Relational Algebra

Relational Query Languages

- Languages for describing queries on a relational database
- Allow manipulation and retrieval of data from a database.
- 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.

Remark: There are new developments (e.g. SQL3) with the goal: SQL=PL

Formal Relational Query Languages

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

¶ *Relational Algebra*: More operational, very useful for representing execution plans.

· *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

Why is Relational Algebra Important?

❖ As a theoretical foundation of the relational data model and query languages.

❖ It introduces a terminology that is important to talk about relational databases (e.g. join,…)

❖ As a language to specify plans that implement SQL queries (→query optimization; implemetation of relational DBMS)

❖ Some people believe that knowing relational algebra makes it easy to write correct SQL queries.

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*

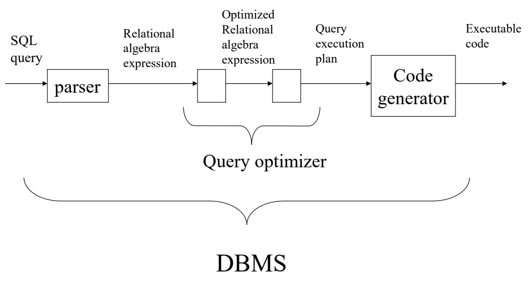
5

Relational Algebra

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, Cartesian product
- *Derived operators*: set intersection, division, join
- *Procedural*: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression

6

Relational Algebra in a DBMS



7

Relational Algebra Operators/Operations

- 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 relation 1, but not in relation 2
  - *Union* (;) Tup~~le~~s in relation 1 or in relation 2 or in both
- Additional operations:
  - Intersection, *join (natural join, theta join, equi-join, outer join)*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*!
- Relational Algebra is “closed”. The operators take one or more relations as inputs and give a new relation as a result.

Select Operation

- Notation: $\sigma_p(r)$
- p is called the selection predicate
- Defined as:

  $$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

  Where p is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
  Each term is one of:

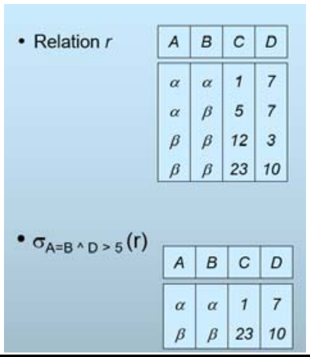
  <attribute> *op* <attribute> or <constant>

  where *op* is one of: $=, \neq, >, \geq. <. \leq$
- Example of selection:

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

  $\sigma_{\text{Salary} > 40000}\ (\textit{Employee})$

Select Operation Example



- Relation *r*

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

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Employee

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary} > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

Project Operation

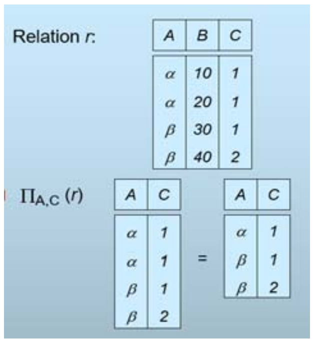
- Notation:

  $$\Pi_{A1, A2, \ldots, Ak}\ (r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- E.g. To eliminate the *branch-name* attribute of *account*

  $$\Pi_{\textit{account-number, balance}}\ (\textit{account})$$

Project Operation Example


Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)

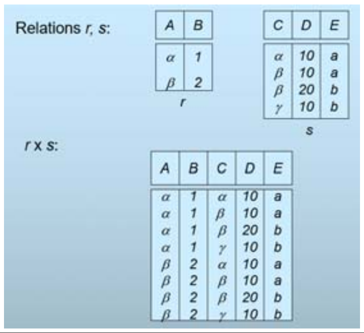
Name	Salary
John	20000
John	60000

Cartesian-Product Operation

- Notation *r* x *s*
- Defined as:

  r x *s* = {*t q* | *t* ∈ *r* **and** *q* ∈ *s*}
- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.
- Very rare in practice; mainly used to express joins

Cartesian Product Example



Relations *r*, *s*:

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

r x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Union Operation

- Notation: $r \cup s$
- Defined as:

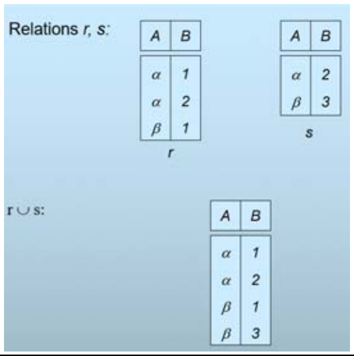
  $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- For $r \cup s$ to be valid we need union compatibility.

