Relational Algebra

Relational Query Languages

Recall: Query = “Retrieval Program”
Language Examples:

Theoretical:

1. *Relational Algebra*
2. *Relational Calculus*
   a. Tuple Relational Calculus (TRC)
   b. Domain Relational Calculus (DRC)

Practical:

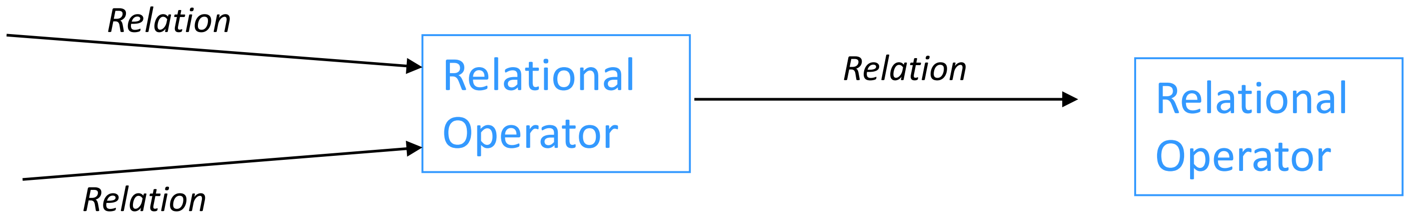
1. *SQL* (originally: SEQUEL from System R)
2. *Quel* (used in Ingres)
3. *Datalog* (Prolog-like – used in research lab systems)

- Theoretical QLs give semantics to Practical QLs

Relational Algebra

- Basic Operators
  1. select (σ)
  2. project (π)
  3. union (∪)
  4. set difference (–)
  5. cartesian product (×)
  6. rename (ρ)

- Closure Property



Select (σ)

Notation: $\sigma_{predicate}$ (*Relation*)

Relation: *Can be name of table or result of another query*

Predicate:

1. Simple
   - attribute$_1$ = attribute$_2$
   - attribute = constant value *(also: ≠, <, >, ≤, ≥)*

2. Complex
   - predicate AND predicate
   - predicate OR predicate
   - NOT predicate

Idea:

Select rows from a relation based on a predicate

Bank Database

Account

bname	<u>acct_no</u>	balance
Downtown	A-101	500
Mianus	A-215	700
Perry	A-102	400
R.H.	A-305	350
Brighton	A-201	900
Redwood	A-222	700
Brighton	A-217	750

Depositor

cname	acct_no
Johnson	A-101
Smith	A-215
Hayes	A-102
Turner	A-305
Johnson	A-201
Jones	A-217
Lindsay	A-222

Customer

<u>cname</u>	cstreet	ccity
Jones	Main	Harrison
Smith	North	Rye
Hayes	Main	Harrison
Curry	North	Rye
Lindsay	Park	Pittsfield
Turner	Putnam	Stanford
Williams	Nassau	Princeton
Adams	Spring	Pittsfield
Johnson	Alma	Palo Alto
Glenn	Sand Hill	Woodside
Brooks	Senator	Brooklyn
Green	Walnut	Stanford

Branch

<u>bname</u>	bcity	assets
Downtown	Brooklyn	9M
Redwood	Palo Alto	2.1M
Perry	Horseneck	1.7M
Mianus	Horseneck	0.4M
R.H.	Horseneck	8M
Pownel	Bennington	0.3M
N. Town	Rye	3.7M
Brighton	Brooklyn	7.1M

Borrower

cname	lno
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17
Adams	L-16

Loan

bname	lno	amt
Downtown	L-17	1000
Redwood	L-23	2000
Perry	L-15	1500
Downtown	L-14	1500
Mianus	L-93	500
R.H.	L-11	900
Perry	L-16	1300

Select (σ)

Notation: $\sigma_{predicate}(Relation)$

$\sigma_{\text{bcity} = \text{"Brooklyn"}}(\text{branch}) =$

bname	bcity	assets
Downtown	Brooklyn	9M
Brighton	Brooklyn	7.1M

$\sigma_{\text{assets} > \$8M}(\sigma_{\text{bcity} = \text{"Brooklyn"}}(\text{branch})) =$

bname	bcity	assets
Downtown	Brooklyn	9M

Project (π)

Notation: $\pi_{A1,\ ...,\ An}$ (*Relation*)

