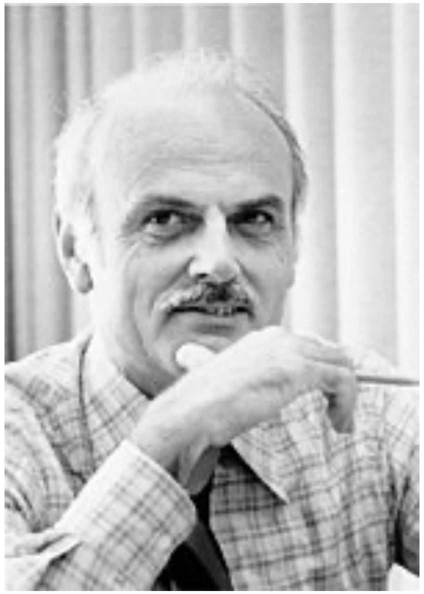
RELATIONAL ALGEBRA
& RELATIONAL CALCULUS

2110322 Database Systems
@ Chula Engineering

Modified from Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

WHY IS RELATIONAL ALGEBRA IMPORTANT?

- Relational algebra theory was introduced by Edgar F. Codd.
- The relational algebra uses concepts from set theory *(that you learned in Discrete Math!)*, and adds additional constraints.
- It is the theoretical foundation of the relational data model and query languages.



QUERY LANGUAGES

allow manipulation and retrieval of data from a database.

Early versions of SQL:

- Query Languages != programming languages
- QLs not expected to be "Turing complete".
- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.

Now!!!:

- SQL is a domain-specific programming language that is Turing complete.
- Thus qualifying SQL as a programming language.

Latest version is SQL:2023

FORMAL RELATIONAL QUERY LANGUAGES

Two mathematical Query Languages form the basis for **real** languages (e.g. SQL), and for implementation:

- **Relational Algebra**
  - Operational (procedural), useful for representing execution plans.
- **Relational Calculus**
  - Declarative (non-procedural), lets users describe what they want, rather than how to compute it.

EXAMPLE INSTANCES

STUDENT (<u>*sid*:integer</u>, *sname*:string, *rating*:integer, age:real)

BOAT (<u>*bid*:integer</u>, *bname*:string, *color*:string)

RESERVE (<u>*sid*:integer</u>, <u>*bid*:integer</u>, <u>*day*:date</u>)

Instance S1 of *STUDENT*

<u>sid</u>	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

Instance S2 of *STUDENT*

<u>sid</u>	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

Instance B1 of *BOAT*

<u>bid</u>	bname	color
101	สุพรรณ	ทอง
103	หงส์	ขาว
105	โบ้ท	ส้ม

Instance R1 of *RESERVE*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/20
58	103	11/12/20

IMPORTANT NOTE ON NOTATION

ENG_STUDENT

student_name	*student_id*
ลิซ่า	6130000121
เจนนี่	6130000221
วี	6130000321
โรเซ่	6130000421

SCI_STUDENT

student_name	*student_id*
จีมิน	6130000521
เจ-โฮป	6130000621
วี	6130000721
จองกุก	6130000821

When you see operators between lowercase letters…

$$r - s$$

They refer to relational algebra operators

eng_student – sci_student

student_name	*student_id*
ลิซ่า	6130000121
เจนนี่	6130000221
โรเซ่	6130000421

When you see operators between all uppercase letters…

$$R - S$$

They refer to operators on schemas

$R = (student\_id, age)$
$S = (student\_id)$

$R\text{-}S = (age)$

DOT NOTATION

ENG_STUDENT

student_name	***student_id***
ลิซ่า	6130000121
เจนนี่	6130000221
วี	6130000321
โรเซ่	6130000421

SCI_STUDENT

student_name	***student_id***
จีมิน	6130000521
เจ-โฮป	6130000621
วี	6130000721
จองกุก	6130000821

If we have two or more relations which feature the same attribute names, we could confuse them. To prevent this, we can use **dot notation**.

ENG_STUDENT.student_id

RELATIONAL ALGEBRA OPERATION

- Operation can be unary or binary
- Every operation in algebra accepts one (unary) or two (binary) relation instances as arguments and returns a relation instance as a result.
- Since each operation returns a relation, operations can be composed.
- Queries in algebra are composed using a collection of operators.

