Relational Algebra

S Ranbir Singh

CS344@iitg

Chapter 4: Database Management System, 3rd Ed. Ramakrishnan & Gehrke

Relational Query Languages

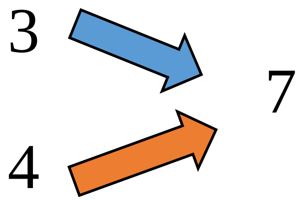
- Languages for describing queries on a relational database
- *Structured Query Language* (SQL)
  - Predominant application-level query language
  - Declarative
- *Relational Algebra*
  - Intermediate language used within DBMS
  - Procedural
- Relational Calculus:
  - Lets users describe what they want, rather than how to compute it.
  - Non-operational, declarative.

What is an Algebra?

- A language based on operators and a domain of values
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations (and the operators are as described later), we get the *relational algebra*
- We refer to the expression as a *query* and the value produced as the *query result*

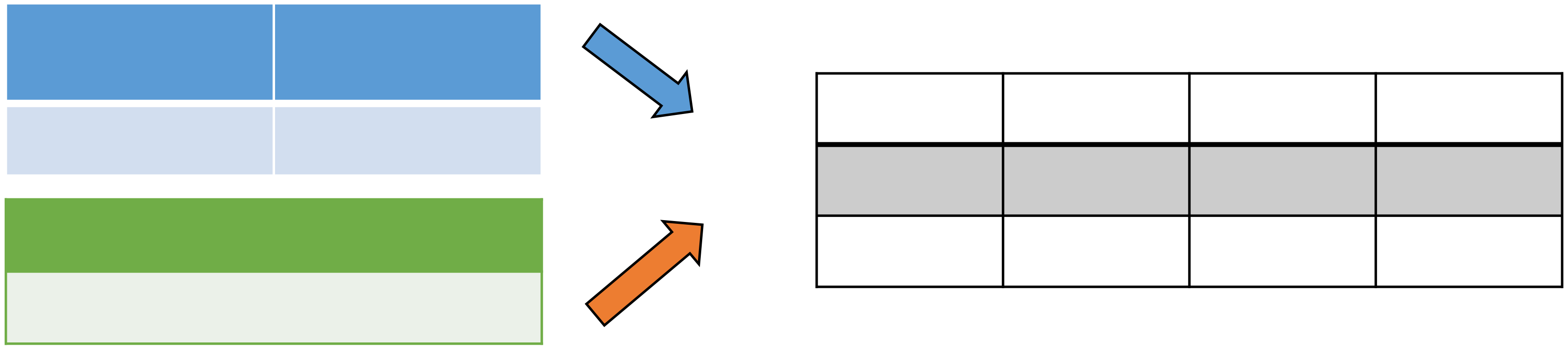
Algebra

- In math, algebraic operations like +, -, x, /.
- Operate on numbers: input are numbers, output are numbers.
- Can also do Boolean algebra on sets, using union, intersect, difference.
- Focus on **algebraic identities**, e.g.
    - $x\,(y+z) = xy + xz$.
- (Relational algebra lies between propositional and 1st-order logic.)



Relational Algebra

- Every operator takes one or two relation instances
- A relational algebra expression is recursively defined to be a relation
  - Result is also a relation
  - Can apply operator to
    - Relation from database
    - Relation as a result of another operator