- Relation: name of a table or result of another query
- Each A_i is an attribute
- Idea: π selects columns (vs. σ which selects rows)

$\pi_{\text{cstreet, ccity}}$ (customer) =

cstreet	ccity
Main	Harrison
North	Rye
Park	Pittsfield
Putnam	Stanford
Nassau	Princeton
Spring	Pittsfield
Alma	Palo Alto
Sand Hill	Woodside
Senator	Brooklyn
Walnut	Stanford

Project (π)

$\pi_{\text{bcity}}(\sigma_{\text{assets} > 5\text{M}}(\text{branch})) =$

bcity
Brooklyn
Horseneck

Question: Does the result of Project always have the same number of tuples as its input?

Union (∪)

Notation: $Relation_1 \cup Relation_2$

R ∪ S valid only if:

1. *R, S have same number of columns (arity)*
2. *R, S corresponding columns have same name and domain (compatibility)*

Example:

$$(\pi_{cname}(\text{depositor})) \cup (\pi_{cname}(\text{borrower})) =$$

cname
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Adams

Schema:

Depositor	
cname	**acct_no**

Borrower	
cname	**lno**

Set Difference (–)

Notation: *Relation$_1$* - *Relation$_2$*

R - S valid only if:

1. *R, S have same number of columns (arity)*
2. *R, S corresponding columns have same domain (compatibility)*

Example:

$$(\pi_{\text{bname}} (\sigma_{\text{amount} \geq 1000} (\text{loan}))) - (\pi_{\text{bname}} (\sigma_{\text{balance} < 800} (\text{account}))) =$$

bname	[redacted]
Downtown	[redacted]
Redwood	[redacted]
Perry	[redacted]

[redacted]

bname	[redacted]
Mianus	[redacted]
[redacted]	[redacted]
Redwood	[redacted]
[redacted]	[redacted]

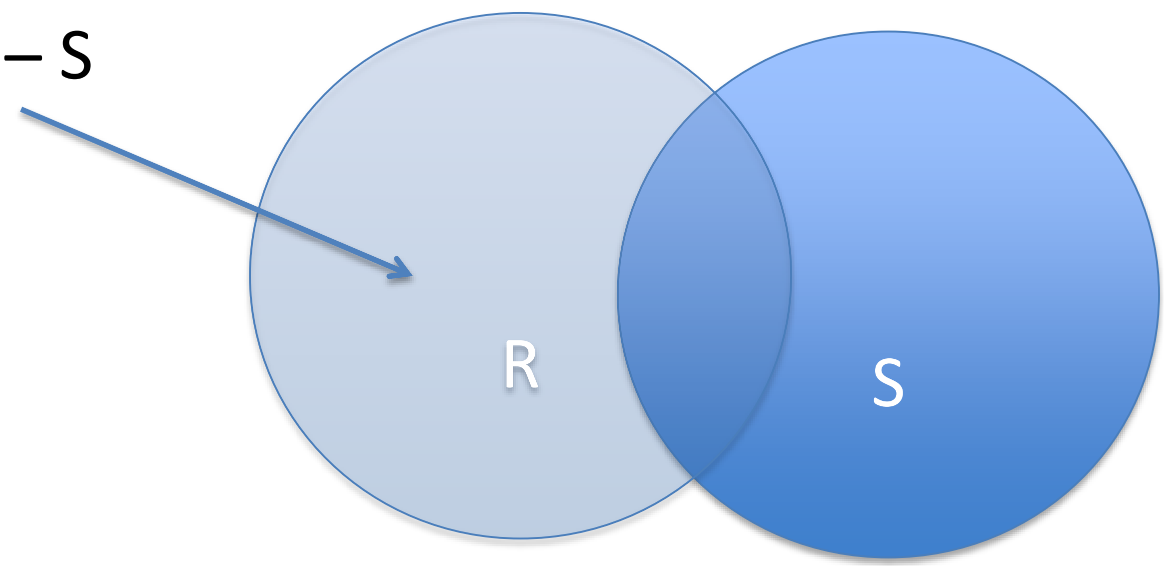
bname
Downtown
Perry

What About Intersection?

