Ave	stamps

Person

<i>name</i>	<i>address</i>
John	123-34 Ave
Joe	125-34 Ave
Bob	7 Whyte Ave

Result is a relation (meaning no duplicates)!

Expressions

$\pi_{id, name} (\sigma_{hobby='stamps' \text{ OR } hobby='biking'} (\text{Person}))$

<i>id</i>	<i>name</i>	<i>address</i>	<i>hobby</i>
1122	John	123-34 Ave	hockey
1133	Joe	125-34 Ave	biking
2232	Bob	7 Whyte Ave	hockey
5678	John	123-34 Ave	stamps

Person

<i>id</i>	<i>name</i>
1133	Joe
5678	John

Set Operators

- Relation \sim a set of tuples
 - Set operations apply
- Operations: \cap , \cup , $-$ (set difference)
- Defined (or meaningful) between relations that have the same structure (called *union compatible relations*)

Union Compatible Relations

- Two relations are *union compatible* if
 - Both have the same number of columns
 - Names of attributes are the same in both
 - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using *union*, *intersection*, and *set difference*

Example

Tables:

Person (*SSN, Name, Address, Hobby*)

Professor (*Id, Name, Office, Phone*)

are not union compatible, but

$\pi_{name}(\text{Person})$ and $\pi_{name}(\text{Professor})$
are union compatible and

$\pi_{name}(\text{Person}) - \pi_{name}(\text{Professor})$
makes sense.

Cartesian Product

- $R \times S = \{ \langle x, y \rangle \mid x \text{ in } R, y \text{ in } S \}$
 - $R \times S$ is the set of all concatenated tuples $\langle x, y \rangle$, where x is a tuple in R and y is a tuple in S
 - Relations don't have to be union-compatible
- $R \times S$ can be huge (and expensive to compute)
 - Factor of two in the size of each row
 - Quadratic in the number of rows

<i>a</i>	<i>b</i>
<i>a</i> ₁	<i>b</i> ₁
<i>a</i> ₂	<i>b</i> ₂

R

<i>c</i>	<i>d</i>
<i>c</i> ₁	<i>d</i> ₁
<i>c</i> ₂	<i>d</i> ₂

S

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁	<i>d</i> ₁
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₂	<i>d</i> ₂
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₁	<i>d</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂	<i>d</i> ₂

$R \times S$

Renaming

- $\rho_{\mathbf{x}}(A_1, A_2, \dots A_n) \text{ expr}$
 - *expr* returns a relation
 - Rename the resulting relation to \mathbf{x} with the first column in the result relation to A_1 , the second to A_2 , etc.
- Example
 - Let $R(a,b)$ be a relation with two columns
 - $\rho_{(p, q, r, s)} R \times R$ is a relation with 4 columns p, q, r and s .
- Common usage
 - To clean up the result
 - To prepare the result for the next operation

Example

Transcript (*sid, cid, sem, grade*)

Teaching (*pid, cid, sem*)

Transcript \times *Teaching* is not defined, but the following is:

$$\rho_{(sid, cid1)}(\pi_{sid, cid}(\text{Transcript})) \times \rho_{(pid, cid2)}\pi_{pid, cid}(\text{Teaching})$$

The result is a relation with 4 attributes:

sid, cid1, pid, cid2

Join (derived operator)

A (*general* or *theta*) *join* of R and S is the expression

$$R \bowtie_{\text{join-condition}} S$$

where *join-condition* is a *conjunction* of terms:

$$A_i \text{ oper } B_i$$

in which A_i is an attribute of R ; B_i is an attribute of S ;
and *oper* is one of $=, <, >, \geq, \neq, \leq$.

Equivalent to

$$\sigma_{\text{join-condition}' } (R \times S)$$

where *join-condition* and *join-condition'* are the same,
except for possible renaming of attributes (next)

Join and Renaming

- **Problem:** If R and S have attributes with the same name, then the Cartesian product is not well-defined
- **Two Solutions:**
 - Rename attributes prior to forming the product and use new names in *join-condition*'.
 - Common attribute names are qualified with relation names in the result of the join

Join – Example

Find employees who earn more than their managers.

Tables: Employee(name, id, salary, mngrId)
Manager(name, id, salary)

$\pi_{\text{Employee.name}} (\text{Employee} \bowtie_{\text{mngrId=manager.id AND Employee.salary > Manager.salary}} \text{Manager})$

The join yields a table with attributes:

Employee.name, Employee.id, Employee.salary, mngrId
Manager.name, Manager.id, Manager.salary

Equijoin - Example

Equijoin: Join condition is a conjunction of *equalities*.

$\pi_{name,cid}(\text{Student} \bowtie_{id=sid} \sigma_{grade='A'}(\text{Transcript}))$

Student

<i>id</i>	<i>name</i>	<i>addr</i>	<i>status</i>
111	John
222	Mary
333	Bill
444	Joe

Transcript

<i>sid</i>	<i>cid</i>	<i>sem</i>	<i>grade</i>
111	114	F10	B
222	115	F10	A
333	201	W10	A

Mary	115
Bill	201

The equijoin is very useful since it combines related data in different relations.

