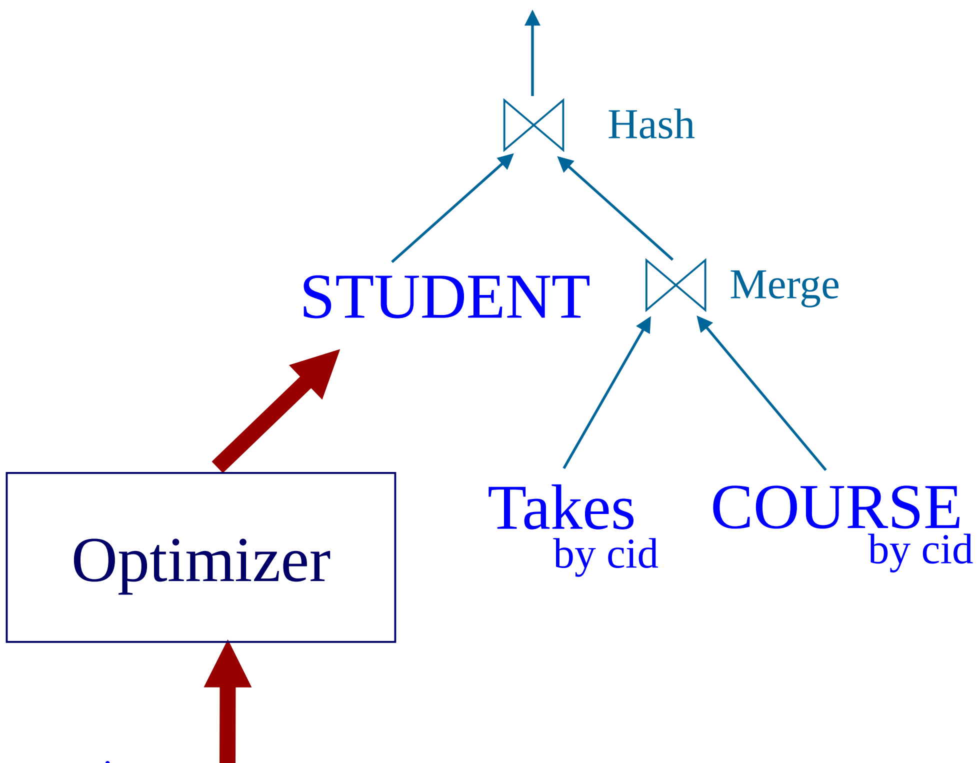
Relational Algebra & Calculus

Roberto Marabini

EDAT

2017

Query Plan



SELECT *
FROM STUDENT, Takes, COURSE
WHERE STUDENT.sid = Takes.sID
AND Takes.cID = cid;

