Databases I.

1

RELATIONAL QUERY LANGUAGES - RELATIONAL ALGEBRA

Operations for the Relational Model

- Operations on relations can be expressed according to two basic formalisms:
  - **Relational Algebra**: queries are specified by applying specialized operators to relations
  - **Relational Calculus**: queries are expressed through logical formulas that must be verified by the tuples that are result of queries
- The two formalisms (under certain assumptions) are equivalent

3

Relational Algebra

- It consists of five *basic* operations:
  - Union
  - Difference
  - Cartesian Product
  - Projection
  - Selection
- These operations fully define the relational algebra

4

Relational Algebra

- Each operation returns a relation as result: it is thus possible to apply an operation to the result of another operation (*closure property*)
- There are some additional operations that can be expressed in terms of the five basic operations
  - Join
  - Intersection
  - division

5

Relational Algebra

- Such additional operations do not increase the *expressive power* of the set of the basic operations
- However, these additional operations are used as shorthand; the *join* is the most important of these operations
- With respect to the notation by name, it can be useful to introduce another operation, called *renaming*, that allows one to modify the attribute names

6

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Relational Algebra - Union

- The union of relations R and S, denoted by R ∪ S
- is the set of tuples that are in R, in S or in both
- The union of two relations can only be executed if the relations have the *same degree*; in addition the first attribute of R must be *compatible* with the first attribute of S, the second attribute of R must be *compatible* with the second attribute of S, and so forth
  - The duplicate tuples are eliminated
  - The degree of the resulting relation is the same as the degree of the input relations

8

Relational Algebra – Union

- Example

A	B	C
a	b	c
d	a	f
c	b	d

R

D	E	F
b	g	a
d	a	f

S

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

R∪S

Relational Algebra - Difference

- The difference of relations R and S, denoted as R – S
  is the set of tuples that are in R and are not in S
- The difference, like the union, can only be executed if the input relations have the same degrees and compatible attributes
- The degree of the resulting relation is the same as the degree of the input relations

Relational Algebra – Difference Example

A	B	C
a	b	c
d	a	f
c	b	d

R

D	E	F
b	g	a
d	a	f

S

A	B	C
a	b	c
c	b	d

R-S

11

Intersection of two relations

Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$

A	B
α	2

Example Instances

- "Sailors" and "Reserves" relations for our examples. "bid"= boats. "sid": sailors
- We'll use positional or named field notation, assume that names of fields in query results are `inherited' from names of fields in query input relations.

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Cartesian Product (Cross-Product)

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- *Renaming operator*: $\rho\,(C(1 \to sid1, 5 \to sid2),\ S1 \times R1)$

Cartesian Product (symbol is x)

Relations *r, s*:

A	B	C
a	b	c
c	d	e
g	a	d

B	D
b	r
q	s

r x *s*:

A	R.B	C	S.B	D
a	b	c	b	r
a	b	c	q	s
c	d	e	b	r
c	d	e	q	s
g	a	d	b	r
g	a	d	q	s

Sample: R3 := R1 X R2

R1(

A,	B)
1	2
3	4

R2(

B,	C)
5	6
7	8
9	10

R3(

A,	R1.B,	R2.B,	C)
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??, what are the consequences?)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Selection

- Selects rows that satisfy *selection condition*.
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Selection of rows (tuples)

The relation **r :**

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Select tuples with A=B and D > 5

$\sigma_{A=B \text{ and } D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Projection=Selection of Columns (Attributes)

Relation ***r***:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Projection
(Select A and C)

$\Pi_{A, C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Joins

- *Condition Join*: $R \bowtie_C S = \sigma_C(R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product. Filters tuples not satisfying the join condition.
- Sometimes called a *theta-join*.

Joins

- *Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities*.**

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$\pi_{sid,..,age,bid,..}(S1 \bowtie_{sid} R1)$$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.