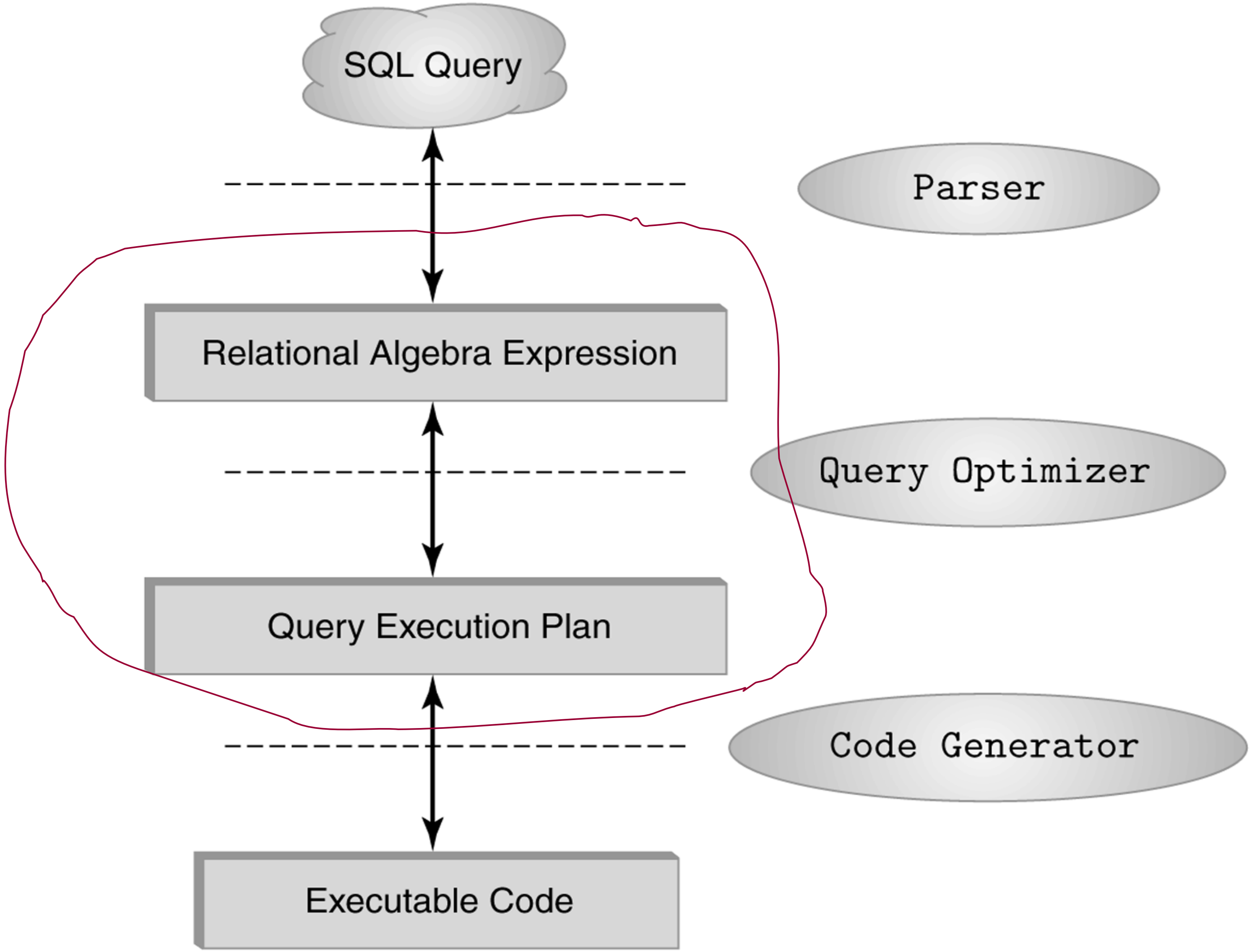


Relational Algebra Operations

- *Domain*: set of relations
- 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 derived operations:
  - Intersection, *join*, division, renaming.

- Since each operation returns a relation, operations can be *composed*!

The Role of Relational Algebra in a DBMS



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*!

Produces table containing subset of columns of argument table

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

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*!

Produces table containing subset of columns of argument table

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

Relation: S

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

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

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{age}(S)$

age
35.0
55.5

Project Operator

Result is a table (no duplicates); can have fewer tuples than the original

Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Hobby}$(Person)

Name	*Hobby*
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

SELECT Name, Address FROM Person;

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

SELECT Name, Address FROM Person;

SELECT * FROM Person; ????

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result!
- *Schema* of result identical to schema of (only) input relation.
- Selection conditions:
  - *simple conditions* comparing attribute values (variables) and / or constants or
  - *complex conditions* that combine simple conditions using logical connectives AND and OR.

Produce table containing subset of rows of argument table satisfying condition

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

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result!
- *Schema* of result identical to schema of (only) input relation.
- Selection conditions:
  - *simple conditions* comparing attribute values (variables) and / or constants or
  - *complex conditions* that combine simple conditions using logical connectives AND and OR.