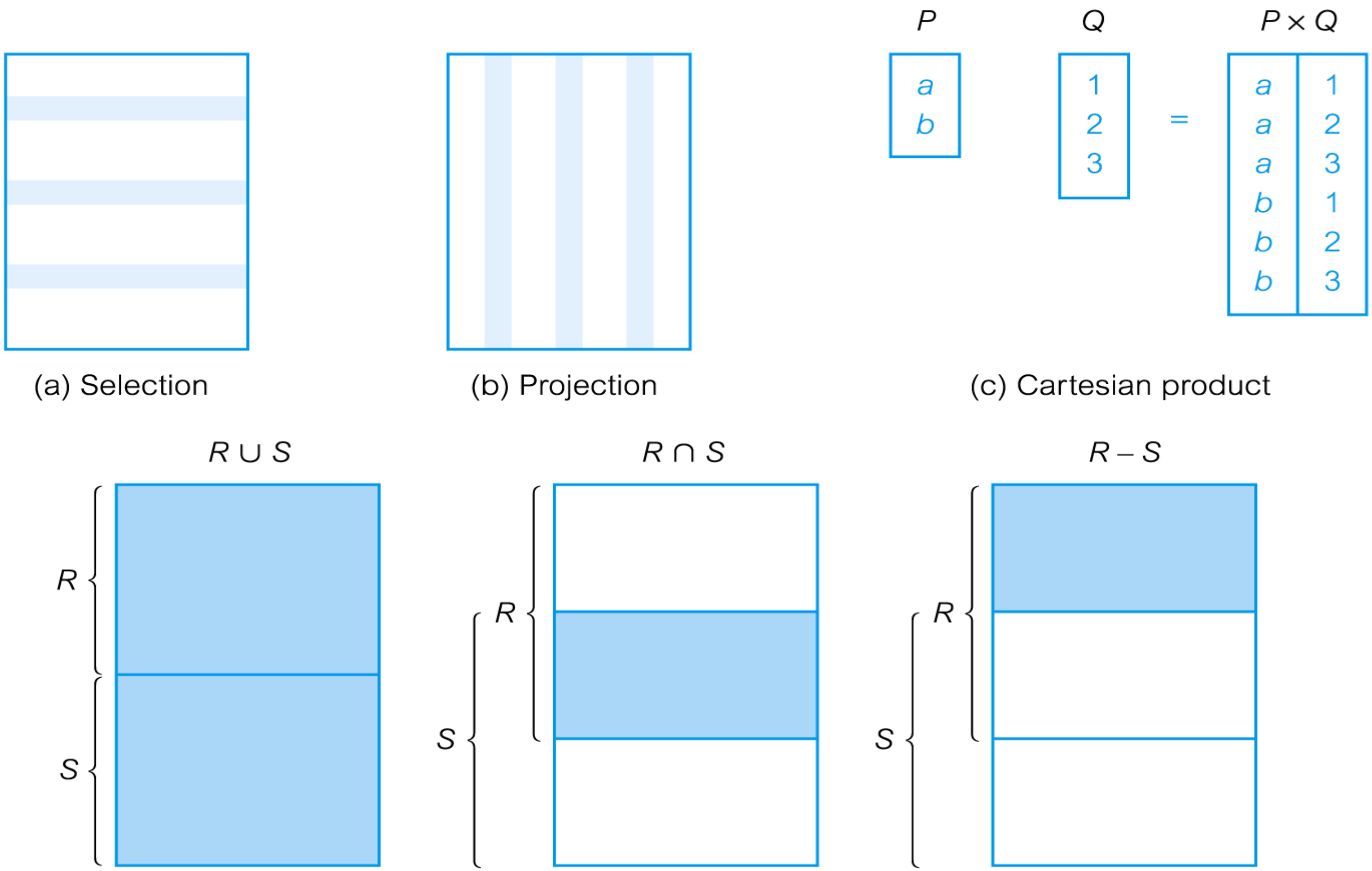
Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - *Relational Algebra*: More operational (procedural), very useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it: Non-operational, *declarative*.

Relational Algebra

- 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 operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
  - Aggregation (sum, avg, etc.)
- Since each operation returns a relation, *operations can be composed*: algebra is "closed".

Relational Algebra Operations



(a) Selection (b) Projection (c) Cartesian product

(d) Union (e) Intersection (f) Set difference

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 input relation.
- Projection operator has to eliminate *duplicates*! Why?
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it (by DISTINCT). Why not?

Relation *r*

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

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

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! Why?
- *Schema* of result identical to schema of input relation.
- What is Operator composition?
- Selection is commutative

Relation *r*

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

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

A	B	C	D
α	α	1	7
β	β	23	10

Union, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - \`Corresponding' fields have the same type.
- What is the *schema* of result?

Union – Example

Relations *r, s:*

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

Difference. Example

Relaciones *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

Cross-Product (Cartesian Product)

- Each row of S1 is paired with each row of R1.

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

Renaming operator

Name the result of an operation

Refer to the same relation by several names

Example:

$$\rho_X(E)$$

Assigns the result of Expression E to name X

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

Renames Attributes as: $A1, A2, \ldots, An$.

Notation: $\leftarrow$

A Set of Logical Operations: The Relational Algebra

- Six basic operations:
  - Projection $\pi_{\overline{\alpha}}$ (R)
  - Selection σ_{θ} (R)
  - Union $R_1 \cup R_2$
  - Difference $R_1 - R_2$
  - Product R_1 X R_2
  - Rename $\rho_{\overline{\alpha} \to \overline{\beta}}$ (R)
- And some other useful ones:
  - Join $R_1 \bowtie_{\theta} R_2$
  - Intersection $R_1 \cap R_2$
  - Division R_1 / R_2