Remember:

$$R \cap S = R - (R - S)$$

R – S



Cartesian Product (×)

Notation: $Relation_1 \times Relation_2$

R × S like cross product for mathematical relations:

- *every tuple of R appended to every tuple of S*
- *flattened!!!*

Example:

depositor × borrower =

depositor. cname	acct_no	borrower. cname	lno
Johnson	A-101	Jones	L-17
Johnson	A-101	Smith	L-23
Johnson	A-101	Hayes	L-15
Johnson	A-101	Jackson	L-14
Johnson	A-101	Curry	L-93
Johnson	A-101	Smith	L-11
Johnson	A-101	Williams	L-17
Johnson	A-101	Adams	L-16
Smith	A-215	Jones	L-17
…	…	…	…

How many tuples in the result?

A: 56

Rename (ρ)

Notation: $\rho_{\text{identifier}}\ (\mathit{Relation})$

renames a relation, or

Notation: $\rho_{\text{identifier}_0\ (\text{identifier}_1,\ \ldots,\ \text{identifier}_n)}\ (\mathit{Relation})$

renames relation and columns of n-column relation

Use:

massage relations to make $\cup$, $-$ valid, or $\times$ more readable

Rename (ρ)

Notation: $\rho_{\text{identifier}_0\,(\text{identifier}_1,\,\ldots,\,\text{identifier}_n)}\,(\textit{Relation})$

Example:

$\rho_{\text{result (dcname, acctno, bcname, lno)}}\,(\text{depositor} \times \text{borrower}) =$

result

dccname	**acctno**	**bcname**	**lno**
Johnson	A-101	Jones	L-17
Johnson	A-101	Smith	L-23
Johnson	A-101	Hayes	L-15
Johnson	A-101	Jackson	L-14
Johnson	A-101	Curry	L-93
Johnson	A-101	Smith	L-11
Johnson	A-101	Williams	L-17
Johnson	A-101	Adams	L-16
Smith	A-215	Jones	L-17
…	…	…	…

Example Query in RA

- *Determine **lno** for loans that are for an amount that is larger than the amount of some other loan. (i.e. **lno** for all non-minimal loans)*

Can do in steps:

$\text{Temp}_1 \leftarrow \ldots$

$\text{Temp}_2 \leftarrow \ldots \text{Temp}_1 \ldots$

$\ldots$

Example Query in RA

1. Find the base data we need

$$Temp_1 \leftarrow \pi_{\text{lno,amt}} (\text{loan})$$

lno	amt
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-93	500
L-11	900
L-16	1300

2. Make a copy of (1)

$$Temp_2 \leftarrow \rho_{\text{Temp2 (lno2,amt2)}} (Temp_1)$$

lno2	amt2
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-93	500
L-11	900
L-16	1300

Example Query in RA

3. Take the cartesian product of 1 and 2

$$Temp_3 \leftarrow Temp_1 \times Temp_2$$

lno	amt	lno2	amt2
L-17	1000	L-17	1000
L-17	1000	L-23	2000
…	…	…	…
L-17	1000	L-16	1300
L-23	2000	L-17	1000
L-23	2000	L-23	2000
…	…	…	…
L-23	2000	L-16	1300
…	…	…	…

Example Query in RA

4. Select non-minimal loans

$$Temp_4 \leftarrow \sigma_{amt > amt2} (Temp_3)$$

5. Project on lno

$$Result \leftarrow \pi_{lno} (Temp_4)$$

… or, if you prefer…

- $\pi_{lno} (\sigma_{amt > amt2} (\pi_{lno,amt} (loan) \times (\rho_{Temp2\ (lno2,amt2)} (\pi_{lno,amt} (loan)))))$

Review

Theoretical Query Languages

Relational Algebra

1. SELECT (σ)
2. PROJECT (π)
3. UNION ($\cup$)
4. SET DIFFERENCE (–)
5. CARTESIAN PRODUCT ($\times$)
6. RENAME (ρ)

- *Relational algebra gives semantics to practical query languages*
- *Above set: minimal relational algebra*

→ *will now look at some redundant (but useful!) operators*

Review

Express the following query in the RA:

Find the names of customers who have both accounts and loans

$T_1 \leftarrow \rho_{T1\ (cname2,\ lno)}\ (borrower)$

$T_2 \leftarrow depositor \times T_1$

$T_3 \leftarrow \sigma_{cname\ =\ cname2}\ (T_2)$

$Result \leftarrow \pi_{\ cname}\ (T_3)$

Above sequence of operators (ρ, ×, σ) very common.

Motivates additional (redundant) RA operators.