Natural Join

- Special case of equijoin:
 - join condition equates *all* and *only* those attributes with the same name (condition doesn't have to be explicitly stated)
 - duplicate columns eliminated from the result

Transcript (<i>sid</i> , <i>cid</i> , <i>sem</i> , <i>grade</i>) Teaching (<i>pid</i> , <i>cid</i> , <i>sem</i>)

Transcript \bowtie Teaching =

$\pi_{sid, \text{Transcript.}cid, \text{Transcript.}sem, grade, pid}$

(Transcript $\bowtie_{\text{Transcript.}cid=\text{Teaching.}cid \text{ AND } \text{Transcript.}sem=\text{Teaching.}sem}$ Teaching)

Natural Join Means

- More generally:

$$R \bowtie S = \pi_{attr-list} (\sigma_{join-cond} (R \times S))$$

- $attr-list = attributes(R) \cup attributes(S)$
(duplicates are eliminated)
- $join-cond$ has the form
 $A_1 = A_1 \text{ AND } \dots \text{ AND } A_n = A_n$
- $\{A_1 \dots A_n\} = attributes(R) \cap attributes(S)$

Example

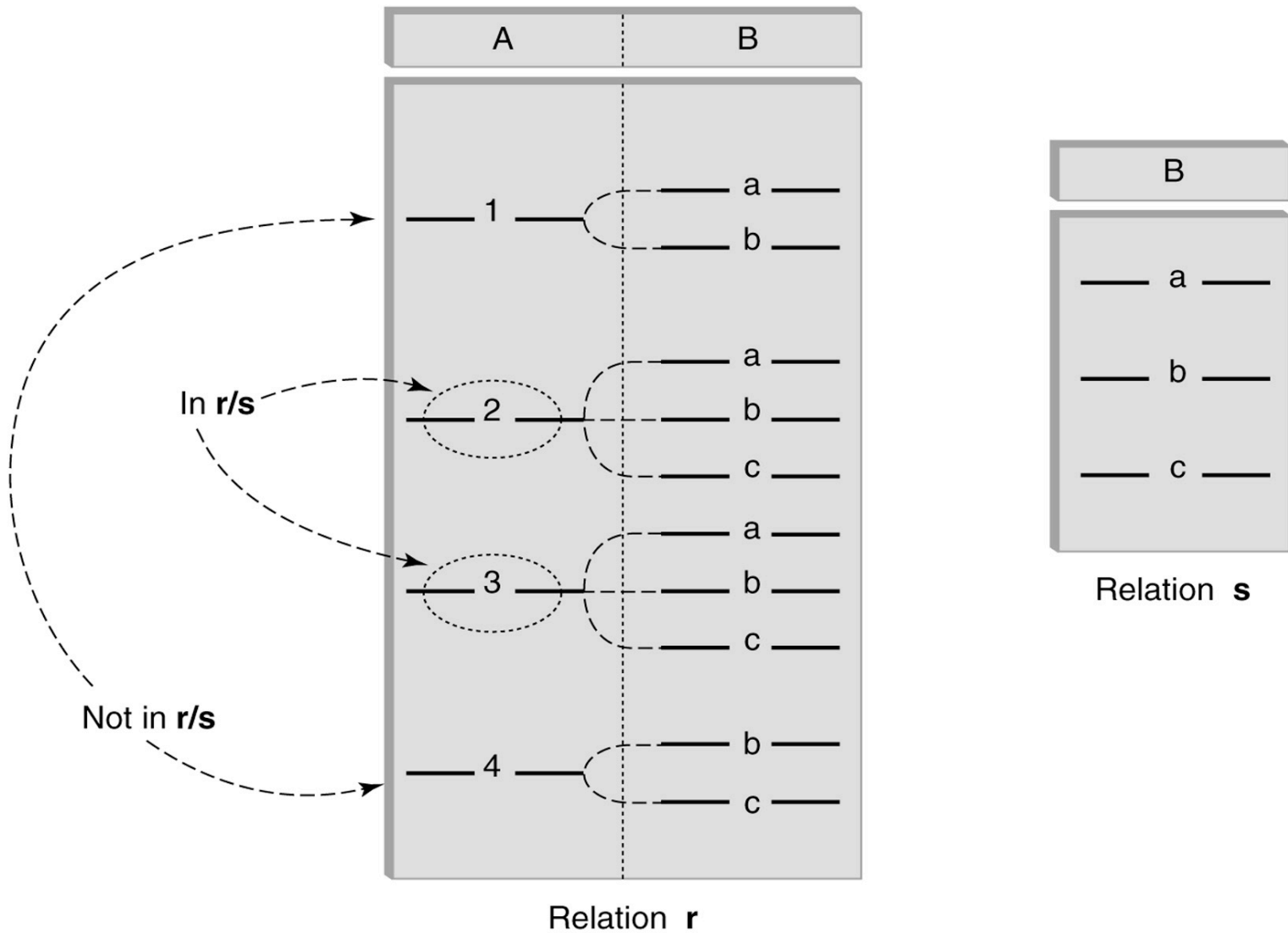
- List the id's of students who took at least two different courses.

Transcript(*sid*, *cid*, *sem*, *grade*)

- *Transcript* \bowtie *Transcript* won't work!
- Solution?

Division (derived operator)

- Finds tuples in one relation, r , that match *all* tuples in another relation, s
 - $r (A_1, \dots A_n, B_1, \dots B_m)$
 - $s (B_1 \dots B_m)$
 - r/s , with attributes $A_1, \dots A_n$, is the set of all tuples $\langle a \rangle$ such that for every tuple $\langle b \rangle$ in s , $\langle a, b \rangle$ is in r
- Can be expressed in terms of projection, set difference, and cross-product



Division - Example

- List the Ids of students who have passed all courses taught in winter 2010
- *Numerator*:
 - *sid* and *cid* for every course passed by every student:

$$\pi_{sid, cid}(\sigma_{Grade \neq 'F'}(\text{Transcript}))$$

- *Denominator*:
 - *cid* of all courses taught in winter 2010

$$\pi_{cid}(\sigma_{Sem='W10'}(\text{Teaching}))$$

- Result is *numerator/denominator*