Produce table containing subset of rows of argument table satisfying condition

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

Relation: S

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

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

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

Select Operator

SELECT * FROM Person WHERE Hobby='stamps'

Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby=\text{'stamps'}}$(Person)

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition

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

Select Operator

SELECT * FROM Person WHERE Hobby='stamps' AND Name='John';

Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby=\text{'stamps' AND Name='John'}}(\text{Person})$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps

Selection and Projection

$$\pi_{\mathit{Id,\ Name}} (\sigma_{\mathit{Hobby}=\text{'stamps'}\ \text{OR}\ \mathit{Hobby}=\text{'coins'}} (\text{Person}))$$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*
1123	John
9876	Bart

Result

SELECT Id, Name FROM Person WHERE Hobby=‘stamps’ OR Hobby=‘coin’

Selection and Projection

Relation: S

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

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

sname	rating
yuppy	9
rusty	10

SELECT sname, rating FROM S WHERE rating > 8

Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Boats(*bid*: integer, *bnarne*: string, *coloT*: string)
Reserves(*sid*: integer, *bid*: integer, *day*: date)

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

An Instance of Boats

sid		*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

An Instance of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

An Instance of Reserves

Find names of sailors who've reserved boat #103

$$\pi_{sname}(\sigma_{sid=Rsid}(\sigma_{bid=103}\rho(T1(4\rightarrow Rsid),Reserves\times Sailors)))$$

$$\pi_{sname}(\sigma_{sid=Rsid\ AND bid=103}\rho(T1(4\rightarrow Rsid),Reserves\times Sailors))$$

Set Operators

- Relation is a set of tuples, so set operations should apply: $\cap$, $\cup$, $-$ (set difference)
- Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- Hence, scope of set operations limited to *union compatible relations*

Union Compatible Relations

- Two relations are *union compatible* if
  - Both have 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 <u>not</u> union compatible.

But

π_{Name} (Person) and π_{Name} (Professor)

<u>are</u> union compatible so

π_{Name} (Person) - π_{Name} (Professor)

makes sense.

Union, Intersection, Set-Difference

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

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

$S1 \cap S2$

UNION, INTERSECT, EXCEPT

$$S1 \cup S2$$

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

$$S1 \cap S2$$

```
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

$$S1 - S2$$

```
SELECT column_name(s) FROM table1
MINUS
SELECT column_name(s) FROM table2;
```

```
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

UNION, INTERSECT, EXCEPT

$$S1 \cup S2$$

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

$$S1 \cap S2$$

```
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

$$S1 - S2$$

```
SELECT column_name(s) FROM table1
MINUS
SELECT column_name(s) FROM table2;
```

```
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

MySQL

```
select a.id from table1 as a where <condition>
        AND a.id NOT IN
                    (select b.id from table2 as b where <condition>);
```

UNION, INTERSECT, EXCEPT

$$S1 \cup S2$$

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

$$S1 \cap S2$$

```
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

$$S1 - S2$$

```
SELECT column_name(s) FROM table1
MINUS
SELECT column_name(s) FROM table2;
```

```
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

Renaming

MySQL

```
select a.id from table1 as a where <condition>
        AND a.id NOT IN
                    (select b.id from table2 as b where <condition>);
```

EXCEPT/MINUS

$$S1 - S2$$

```
SELECT column_name(s) FROM table1
MINUS
SELECT column_name(s) FROM table2;
```

```
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

MySQL

```
SELECT * FROM table1 AS a WHERE <condition>
        AND a.id NOT IN
                    (SELECT b.id FROM table2 AS b WHERE <condition>);
```

OR

```
SELECT * FROM table1 a WHERE NOT EXISTS
        ( SELECT * FROM table2 b WHERE a.id = b.id);
```

Cartesian Product

- If *R* and *S* are two relations, $R \times S$ is the set of all concatenated tuples <*x*,*y*>, where *x* is a tuple in *R* and *y* is a tuple in *S*
  - *R* and *S* need not be union compatible.
  - *But R* and *S* must have distinct attribute names.

A	*B*
x1	x2
x3	x4

R

A	*C*
y1	y2
y3	y4

S

A	*B*	*A*	*C*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

Cartesian Product

- If *R* and *S* are two relations, $R \times S$ is the set of all concatenated tuples <*x*,*y*>, where *x* is a tuple in *R* and *y* is a tuple in *S*
  - *R* and *S* need not be union compatible.
  - *But R* and *S* must have distinct attribute names.

A	*B*
x1	x2
x3	x4

R

A	*C*
y1	y2
y3	y4

S

A	*B*	*A*	*C*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

