RDBMS

Relational Algebra

Relational algebra

- Operands: relations (tables)
- Closure: the result of any operation is another relation
- Complete: all combinations of operators allowed
- Unary operators (single operand):
  sélection (noté σ), projection (π)
- Binary operators:
  Cartesian product ($\times$), join ($\bowtie$), union ($\cup$), intersection ($\cap$), set difference ($-$), division ($/$)

Outline

- For each of these 8 operators:
  - the operation
  - syntax (notation)
  - semantics (expected result)
  - schema
  - some annotation
  - an example

Selection σ

- Goal: only select some tuples (lines) of a relation

Country	name	capital	population	surface
	Austria	Vienna	8	83
	UK	London	56	244
	Switz.	Berne	7	41

We wish to select only countries with a small surface :

small-country = σ [surface < 100] Country

small-Country	name	capital	population	surface
	Austria	Vienna	8	83
	~~UK~~	~~London~~	~~56~~	~~244~~
	Switz.	Berne	7	41

Projection π

- Goal: only keep some attributes (columns) of a relation

Country	name	capital	population	surface
	Austria	Vienna	8	83
	UK	London	56	244
	Switz.	Berne	7	41

We only want to keep name and capital attributes :

capitals = π [name, capital] Country

capitals	name	capital	~~population~~	~~surface~~
	Austria	Vienna	~~8~~	~~83~~
	UK	London	~~56~~	~~244~~
	Switz.	Berne	~~7~~	~~41~~

Side-effect of projection

- Elimination of repeated tuples
  - A projection that does not preserve the primary key of a relation may produce identical tuples in its result
  - The result will only contain one instance of the tuple
  - In SQL, this is not the default behavior, use DISTINCT keyword to force this behavior

R (B , C, D) π (B , C) R

three tuples two tuples

Selection-projection

- We want the capitals of smalls Country:
  - **small-Country = σ [surface < 100] Country**
  - **capitals = π [name, capital] small-Country**

capital-small-Country =

π [name, capital] σ [surface < 100] Country

name	capital	population	surface
Ireland	Dublin	3	70
Austria	Vienna	8	83
UK	London	56	244
Switz.	Berne	7	41

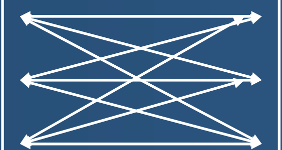
(grey and beige parts eliminated)

Cartesian product $\times$

- Goal: construct all combinations of tuples of two relations (usually before a selection)
- syntax : $R \times S$
- example :

R

A	B
a	b
b	c
c	b



S

C	D	E
c	d	e
b	a	b
a	a	c

$R \times S$

A	B	C	D	E
a	b	c	d	e
a	b	b	a	b
a	b	a	a	c
b	c	c	d	e
b	c	b	a	b
b	c	a	a	c
c	b	c	d	e
c	b	b	a	b
c	b	a	a	c

n tuples

m tuples

n x m tuples

Natural join $\bowtie$

- Goal: create all **significative** combinations of the tuples of two relations
  - significative = bear the same value for the attribute on which the join is performed
- precondition: the two relations have an attribute of a the same type
- example :

R

A	B
a	b
b	c
c	b

S

B	C	D
b	c	d
a	a	b
d	a	c

R $\bowtie$ S

A	B	C	D
a	b	c	d
c	b	c	d

R.B = S.B

Union ∪

- binary operator
- syntax : $R \cup S$
- semantics : adds into a single relation the tuples (lines) of R and S
- schema : schema($R \cup S$) = schema(R) = schema(S)
- precondition : schema(R) = schema(S)
- example :

R1

A	B
a	b
b	b
y	z

R2

A	B
u	v
y	z

R1 ∪ R2

A	B
a	b
b	b
y	z
u	v

Intersection $\cap$

- binary operator
- syntax : R $\cap$ S
- semantics : selects tuples that belong to both R and S
- schema : schema (R $\cap$ S) = schema (R) = schema (S)
- precondition : schema (R) = schema (S)
- example :

R1

A	B
a	b
y	z
b	b

R2

A	B
u	v
y	z

R1 $\cap$ R2

A	B
y	z

Set Difference -

- binary operator
- syntax : R – S
- semantics : selects tuples of R that are not in S
- schema : schema (R – S) = schema (R) = schema (S)
- precondition : schema (R) = schema (S)
- example :

R1

A	B
a	b
y	z
b	b

R2

A	B
u	v
y	z

R1 – R2

A	B
a	b
b	b

Division /

- Goal: treat requests of the type «the ... such that ALL the...»
- let R(A1, ..., An) and V(A1, ..., Am)
  with n>m and A1, ..., Am attributes of the same name in R and V
- R/V = { <am+1, am+2, ..., an> / $\forall$<a1, a2, ..., am> $\in$V,
  $\exists$<a1, a2, ..., am ,am+1, am+2, ..., an> $\in$R}
- examples :

R

A	B	C
1	1	1
1	2	0
1	2	1
1	3	0
2	1	1
2	3	3
3	1	1
3	2	0
3	2	1

V

B	C
1	1
2	0

R/V

A
1
3

V'

B	C
1	1

R/V'

A
1
2
3

V''

B	C
3	5

R/V''

A
/

example division

- R V R/V

STUDENT	COURSE	PASSED
Francois	RDB	yes
Francois	Prog	yes
Jacques	RDB	yes
Jacques	Math	yes
Pierre	Prog	yes
Pierre	RDB	no

COURSE	PASSED
Prog	yes
RDB	yes

STUDENT
Francois

Division

certifications	PILOTE	APPAREIL
	Sierra	737
	Sierra	757
	Sierra	747
	Delta	320
	Delta	757
	Alpha	737
	Alpha	757
	Alpha	747
	Alpha	320
	India	737

avions	APPAREIL
	320

certificationsA = certifications ÷ avions

certificationsA	PILOTE
	Delta
	Alpha

Division

certifications	PILOTE	APPAREIL
	Sierra	737
	Sierra	757
	Sierra	747
	Delta	320
	Delta	757
	Alpha	737
	Alpha	757
	Alpha	747
	Alpha	320
	India	737

avions	APPAREIL
	737
	757
	747

certificationsA = certifications ÷ avions

certificationsA	PILOTE
	Sierra
	Alpha

Examples of algebraic requests

- let us consider the following relations :

Journal (code-j, title, price, type, periodicity)

Depot (no-Depot, name-Depot, adress)

Delivery (no-Depot, code-j, date-deliv, quantity-delivered)

Satisfy these requests :

- What is the price of the journals ?

  π [price] Journal

- Give all known information on weekly journals.

  σ [periodicity = "weekly"] Journal

- Give the codes of the journals delivered in Paris.

  π [code-j] (σ [adress = "Paris"] Depot $\bowtie$ Delivery)

Satisfy these requests :

- Give the number of the depots that receive several journals.

π [no-Depot]
　(σ [code-j $\neq$ code']
　　(π [no-Depot, code'] α [code-j, code'] Delivery)
　　　$\bowtie$ π [no-Depot, code-j] Delivery)

- Note : α [code-j, code'] renames attribute code-j into code'
- Algebraic trees allow to reason on request evaluation order and request optimization

Give the number of the depots that receive several journals :