RELATIONAL ALGEBRA OPERATORS

Basic operators

Selection σ

Retrieves some rows from relation

Projection π

Retrieves desired columns from relation

Cross-product x

Combines two relations

Set-difference –

Eliminate tuples in relation 1 that are not in relation 2

Union ∪

Combines tuples from 2 compatible relations

RELATIONAL ALGEBRA OPERATORS

Additional operators (Not essential, but very useful)

- Join $\bowtie$
- Intersection $\cap$
- Division / or $\div$
- Renaming ρ

SELECTION

$$\sigma_{\text{selection condition}}(RELATION)$$

Selects rows that satisfy selection condition.

Schema of result = schema of input relation.

S2

<u>sid</u>	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

$$\sigma_{rating > 8}(s2)$$

<u>sid</u>	sname	rating	age
28	ลิซ่า	10	23
58	โรเซ่	10	23

SELECTION CONDITION

Selection condition is a Boolean combination of terms that have the forms:

$$\textit{attribute } \mathbf{op} \textit{ constant}$$

$$\textit{attribute1 } \mathbf{op} \textit{ attribute2}$$

- Comparison operators are $=, \neq, <, \leq, >, \geq$
- Terms are combined with $\wedge, \vee$

$$\sigma_p(r) = \{\forall\, t \in r \mid p(t)\}$$

PROJECTION

$$\pi_{\text{list of attributes separated by comma}}(RELATION)$$

Deletes unwanted attributes (not in *projection list*.)

Schema of the result contains only the fields in the projection list, with the same names.

Projection operator eliminates duplicates.

Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it.

S2

sid	*sname*	*rating*	*age*
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

$\pi_{sname,\ rating}(s2)$

sname	*rating*
ลิซ่า	10
เจนนี่	8
วี	5
โรเซ่	10

$\pi_{age}(s2)$

age
23
24

UNION, INTERSECTION, SET-DIFFERENCE

Union, Intersection, and Set-difference operations take two input relations, which must be union-compatible:

- Same number of fields (attributes).
- Corresponding fields have the same type.
- Although field types must be the same (same schema), the names can be different.

Schema of the result = schema of the inputs.

UNION

$$r \cup s$$

The **union** operation concatenates two relations and removes duplicate rows (since relations are sets).

$$r \cup s = \{t \mid t \in r \vee t \in s\}$$

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

S2

sid	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

$s1 \cup s2$

sid	sname	rating	age
22	จิซู	7	25
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

Can we union s1 and r1?

R1

sid	bid	day
22	101	10/10/20
58	103	11/12/20

INTERSECTION

$$r \cap s$$

The **intersection** operation returns all the rows that are in both *relation1* and *relation2*. Note: $r \cap s = r - (r - s)$

$$r \cap s = \{ t \mid t \in r \wedge t \in s \}$$

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

S2

sid	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

s1 ∩ s2

sid	sname	rating	age
31	เจนนี่	8	24
58	โรเซ่	10	23

SET-DIFFERENCE

$$r - s$$

The **set-difference** operation returns all the rows that are in *R* but not in *S*.

$$r - s = \{t \mid t \in r \wedge t \notin s\}$$

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

S2

sid	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

s1- s2

sid	sname	rating	age
22	จิซู	7	25

$$r - s \neq s - r$$

CROSS-PRODUCT

$$r \times s$$

The **cross-product** operation returns all possible combinations of rows in R with rows in S.

$$r \times s = \{<t, q> \mid t \in \mathrm{R} \wedge q \in \mathrm{S}\}$$

The result is every possible pairing of the rows of r and s.

Assume that attributes of R and S are disjoint, that is, $R \cap S = \emptyset$.

If attributes names of R and S are not disjoint, then renaming must be used.

CROSS-PRODUCT

R1

sid	bid	day
22	101	10/10/20
58	103	11/12/20

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

r1 X *s1*

(sid)	bid	day	(sid)	sname	rating	age
22	101	10/10/20	22	จิซู	7	25
22	101	10/10/20	31	เจนนี่	8	24
22	101	10/10/20	58	โรเซ่	10	23
58	103	11/12/20	22	จิซู	7	25
58	103	11/12/20	31	เจนนี่	8	24
58	103	11/12/20	58	โรเซ่	10	23

AxB = C

Cardinality(C) (number of rows) = Cardinality(A) x Cardinality(B)

Degree(C) (number of columns) = Degree(A) + Degree(B)