Joining two relations – Natural Join

- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, the "natural join" of relations *R* and *S* is a relation on schema $R \cup S$ obtained as follows:
- Matches tuples with the same values for all common attributes, and retains only one copy of each common column

Natural Join Example

Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

Natural Join

r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Relational algebra operations

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(\textit{instructor})$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{\textit{ID, salary}}(\textit{instructor})$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	*instructor* × *department*
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(\textit{instructor}) \cup \Pi_{name}(\textit{student})$
	Output the union of tuples from the two input relations.

Find names of sailors who've reserved boat #103

Solution 1: $\pi_{sname}((\sigma_{bid=103}\text{Reserves}) \bowtie Sailors)$

Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Find names of sailors who've reserved a red boat

Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'}, Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'}, Boats) \bowtie \text{Res}) \bowtie Sailors)$$

A query optimizer can find this, given the first solution!

Find sailors who've reserved a red or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}, Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

Find sailors who've reserved a red and a green boat

Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'}, Boats) \bowtie Reserves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'}, Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Outer Join

- Often in joining two relations, a tuple in one relation does not have a matching tuple in the other relation; in other words, there is no matching value in the join attributes.
- We may want tuples from one of the relations to appear in the result even when there are no matching values in the other relation.
- The advantage of an Outer join is that information is preserved; that is, the Outer join preserves tuples that would have been lost by other types of join.
- **R ⟕ S** The **LEFT OUTER JOIN** is a join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation. Missing values in the second relation are set to null.
- **R ⟖ S** The **RIGHT OUTER JOIN** is a join in which tuples from S that do not have matching values in the common attributes of R are also included in the result relation. Missing values in the first relation are set to null.
- **R ⟗ S** The **FULL OUTER JOIN** keeps all tuples in both relations, padding tuples with nulls when no matching tuples are found.

Examples for JOIN operations

We use *DreamHome* database relation schema:

Branch (branchNo, street, city, postcode)

Staff (staffNo, fName, IName, position, sex, DOB, salary, *branchNo*)

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, *ownerNo*, *staffNo*, *branchNo*)

Client (clientNo, fName, IName, telNo, prefType, maxRent, eMail)

PrivateOwner (ownerNo, fName, IName, address, telNo, eMail, password)

Viewing (clientNo, propertyNo, viewDate, comment)

Registration (clientNo, branchNo, *staffNo*, dateJoined)

Ex. 1. List the names and comments of all clients who have viewed a property for rent. (Equijoin operation)

$(\Pi_{clientNo, fName, lName}(Client)) \bowtie_{Client.clientNo = Viewing.clientNo} (\Pi_{clientNo, propertyNo, comment}(Viewing))$

Ex. 2. List the names and comments of all clients who have viewed a property for rent. (Natural join operation)

$(\Pi_{clientNo, fName, lName}(Client)) \bowtie (\Pi_{clientNo, propertyNo, comment}(Viewing))$

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

Ex. 3. Produce a relation consisting of the properties that have been viewed with comments and those that have not been viewed.

$(\Pi_{propertyNo, street, city}(PropertyForRent)) \bowtie Viewing$

Note that properties PL94, PG21, and PG16 have no viewings, but these tuples are still contained in the result with nulls for the attributes from the Viewing relation. (Left outer join.)

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-13	too small
PA14	16 Holhead	Aberdeen	CR62	14-May-13	no dining room
PL94	6 Argyll St	London	null	null	null
PG4	6 Lawrence St	Glasgow	CR76	20-Apr- 13	too remote
PG4	6 Lawrence St	Glasgow	CR56	26-May-13	
PG36	2 Manor Rd	Glasgow	CR56	28-Apr- 13	
PG21	18 Dale Rd	Glasgow	null	null	null
PG16	5 Novar Dr	Glasgow	null	null	null

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G1, G2, \ldots Gn}\Gamma_{F1(A1), F2(A2)\ldots..Fn(An)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation - Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\Gamma_{\mathbf{sum(c)}}(r)$

sum(*c*)
27

Aggregate, GROUPING Operation - Example

- Find the average salary in each department

$$dept\_name \; \Gamma_{\textbf{avg}(salary)} \; (instructor)$$

Instructor

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Result:

dept_name	*salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$dept\_name \; \Gamma \; \textbf{avg}(salary) \; \textbf{as} \; avg\_sal \; (instructor)$$

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.