```
SELECT * FROM R, S;
```

Renaming

- *Renaming operator*: $\rho(C(1 \rightarrow X, 3 \rightarrow Y), R \times S)$

$$\rho(\Pi_{A, \rho(A \rightarrow X)}(R) \times \Pi_{A, \rho(A \rightarrow Y)}(S))$$

R

A	*B*
x1	x2
x3	x4

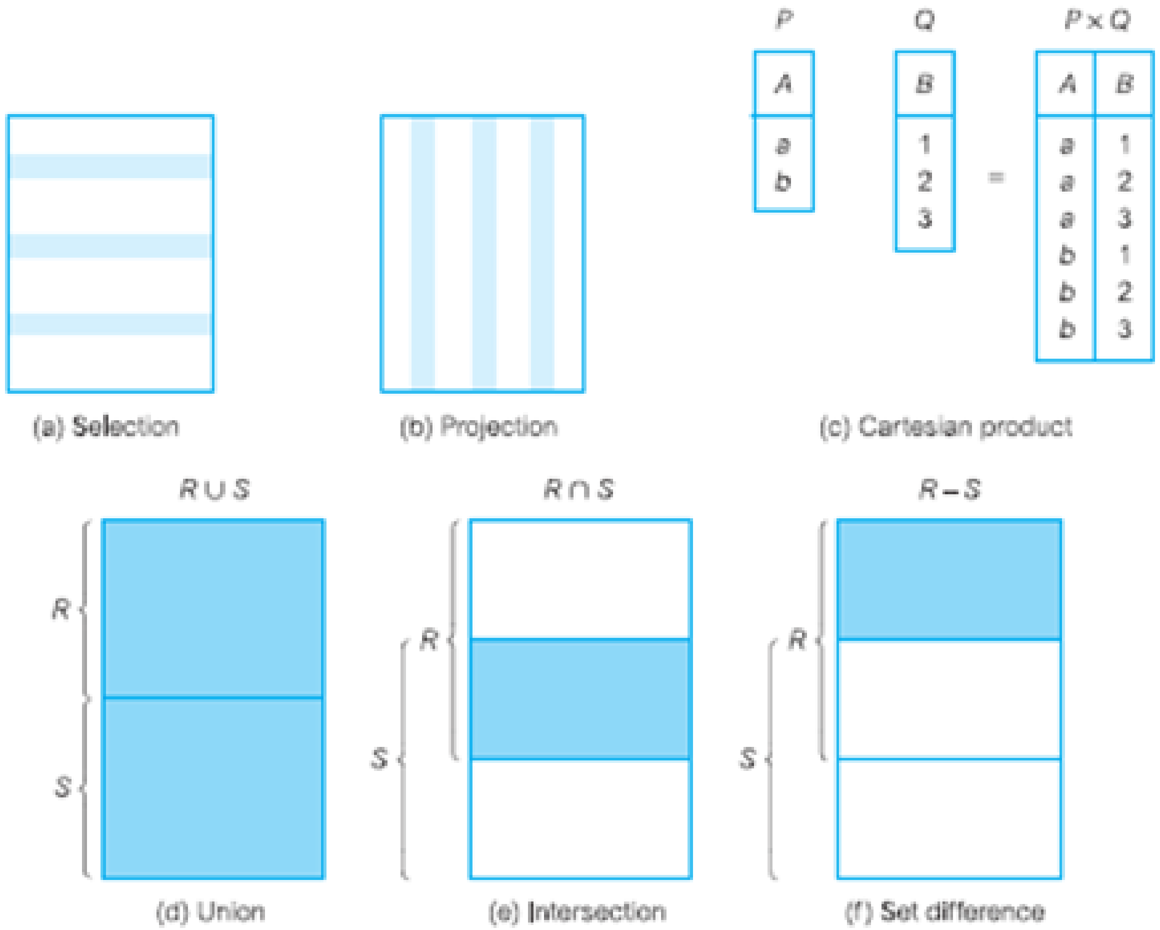
S

A	*C*
y1	y2
y3	y4

$R \times S$

X	*B*	*Y*	*C*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

Summary


(a) Selection
(b) Projection
P
Q
P × Q
A
B
A
B
a
b
1
2
3
=
a
a
a
b
b
b
1
2
3
1
2
3
(c) Cartesian product
R ∪ S
R
S
(d) Union
R ∩ S
R
S
(e) Intersection
R − S
R
S
(f) Set difference

Examples

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Find those professors who have taught ‘csc6710’ but never ‘csc7710’.

Examples

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Find those professors who have taught ‘csc6710’ but never ‘csc7710’.

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’} \wedge crscode!=\text{‘csc7710’}}(Taught))$$

Examples

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

Find those professors who have taught ‘csc6710’ but never ‘csc7710’.

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’} \cancel{\wedge}\, crscode!=\text{‘csc7710’}}(\text{Taught}))$$

Examples

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Find those professors who have taught 'csc6710' but never 'csc7710'.

$$\pi_{ssn}(\sigma_{crscode=\text{'csc6710'}}(\text{Taught}))-$$
$$\pi_{ssn}(\sigma_{crscode=\text{'csc7710'}}(\text{Taught}))$$

Examples

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Find those professors who have taught ‘csc6710’ but never ‘csc7710’.

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’}}(Taught))-$$
$$\pi_{ssn}(\sigma_{crscode=\text{‘csc7710’}}(Taught))$$