  1. *r, s* must have the *same arity* (same number of attributes)

  2. The attribute domains must be *compatible*
- E.g. to find all customers with either an account or a loan

  $\Pi_{customer\text{-}name}\ (depositor)\ \cup \Pi_{customer\text{-}name}\ (borrower)$

Union Example



Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

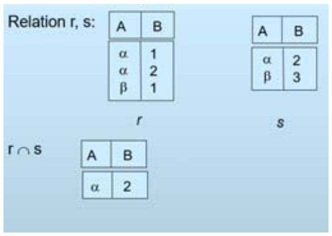
r ∪ s:

A	B
α	1
α	2
β	1
β	3

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Intersection Example



Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∩ s

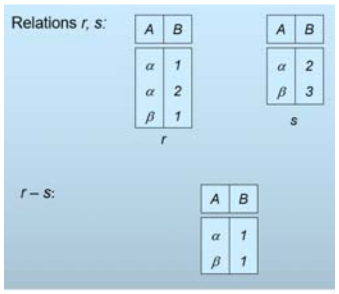
A	B
α	2

Set Difference Operation

- Notation $r - s$
- Defined as:

  $r - s = \{t \mid t \in r$ **and** $t \notin s\}$
- Set differences must be taken between *compatible* relations.
  - *r* and *s* must have the *same arity*
  - attribute domains of *r* and *s* must be compatible

Set Difference Example



Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r – s:

A	B
α	1
β	1

Join Operation

- A combination of a Cartesian product followed by a selection process.
- Pairs two tuples from different relations, if and only if a given join condition is satisfied.
- Can be classified as:
  - Inner join
    - Theta join
    - Equi-join
    - Natural join
  - Outer join
    - Left-outer, Right-outer, and Full-outer

Theta Join

- Theta join combines tuples from different relations provided they satisfy the theta condition.
- The join condition is denoted by the symbol θ

$R1 \bowtie_{\theta} R2$

where θ is a predicate using any of the six relational operators {<, <=, >, >=, =, !=}

Equi Join

- When Theta join uses only equality comparison operator, it is said to be equijoin.

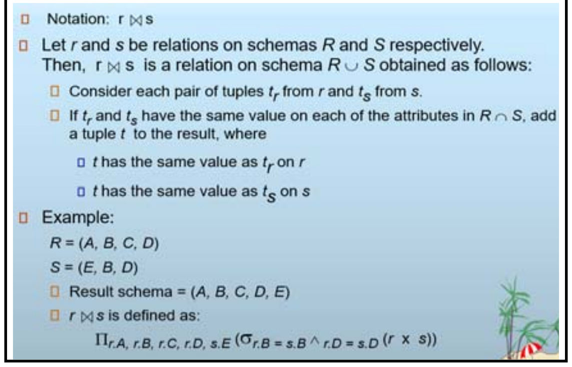
Natural Join

- A type of equi-join in which columns with the same name of associated tables will appear once only.
- Represented by * or $\bowtie$
- The natural join can be applied over two tables provides:
  - The tables have one or more pairs of identically named columns.
  - The columns must be the same data type.

Natural Join (cont…)

- R1* R2 = $\Pi_A(\sigma_C(R1 \times R2))$
- Where:
  - The selection σ_C checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

Natural-Join (Cont…)



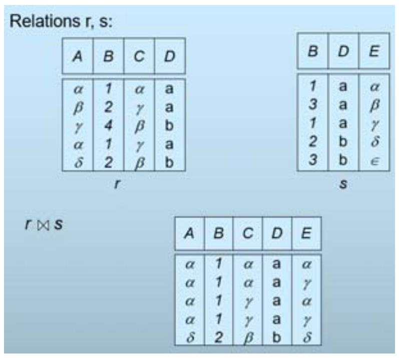
- Notation: r ⋈ s
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  R = (*A, B, C, D*)

  S = (*E, B, D*)
  - Result schema = (*A, B, C, D, E*)
  - *r* ⋈ *s* is defined as:

    $\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$

Natural Join Example-1



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

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

Natural Join Example-2

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee * Dependents

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Natural Join Example-3

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

S=

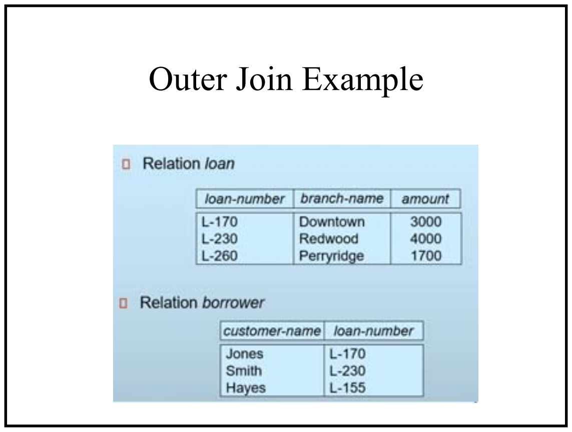
B	C
Z	U
V	W
Z	V

- R * S=

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that do 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



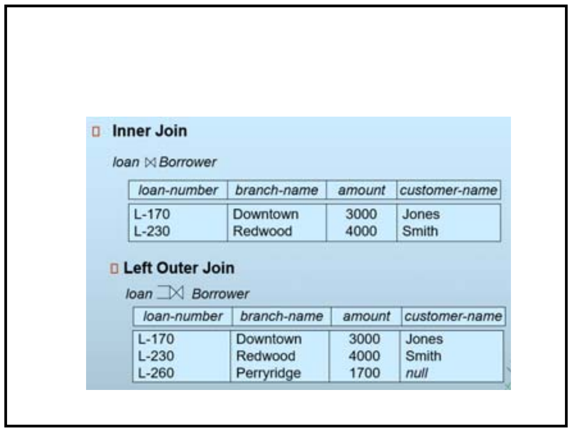
Outer Join Example

- Relation *loan*

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

- Relation *borrower*

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



- **Inner Join**

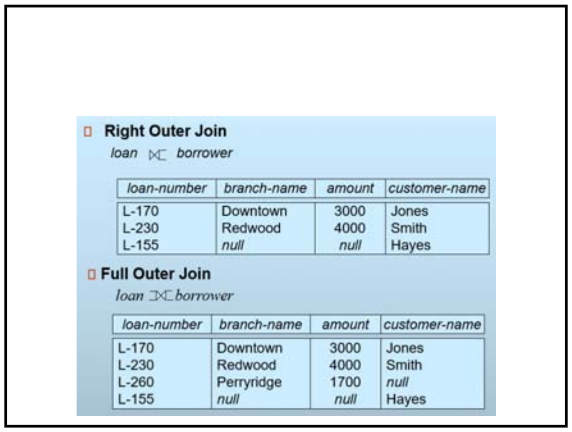
loan $\bowtie$ *Borrower*

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

- **Left Outer Join**

loan ⟕ *Borrower*

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



- **Right Outer Join**

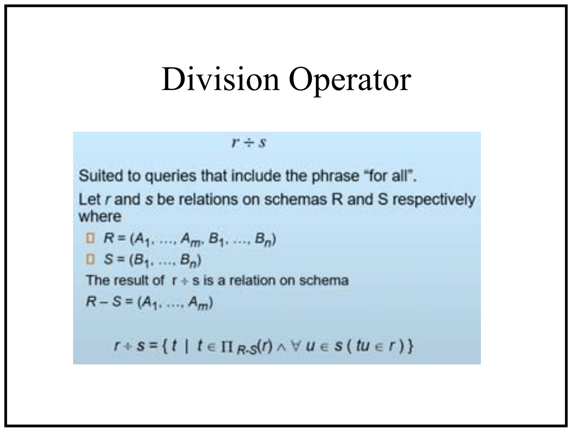
loan ⟖ *borrower*

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

- **Full Outer Join**

loan ⟗ *borrower*

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



Division Operator

$r \div s$

Suited to queries that include the phrase "for all".