Natural Join

Relation 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	$\in$

s

$r \bowtie s$

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

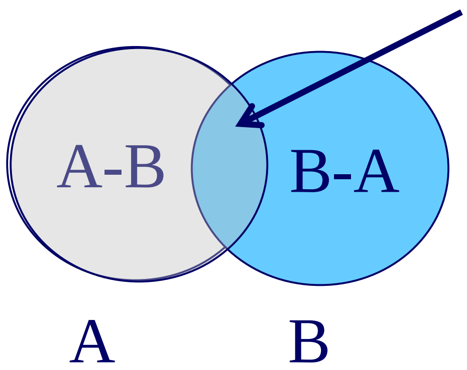
$R >\!<_c S$

Properties of join

-
- Is join commutative? $S1 \bowtie R1 = R1 \bowtie S1$?
- Is join associative? $S1 \bowtie (R1 \bowtie C1) = (S1 \bowtie R1) \bowtie C1$?

Deriving Intersection

Intersection: as with set operations, derivable from difference



$$
\begin{aligned}
A \cap B \\
&\equiv (A \cup B) - (A - B) - (B - A) \\
&\equiv A - (A - B)
\end{aligned}
$$

Division

- 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*/*B* = $\left\{\langle x\rangle \mid \exists \langle x, y\rangle \in A \ \forall \langle y\rangle \in B\right\}$
  - i.e., ***A*/*B* contains all *x* tuples (sailors) such that for *every* *y* tuple (boat) in *B*, there is an *xy* tuple in *A*.**
  - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, the *x* value is in *A*/*B*.
- 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

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

Mini-Quiz

- This completes the basic operations of the relational algebra. Try writing queries for these:
  - The IDs of students named "Bob"
  - The names of students expecting an "A"
  - The names of students in 501-0105 class
  - The sids and names of students not enrolled

Data Instance for Operator Examples

STUDENT

sid	name
1	Jill
2	Qun
3	Nitin

Takes

sid	exp-grade	cid
1	A	550-0105
1	A	700-1005
3	C	501-0105

COURSE

cid	subj	sem
550-0105	DB	F05
700-1005	AI	S05
501-0105	Arch	F05

PROFESSOR

fid	name
1	Ives
2	Saul
8	Roth

Teaches

fid	cid
1	550-0105
2	700-1005
8	501-0105

Even More Operators (Extended Relational Algebra)

Generalized Projection

Aggregation Function

Generalized Projection

Extends the projection operation by allowing arithmetic functions to be used in projection list.

$$\prod_{F1,\ F2,\ \ldots,\ Fn}(E)$$

E is a relation.

F_1, F_2, …, F_n *are arithmetic functions that use constant and attributes from E.*

Aggregation Functions

Input: set of values. Output: single value

- **avg**: average value
- **min**: minimum value
- **max**: maximum value
- **sum**: sum
- **count**: number of values

Notation:

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

E: relational algebra expression

G_1, G_2 …, G_n list of attributes used for grouping.

F_i aggregation functions

A_i : attributes

Aggregation operator: Example 1

Relación *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

Aggregation operator: Example II

Account

branchName	*Account No*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{branchName}\mathcal{g}_{\ sum(balance)}\ (account)$$

branchName	*XXXX*
Perryridge	1300
Brighton	1500
Redwood	700

Modify DataBases: insert, update, delete

- $r \leftarrow r \cup E$ *(insert)*
- $r \leftarrow r - E$ *(delete)*
- *Update: sequence of insert, delete operations.*

Example Queries (Find PK)

Assume the following relations:

BOOKS(DocId, Title, Publisher, Year)

STUDENTS(StId, StName, Major, Age)

AUTHORS(AName, Address)

borrows(DocId, StId, Date)

has-written(DocId, AName)

describes(DocId, Keyword)

Example Queries

Assume the following relations:

BOOKS(DocId, Title, Publisher, Year)

STUDENTS(StId, StName, Major, Age)

AUTHORS(AName, Address)

borrows(DocId^, StId^, Date^)

has-written(DocId^, Aname^)

describes(DocId^, Keyword)

Exercises

1. List the year and title of each book
2. List all information about students whose major is CS
3. List all books published by McGraw-Hill before 1990.
4. List the name of those authors who are living in Davis.
5. List the name of students who are older than 30 and who are not studying CS
6. Rename AName in the relation AUTHORS to Name

Exercises - II

1. List the names of all students who have borrowed a book and who are CS majors
2. List the title of books written by the author ’Silberschatz’.
3. As 2., but not books that have the keyword ’database’
4. Find the name of the youngest student
5. Find the title of the oldest book