```
SELECT x.ssn FROM Taught AS x WHERE x.crscode=‘csc6710’ AND x.ssn
          NOT IN
                   (SELECT ssn FROM Taught WHERE y.crscode=‘csc7710’);
```

```
SELECT x.ssn FROM Taught AS x WHERE x.crscode=‘csc6710’ AND
          NOT EXIST
                   (SELECT y.ssn FROM Taught AS y WHERE y.crscode=‘csc7710’ AND x.ssn=y.ssn);
```

Example

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Return those professors who have taught both 'csc6710' and 'csc7710'.

Example

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Return those professors who have taught both 'csc6710' and 'csc7710'.

$$\pi_{ssn}(\sigma_{crscode=\text{'csc6710'} \wedge crscode=\text{'csc7710'}}(Taught))$$

Example

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Return those professors who have taught both ‘csc6710’ and ‘csc7710’.

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’} \wedge crscode=\text{‘csc7710’}}(Taught))$$

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’}}(Taught)) \cap \pi_{ssn}(\sigma_{crscode=\text{‘csc7710’}}(Taught))$$

```
SELECT ssn FROM Taught WHERE crscode=‘csc6710’
        INTERSECT
                SELECT ssn FROM Taught WHERE crscode=‘csc7710’;
```

Example

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Return those professors who have taught both ‘csc6710’ and ‘csc7710’.

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’} \wedge crscode=\text{‘csc7710’}}(Taught))$$

$$\pi_{ssn}(\sigma_{crscode=\text{‘csc6710’}}(Taught)) \cap$$
$$\pi_{ssn}(\sigma_{crscode=\text{‘csc7710’}}(Taught))$$

```
SELECT x.ssn FROM Taught AS x WHERE x.crscode=‘csc6710’ AND
        x.ssn IN
                (SELECT y.ssn FROM Taught AS y WHERE y. crscode=‘csc7710’);
```

Example

Professor(<u>ssn</u>, profname, status)

Course(<u>crscode</u>, crsname, credits)

Taught(<u>crscode, semester</u>, ssn)

Return those professors who have taught both 'csc6710' and 'csc7710'.

$$\pi_{ssn}(\sigma_{crscode='csc6710' \wedge crscode='csc7710'}(Taught))$$

$$\pi_{ssn}(\sigma_{crscode='csc6710'}(Taught)) \cap$$
$$\pi_{ssn}(\sigma_{crscode='csc7710'}(Taught))$$