COMPOSITION OF OPERATION

R1

sid	bid	day
22	101	10/10/20
58	103	11/12/20

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

$r1 \text{ X } s1$

(sid)	bid	day	(sid)	sname	rating	age
22	101	10/10/20	22	จิซู	7	25
22	101	10/10/20	31	เจนนี่	8	24
22	101	10/10/20	58	โรเซ่	10	23
58	103	11/12/20	22	จิซู	7	25
58	103	11/12/20	31	เจนนี่	8	24
58	103	11/12/20	58	โรเซ่	10	23

$$\sigma_{r1.sid = s1.sid}(r1 \times s1)$$

(sid)	bid	day	(sid)	sname	rating	age
22	101	10/10/20	22	จิซู	7	25
58	103	11/12/20	58	โรเซ่	10	23

RENAME OPERATION

If a relational-algebra expression *E*, then

ρ (X(oldname$_1$ $\rightarrow$ newname$_1$ or position $\rightarrow$ newname$_1$, …), E)

returns the result of expression *E* under the name X, and with the attributes renamed to *newnames*.

Alternative notation:

ρ (X(newname$_1$, newname$_2$, …, newname$_n$), E)

Example

ρ (MYRELATION(a $\rightarrow$ e, 2 $\rightarrow$ k), $r - s$)

or

ρ (MYRELATION(e, k), $r - s$)

Take the set difference of r and s, and call the result myRelation, while renaming the first field *e* and the second field *k*.

R

a	b
x	1
x	2
y	1

S

a	b
x	2
y	3

MYRELATION

e	k
x	1
y	1

JOIN

Join $\bowtie$ is one of the most useful operations, and most commonly used way to combine information from two or more relations.

Join = Cross-product → Selection (& Projection)

Reasons to use Join:

Cross product is meaningless, and the results are much larger → waste storage.

JOINS ⋈

Condition-Join (Theta-Join)

Equijoin

Natural-Join

CONDITION (THETA) JOIN

$$r \bowtie_c s$$

Condition c can refer to attributes of both r and s

$r \bowtie_c s = \sigma_c(r \times s)$

R1

sid	bid	day
22	101	10/10/20
58	103	11/12/20

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

$(r1 \bowtie_{r1.sid<s1.sid} s1)$

(sid)	bid	day	(sid)	sname	rating	age
22	101	10/10/20	58	โรเซ่	10	23
22	101	10/10/20	31	เจนนี่	8	24

EQUIJOIN

$$r \bowtie_{\text{attribute}} s$$

A special case of condition join where the condition c contains only equalities.

$$r1 \bowtie_{sid} s1 = r1 \bowtie_{r1.sid=s1.sid} s1$$

R1

sid	bid	day
22	101	10/10/20
58	103	11/12/20

S1

sid	sname	rating	age
22	จิซู	7	25
31	เจนนี่	8	24
58	โรเซ่	10	23

$$r1 \bowtie_{sid} s1$$

sid	bid	day	sname	rating	age
22	101	10/10/20	จิซู	7	25
58	103	11/12/20	โรเซ่	10	23

Result schema is similar to cross-product, but only one copy of fields for which equality is specified.

NATURAL JOIN

$$r \bowtie s$$

A further special case of equijoin in which equalities are specified on all fields having the same name in R and S.

The results is guaranteed to have no two fields with the same name.

- If the two relations have no attributes in common, then their natural join is simply their cross product.
- If the two relations have more than one attribute in common, then the natural join selects only the rows where all pairs of matching attributes match.

A

l-name	*f-name*	*age*
Bouvier	Selma	40
Bouvier	Patty	40
Smith	Maggie	2

B

l-name	*f-name*	*id*
Bouvier	Selma	1232
Smith	Selma	4423

axb

l-name	*f-name*	*age*	*l-name*	*f-name*	*id*
Bouvier	Selma	40	Bouvier	Selma	1232
Bouvier	Selma	40	Smith	Selma	4423
Bouvier	Patty	40	Bouvier	Selma	1232
Bouvier	Patty	40	Smith	Selma	4423
Smith	Maggie	2	Bouvier	Selma	1232
Smith	Maggie	2	Smith	Selma	4423

Both the *l-name* and the *f-name* match, so select.

Only the *f-names* match, so don't select.

Only the *l-names* match, so don't select.

We remove duplicate attributes…

l-name	*f-name*	*age*	*l-name*	*f-name*	*id*
Bouvier	Selma	40	Bouvier	Selma	1232

The natural join of *A* and *B*

$$a \bowtie b =$$

l-name	*f-name*	*age*	*id*
Bouvier	Selma	40	1232

DIVISION

$r \div s$ or r/s

Not supported as a primitive operator, but useful for expressing queries like: *Find sailors who have reserved **all** boats*.