Switching Gears: An Equivalent, But Very Different, Formalism

- Codd invented a relational calculus that he proved was equivalent in expressiveness
  - More convenient for describing certain things, and for certain kinds of manipulations
- The database uses the relational algebra internally
- Relational calculus query specifies what is to be retrieved rather than how to retrieve it.
- Interested in finding tuples for which a predicate is true.
- To find set of all tuples S such that P(S) is true: {S | P(S)}

Tuple Relational Calculus - Example

- To find details of all staff earning more than 10,000:
  {S | Staff(S) ^ S.salary > 10000}

- To find a particular attribute, such as salary, write:
  {S.salary | Staff(S) ^ S.salary > 10000}

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24- Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Tuple Relational Calculus

1. Can use two quantifiers to tell how many instances the predicate applies to:
   1. Existential quantifier ∃ ('there exists')
   2. Universal quantifier ∀ ('for all')

2. Tuple variables qualified by ∀ or ∃ are called bound variables, otherwise called free variables.

Tuple Relational Calculus

- Existential quantifier used in formulae that must be true for at least one instance, such as:

Staff(S) ^ (∃ B)(Branch(B) ^

(B.branchNo = S.branchNo) ^

B.city = 'London')

- Means 'There exists a Branch tuple with same branchNo as the branchNo of the current Staff tuple, S, and is located in London'.

staffNo	fName	lName	position	sex	DOB	salary	branchNo

Branch

branchNo	street	city	postcode

Tuple Relational Calculus

- Universal quantifier is used in statements about every instance, such as:

$(\forall B)\ (B.city \neq \text{'Paris'})$

Means 'For all Branch tuples, the address is not in Paris'.

- Can also use $\sim(\exists B)\ (B.city = \text{'Paris'})$ which means 'There are no branches with an address in Paris'.

staffNo	fName	lName	position	sex	DOB	salary	branchNo

Branch

branchNo	street	city	postcode

Example - Tuple Relational Cal

staffNo	fName	lName	position	sex	DOB	salary	branchNo

- List the names of all managers who earn more than £25,000.

{S.fName, S.lName | Staff(S) ^
 S.position = 'Manager' ^ S.salary > 25000}

- List the staff who manage properties for rent in Glasgow.

{S | Staff(S) ^ (∃ P) (PropertyForRent(P) ^
 (P.staffNo = S.staffNo) ^
 P.city = 'Glasgow')}

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo

Example - Tuple Relational Calculus

- List the names of staff who currently do not manage any properties.

{S.fName, S.lName | Staff(S) ^ (~(∃ P) (PropertyForRent(P)^(S.staffNo = P.staffNo)))}

Or

{S.fName, S.lName | Staff(S) ^ ((∀ P) (~PropertyForRent(P) v ~(S.staffNo = P.staffNo)))}

staffNo	fName	lName	position	sex	DOB	salary	branchNo

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo

Example - Tuple Relational Calculus

List the names of clients who have viewed a property for rent in Glasgow.

{C.fName, C.lName | Client(C) ^ ((∃ V)(∃ P)
(Viewing(V) ^ PropertyForRent(P) ^
(C.clientNo = V.clientNo)^
(V.propertyNo=P.propertyNo) ^
P.city ='Glasgow'))}

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo

Client

clientNo	fName	lName	telNo	prefType	maxRent

Viewing

clientNo	propertyNo	viewDate	comment

Tuple Relational Calculus

- Expressions can generate an infinite set. For example: {S | ~Staff(S)}

- To avoid this, add restriction that all values in result must be values in the domain of the expression.

Domain Relational Calculus

- Uses variables that take values from domains instead of tuples of relations.

- If F(d1, d2, . . . , dn) stands for a formula composed of atoms and d1, d2, . . . , dn represent domain variables, then:

  {d1, d2, . . . , dn | F(d1, d2, . . . , dn)}

is a general domain relational calculus expression.

Example - Domain Relational Calculus

- Find the names of all managers who earn more than £25,000.

$$\{fN, lN \mid (\exists sN, posn, sex, DOB, sal, bN)$$
$$(Staff\,(sN, fN, lN, posn, sex, DOB, sal, bN) \wedge$$
$$posn = \text{'Manager'} \wedge sal > 25000)\}$$