```
SELECT x.ssn FROM Taught AS x WHERE x.crscode='csc6710' AND
      EXIST
           (SELECT y.ssn FROM Taught AS y WHERE y.crscode='csc7710' AND x.ssn=y.ssn);
```

Derived Operators Join and Division

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

$T \bowtie U$

A	B	C
a	1	x
a	1	y

(g) Natural join

$T \triangleright_B U$

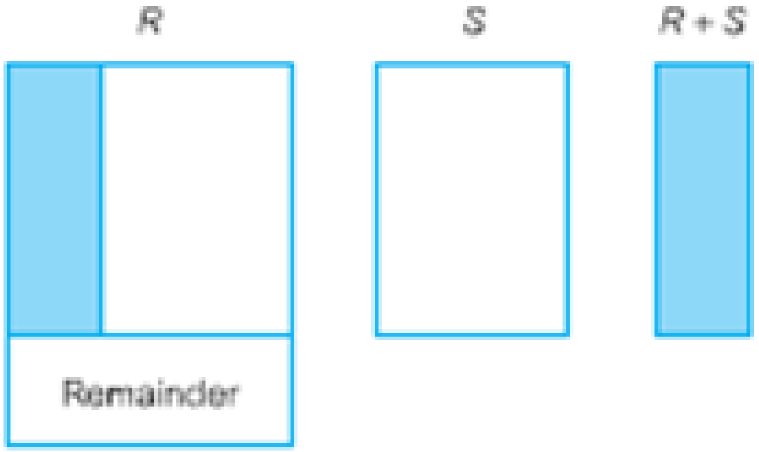
A	B
a	1

(h) Semijoin

$T ⟕_B U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join



(j) Division (shaded area)

V

A	B
a	1
a	2
b	1
b	2
c	1

W

B
1
2

$V \div W$

A
a
b

Example of division

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, might be able to compute more efficiently.
- 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

$$S1 \bowtie_{R.sid = S.sid} R1$$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.
  Without specified condition
  means the natural join of A and B. $A \bowtie B$

Natural Join (cont'd)

- More generally:

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

where

$attr\text{-}list = attributes\ (R) \cup attributes\ (S)$

(duplicates are eliminated) and *join-cond* has the form:

$R.A_1 = S.A_1$ AND … AND $R.A_n = S.A_n$

where

$\{A_1 \ldots A_n\} = attributes(R) \cap attributes(S)$

SQL JOIN

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

SQL JOIN

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
    - INNER JOIN
    - OUTER JOIN
        - LEFT OUTER JOIN
        - RIGHT OUTER JOIN
        - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name	loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

SQL JOIN

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

SQL JOIN

loan-number	branch-name	amount
L-170 L-230 L-260	Downtown Redwood Perryridge	3000 4000 1700

loan

customer-name	loan-number
Jones Smith Hayes	L-170 L-230 L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name	loan-number
L-170 L-230	Downtown Redwood	3000 4000	Jones Smith	L-170 L-230

loan-number	branch-name	amount	customer-name
L-170 L-230	Downtown Redwood	3000 4000	Jones Smith

SQL JOIN

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name	loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	null	null

SQL JOIN

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name	loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
null	null	null	Hayes	L-155

SQL JOIN

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

SELECT * FROM table1 join_type table2 [ON (join_condition)]

Types of JOINs

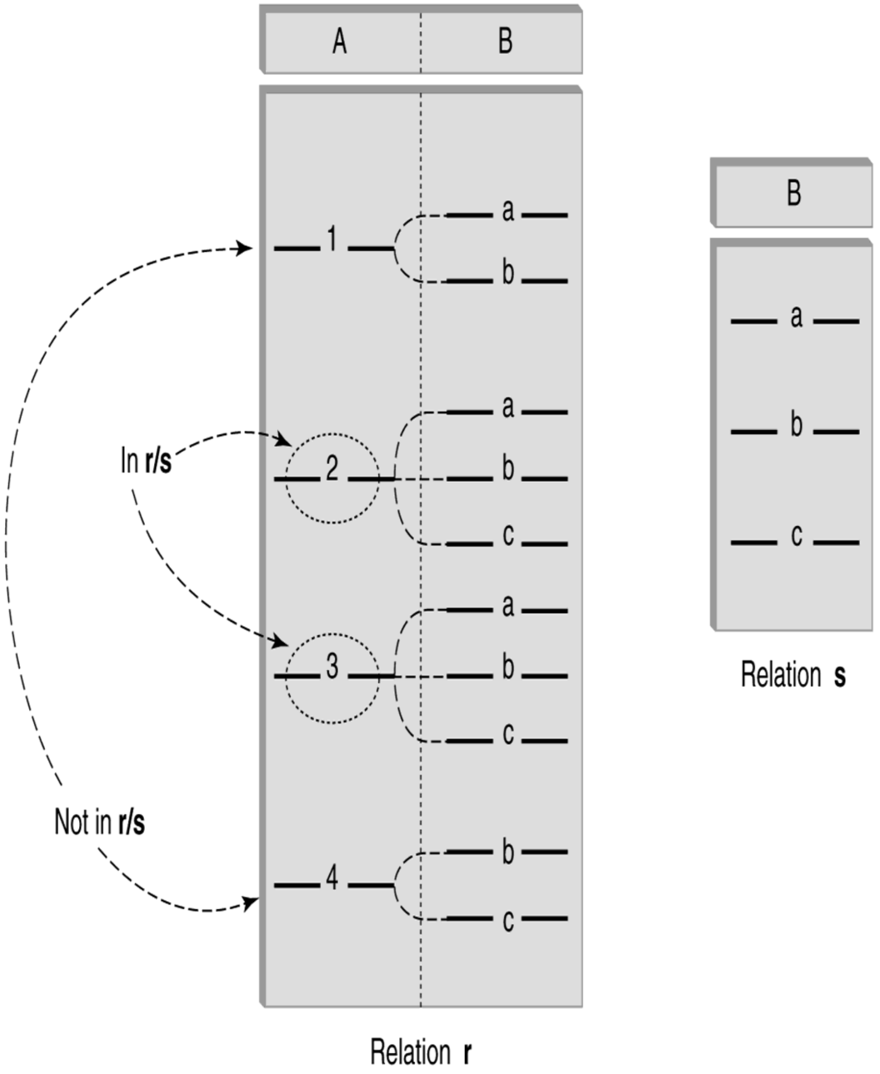
- EQUI JOIN
  - INNER JOIN
  - OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- NON EQUI JOIN
- NATURAL JOIN
- CROSS JOIN

loan-number	branch-name	amount	customer-name	loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	null	null
null	null	null	Hayes	L-155

Division

- Goal: Produce the tuples in one relation r, that match *all* tuples in another relation s
  - $r\ (A_1, \ldots A_n, B_1, \ldots B_m)$
  - $s\ (B_1 \ldots B_m)$