Relational Algebra
Additional Operators

1. Natural Join ($\bowtie$)
2. Division ($\div$)
3. Generalized Projection (π)
4. Aggregation
5. Outer Joins ($⟖$ $⟕$ $⟗$)
6. Update ($\leftarrow$) (we've already been using this)

- *1&2 Redundant: Can be expressed in terms of minimal RA*
  e.g. depositor $\bowtie$ *borrower* =
  $\pi \ldots(\sigma\ldots(\text{depositor} \times \rho\ldots(\text{borrower})))$
- *3 – 6 Added for extra power*

Natural Join

Notation: $Relation_1 \bowtie Relation_2$

Idea: combines ρ, ×, σ

r

A	B	C	D
1	α	+	10
2	α	-	10
2	α	-	20
3	β	+	10

$\bowtie$

s

E	B	D
‘a’	α	10
‘a’	α	20
‘b’	β	10
‘c’	β	10

=

A	B	C	D	E
1	α	+	10	‘a’
2	α	-	10	‘a’
2	α	-	20	‘a’
3	Β	+	10	‘b’
3	β	+	10	‘c’

$$\text{depositor} \bowtie \text{borrower}$$

$$\equiv$$

$$\pi_{\text{cname,acct\_no,lno}} (\sigma_{\text{cname=cname2}} (\text{depositor} \times \rho_{t(\text{cname2,lno})} (\text{borrower})))$$

Division

Notation: *Relation*$_1$ ÷ *Relation*$_2$

Idea: expresses "for all" queries

r

A	B
α	1
α	2
α	3
β	1
γ	1
γ	3
γ	4
γ	6
δ	1
δ	2

÷

s

B
1
2

=

A
α
δ

Query: Find values for A in r which have corresponding B values for all B values in s

Division

Another way to look at it: $\div$ and $\times$

$17 \div 3 = 5$

The largest value of i such that: $i \times 3 \leq 17$

Relational Division

r

A	B
α	1
α	2
α	3
β	1
γ	1
γ	3
γ	4
γ	6
δ	1
δ	2

$\div$

s

B
1
2

=

t

A
α
δ

The largest value of t such that: $(t \times s \subseteq r)$

Division

A More Complex Example

r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

÷ *s*

D	E
a	1
b	1

= *t*

A	B	C
α	a	γ
γ	a	γ

?

Division Adds No Power

Definition in terms of the basic algebra operation
Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \prod_{R-S} (r) - \prod_{R-S} ((\prod_{R-S} (r) \times s) - \prod_{R-S,S}(r))$$

To see why

- $\prod_{R-S,S}(r)$ simply reorders attributes of r
- $\prod_{R-S}(\prod_{R-S} (r) \times s) - \prod_{R-S,S}(r))$ gives those tuples t in

  $\prod_{R-S} (r)$ such that for some tuple $u \in s$, $tu \notin r$.

Generalized Projection

Notation: $\pi_{e_1,\ldots,e_n}$ (*Relation*)

$e_1, \ldots, e_n$ can include arithmetic expressions – not just attributes

Example

credit =

cname	limit	balance
Jones	5000	2000
Turner	3000	2500

Then…

$\pi_{\text{cname, limit - balance}}$ (credit) =

cname	limit-balance
Jones	3000
Turner	500

Aggregate Functions and Operations

- An **aggregate 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

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

  - E is any relational-algebra expression
  - G_1, G_2 …, 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

$$\mathcal{g}_{\text{sum(c)}}(\text{r})$$

sum-C
27

No grouping

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{\textit{branch-name}}\,\mathcal{G}_{\,\textit{sum(balance)}}\,(\textit{account})$$

branch-name	*sum(balance)*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

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

$$_{\textit{branch-name}}\, \mathcal{G}\, _{\textit{sum(balance)}\ \underline{\textit{as sum-balance}}}\,(\textit{account})$$

Outer Joins

Motivation:

loan =

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700

borrower =

cname	lno
Jones	L-170
Smith	L-230
Hayes	L-155

=

loan $\bowtie$ borrower =

bname	lno	amt	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

Join result loses…

→ *any record of Perry*
→ *any record of Hayes*

Outer Joins

loan =

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700

borrower =

cname	lno
Jones	L-170
Smith	L-230
Hayes	L-155

1. Left Outer Join (⟕)

- *preserves all tuples in left relation*

loan ⟕ borrower =

bname	lno	amt	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perry	L-260	1700	$\perp$