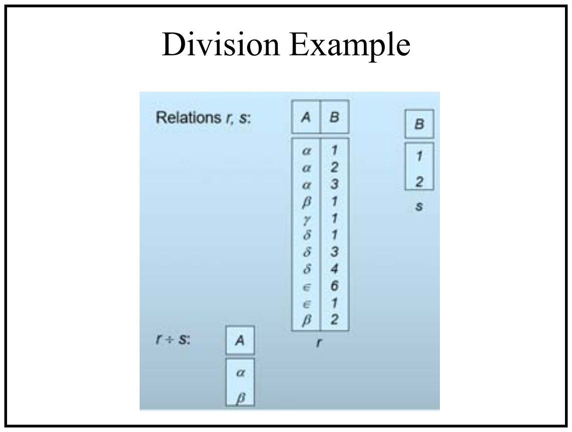
Let *r* and *s* be relations on schemas R and S respectively where

- $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
- $S = (B_1, \ldots, B_n)$

The result of $r \div s$ is a relation on schema

$R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s \, (tu \in r) \}$$



Division Example

Relations *r*, *s*:

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

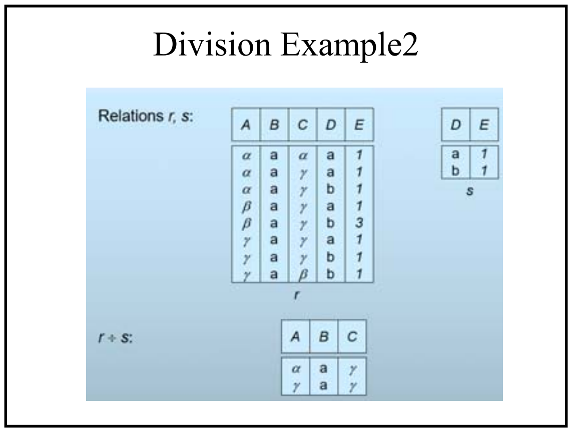
r

B
1
2

s

$r \div s$:

A
α
β



Division Example2

Relations *r*, *s*:

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

r

D	E
a	1
b	1

s

$r \div s$:

A	B	C
α	a	γ
γ	a	γ

Other Examples of Division A/B

vqr	sqr
v4	s4
v4	s5
v4	s6
v4	s7
v5	s4
v5	s5
v6	s5
v7	s5
v7	s7

D

sqr
s5

E4

sqr
s5
s7

E5

sqr
s4
s5
s7

E6

vqr
v4
v5
v6
v7

D Æ4

vqr
v4
v7

D Æ5

vqr
v4

D Æ6

Rename Operation

- Allows us to name the results of relational-algebra expressions.
- Changes the schema, not the instance
- Allows us to refer to a relation by more than one name.

Example:

$$\rho_X(E)$$

returns the expression *E* under the name *X*

If a relational-algebra expression *E* has arity *n*, then

$$\rho_{X\,(A1,\,A2,\,\ldots,\,An)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to *A1*, *A2*, …., *An*.

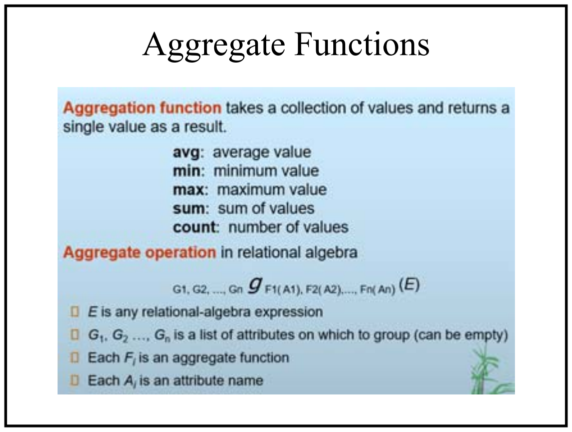
Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{LastName,\ SocSocNo}$ **(Employee)**

LastName	SocSocNo
John	999999999
Tony	777777777



Aggregate Functions

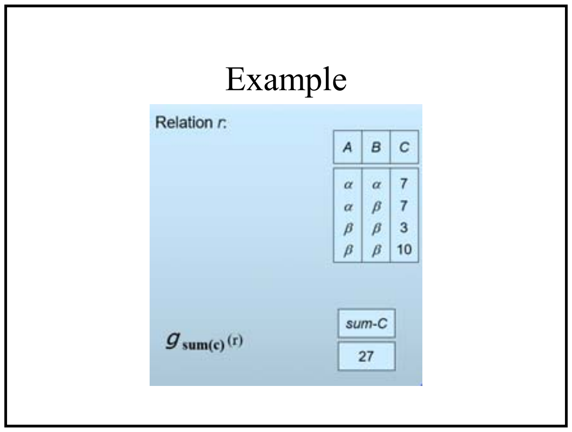
Aggregation function takes a collection of values and returns a single value as a result.