  - *r/s*, with attributes $A_1, \ldots A_n$, is the set of all tuples *<a>* such that for every tuple *<b>* in *s*, *<a,b>* is in *r*

Example: List of sailors who've reserved all boats



Examples of Division A/B

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

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Division

Given two relations(tables): R(x,y) , S(y).

R(x,y) div S(y) means gives all distinct values of x from R that are associated with all values of y in S.

Computation of Division : R(x,y) div S(y)
Steps:

- Find out all possible combinations of S(y) with R(x) by computing R(x) x(cross join) S(y)
  - r1 = R(x) X S(y)

- Subtract actual R(x,y) from r1, say r2
  - r2 = r1 –R(x,y)

$$\pi_X((\pi_X(A)\times B)-A)$$

- x in r2 are those that are not associated with every value in S(y); therefore R(x)-r2(x) gives us x that are associated with all values in S

$$\pi_X(A) - \pi_X((\pi_X(A)\times B)-A)$$

Find the names of sailors who've reserved all boats

$$\rho\ (Tempsids, (\pi_{sid,bid} \mathrm{Re}serves) \,/\, (\pi_{bid}\ Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

Find the names of sailors who've reserved all boats

$$\rho\ (Tempsids, (\pi_{sid,bid} Reserves) \ / \ (\pi_{bid}\ Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

To find sailors who've reserved all 'red boats:

$$\ldots\ldots \qquad / \ p_{bid}(s_{color='red'}\ Boats)$$

Find the names of sailors who've reserved all boats

$$\rho\ (Tempsids, (\pi_{sid,bid} \text{Reserves}) \,/\, (\pi_{bid}\ Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

```
SELECT sid FROM Reserves WHERE bid NOT IN
   ( SELECT sid FROM (
                        (SELECT sid , bid FROM
                            (  SELECT bid FROM Boats                    //S(y) x R(x)
                             CROSS JOIN
                             SELECT DISTINCT sid FROM Reserves
                            ) AS r1                                     // r1(x,y) = S(y) x R(x)
                        )
                        EXCEPT
                        (SELECT sid , bid FROM Reserves)
                      ) AS r2                                           // r2=r1 – R
  );                                                                    //  R / S
```

Find the names of sailors who've reserved all boats

$$\rho\ (Tempsids, (\pi_{sid,bid} \mathrm{Reserves}) \,/\, (\pi_{bid}\ Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

```
SELECT * FROM Reserves AS r WHERE
    NOT EXISTS
    ( (SELECT b.bid FROM Boats AS b )
          EXCEPT
      (SELECT t.bid FROM Reserves AS t WHERE t.bid = r.bid )
    );
```

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

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

  - 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

$\mathcal{G}_{\mathbf{sum(c)}}\,(\mathrm{r})$

sum-C
27

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

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

branch-name	*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

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