$\perp$ = NULL

Outer Joins

loan =

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700

borrower =

cname	lno
Jones	L-170
Smith	L-230
Hayes	L-155

2. Right Outer Join ($⟖$)

- *preserves all tuples in right relation*

loan $⟖$ borrower =

bname	lno	amt	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
$\perp$	L-155	$\perp$	Hayes

$\perp$ = NULL

Outer Joins

loan =

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700

borrower =

cname	lno
Jones	L-170
Smith	L-230
Hayes	L-155

3. Full Outer Join (⟗)

- *preserves all tuples in both relations*

loan ⟗ borrower =

bname	lno	amt	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perry	L-260	1700	$\perp$
$\perp$	L-155	$\perp$	Hayes

$\perp$ = NULL

Update

Notation: *Identifier* $\leftarrow$ *Query*

Common Uses:

1. Deletion: $r \leftarrow r - s$
   e.g., account $\leftarrow$ account $- \sigma_{bname=Perry}$ (account)
   (deletes all Perry accounts)

2. Insertion: $r \leftarrow r \cup s$
   e.g., branch $\leftarrow$ branch $\cup$ {(Waltham, Boston, 7M)}
   (inserts new branch with
   bname = Waltham, bcity = Boston, assets = 7M)

   e.g., depositor $\leftarrow$ depositor $\cup$ ($\rho_{temp\ (cname,acct\_no)}$ (borrower))
   (adds all borrowers to depositors, treating lno's as acct_no's)

3. Update: $r \leftarrow \pi_{e1,\ldots,en}$ (r)
   e.g., account $\leftarrow \pi_{bname,acct\_no,bal*1.05}$ (account)
   (adds 5% interest to account balances)

Views

- Limited access to DB.
- Tailored schema

- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

$$\Pi_{\textit{customer-name, loan-number}} (\textit{borrower} \bowtie \textit{loan})$$

- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a *view*.

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** <query expression>

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v.*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

- View definition is not the same as creating a new relation by evaluating the query expression Rather, a view definition causes the saving of an expression to be substituted into queries using the view.

View Examples

- Consider the view (named *all-customer*) consisting of branches and their customers.

  create view *all-customer* **as**

  $$\Pi_{\textit{branch-name, customer-name}}\ (\textit{depositor} \bowtie \textit{account})$$
  $$\cup\ \Pi_{\textit{branch-name, customer-name}}\ (\textit{borrower} \bowtie \textit{loan})$$

- We can find all customers of the Perryridge branch by writing:

  $$\Pi_{\textit{customer-name}}$$
  $$(\sigma_{\textit{branch-name} = \text{“Perryridge”}}\ (\textit{all-customer}))$$

Updates Through View

- Database modifications expressed as views must be translated to modifications of the actual relations in the database.
- Consider the person who needs to see all loan data in the *loan* relation except *amount.* The view given to the person, *branch-loan,* is defined as:

  create view *branch-loan* **as**

  $$\Pi_{branch\text{-}name,\ loan\text{-}number}\ (loan)$$

- Since we allow a view name to appear wherever a relation name is allowed, the person may write:

$$branch\text{-}loan \leftarrow branch\text{-}loan \cup \{(\text{“Perryridge”, L-37})\}$$

Updates Through Views (Cont.)

- The previous insertion must be represented by an insertion into the actual relation *loan* from which the view *branch-loan* is constructed.
- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either.
  - rejecting the insertion and returning an error message to the user.
  - inserting a tuple (“L-37”, “Perryridge”, *null*) into the *loan* relation

Updates Through Views (Cont.)

- Some updates through views are impossible to translate into database relation updates
  - create view v as $\sigma_{\textit{branch-name} = \text{“Perryridge”}}$ (*account*))

    $v \leftarrow v \cup$ (L-99, Downtown, 23)
- Others cannot be translated uniquely
  - *all-customer* $\leftarrow$ *all-customer* $\cup$ (Perryridge, John)
    - Have to choose loan or account, and create a new loan/account number!

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly on v_2 or there is a path of dependencies from v_1 to v_2

View Expansion

- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

  repeat
     Find any view relation v_i in e_1
     Replace the view relation v_i by the expression defining v_i
  until no more view relations are present in e_1
- As long as the view definitions are not recursive, this loop will terminate