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



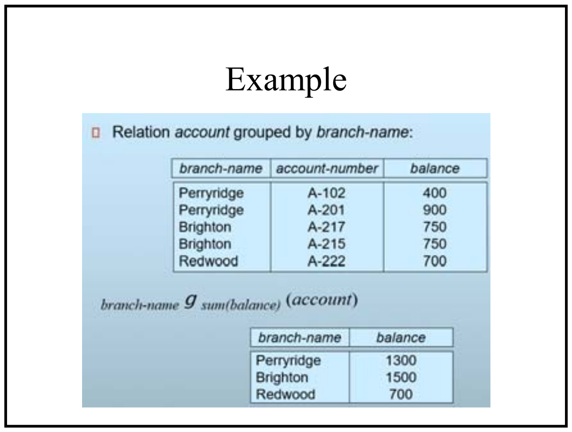
Example

Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\textbf{sum(c)}}$ (r)

sum-C
27



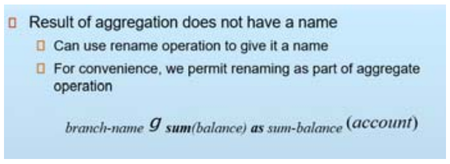
Example

- 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



- 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

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

Finally: RA has Limitations !

- Cannot compute "transitive closure"

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write C program

Practice Exercise

Consider the following database schema and write Relational Algebra expressions and SQL codes to answer Queries 1-16.

- Emp(empNo, name, gender, city, salary, depNo)
- Dept(depNo, depName, depLoc)
- Project(pNo, pName, pDep, pDuration, pCost)
- EmpProj(empNo, pNum, startDate)

Example Queries

- Q1: Retrieve empNo of all those employees who work on at least one project.

Result ← Projection $_{empNo}$ (EmpProj)

- Q2: Retrieve the names of all those employees who work on at least one project.

R1 ← Projection $_{empNo}$ (EmpProj)
R2 ← R1 * Emp
Result ← Projection $_{name}$ (R2)

- Q3: Retrieve empNo of all those employees who work on at least two projects.

R1 ← $_{empNo}$ g $_{count(empNo)\ as\ cnt}$ (EmpProj)
R2 ← Selection $_{cnt > 1}$ (R1)
Result ← Projection $_{empNo}$(R2)

- Q4: Retrieve the names of all those employees who work on at least two projects.

R1 ← $_{\text{empNo}}$ g $_{\text{count(empNo) as cnt}}$ (EmpProj)

R2 ← Selection $_{\text{cnt > 1}}$ (R1)

R3 ← R2 * Emp

Result ← Projection $_{\text{name}}$ (R2)

- Q5: Retrieve the names of all those employees who don't work on any project.

R1 ← Projection $_{\text{empNo}}$ (EmpProj)

R2 ← Projection $_{\text{ssn}}$ (Emp)

R3 ← R1 – R2

R4 ← R3 * Emp

Result ← Projection $_{\text{name}}$ (R4)

- Q6: Retrieve the names of all those employees who work on all projects on which e4 works.

R1 ← Projection $_{\text{empNo}}$ (EmpProj)

R2 ← Projection $_{\text{ssn}}$ (Emp)

R3 ← R1 – R2

R4 ← R3 * Emp

Result ← Projection $_{\text{name}}$ (R4)

- Q7: Retrieve the names of all those projects on which no employee works.
- Q8: Retrieve the names of all those projects on which more than 3 employee works.
- Q9: Retrieve the names of all those employees of the CS department who work on at least two projects.
- Q10: Retrieve the names of all those employees of the CS department who work on at most two projects.
- Q11: Retrieve the names of the departments that have at least 3 employees and execute at least one project.
- Q12: Retrieve the names of all those departments who don't execute any project.
- Q13: Retrieve the names of all those departments with two or more employees who don't execute any project.

- Q14: Retrieve the name of the department which executes maximum number of projects.
- Q15: Retrieve the name of the department which executes minimum number of projects.
- Q16: Retrieve the name of the department which executes second highest number of projects.
- Q17: Retrieve the name of the project on which maximum number of female employees work.
- Q18: Retrieve the name of the department with maximum number of female employee.
- Q19: Retrieve the name of the employees who work on all those projects on which employees of 'New Delhi' work.