Let A have 2 fields, x and y; B have only field y:

$$a \div b = \{\langle x\rangle \mid \exists\langle x, y\rangle \in A \wedge \forall\langle y\rangle \in B\}$$

Two interpretations:

- a/b contains all x tuples (sailors) such that for every y tuple (boat) in B, there is an xy tuple in A.
- If the set of y values associated with an x value in r contains all y values in s, the x value is in r/s.

In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A.

EXAMPLES OF DIVISION A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

a/b1

sno
s1
s2
s3
s4

a/b2

sno
s1
s4

a/b3

sno
s1

ASSIGNMENT OPERATION

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries, write query as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as a result of the query.
- Assignment must always be made to a temporary relation variable.
- Example: Write *a*/*b* as

$$TEMP1 \leftarrow (\pi_x(a) \times b) - a$$

$$TEMP2 \leftarrow \pi_x(temp1)$$

$$result = \pi_x(a) - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

EXTENDED OPERATIONS

Aggregate Function

Outer Join

AGGREGATE FUNCTIONS

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

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

- 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 (i.e. avg, min, max, etc.)
- Each A_i is an attribute name

AGGREGATE FUNCTIONS

R

a	b	c
α	α	7
α	β	7
β	β	3
β	β	10

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

sum-c
27

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

a	sum-c
α	14
β	13

OUTER JOIN

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses ***null*** values:

  null signifies that the value is unknown or does not exist

  All comparisons involving *null* are (roughly speaking) **false** by definition.

(Will study precise meaning of comparisons with nulls later)

OUTER JOIN

LOAN

loan-number	***branch-name***	***amount***
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

BORROWER

customer-name	***loan-number***
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

OUTER JOIN

LOAN

loan-number	branch-name	amount
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

BORROWER

customer-name	loan-number
Simpson	L-170
Wiggum	L-230
Flanders	L-155

Inner Join
loan ⋈ borrower

loan-number	branch-name	amount	customer-name
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum

Left Outer Join
loan ⟕ borrower

loan-number	branch-name	amount	customer-name
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*

OUTER JOIN

LOAN

loan-number	branch-name	amount
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

BORROWER

customer-name	loan-number
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

Right Outer Join

loan ⟖ borrower

loan-number	branch-name	amount	customer-name
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-155	*null*	*null*	Flanders

Full Outer Join

loan ⟗ borrower

loan-number	branch-name	amount	customer-name
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*
L-155	*null*	*null*	Flanders

RELATIONAL CALCULUS

RELATIONAL CALCULUS

Relational calculus is a formal query language for relational model.

- It provides a declarative way to specify database queries (relational algebra provides a more procedural way).
- The relational algebra and the relational calculus are essentially logically equivalent: for any algebraic expression, there is an equivalent expression in the calculus, and vice versa.

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

"RETRIEVE THE PHONE NUMBERS AND NAMES OF BOOKSTORES THAT SUPPLY *HARRY POTTER*"

BOOKSTORES (*bookstoreid*, *storename*, *address*, *city*, *zip*, *phone*)
BOOK (*bookstoreid*, *isbn*, *title*, *type*, *author*, *price*, *publisher*)

Relational algebra

1. Select from BOOK for *title* = 'Harry Potter'.
2. Join BOOKSTORES and BOOK by *bookstoreid*.
3. Project the result to obtain *storename* and *phone*.

Relational calculus

Get *storename* and *phone* for supplies such that there exists a *title* with the same *bookstoreId* value and with a *title* value of 'Harry Potter'.

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS EQUIVALENCY

Find the names of all customers who have a loan at the Riverside branch.

$$\pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{'Riverside'}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

borrower	
customer-name	***loan-number***
Patty	1234
Apu	3421

loan		
loan-number	***branch-name***	***amount***
1234	Riverside	1,923.03
3421	Irvine	123.00

$$\{\langle X\rangle \mid \langle X,Y\rangle \in \text{borrower} \wedge \exists A,B,C(\langle A,B,C\rangle \in \text{loan} \wedge B = \text{'Riverside'} \wedge Y{=}A)\}$$

RELATIONAL ALGEBRA & RELATIONAL CALCULUS 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.
- The relational algebra and the relational calculus are essentially logically equivalent.