Relational Algebra SPJRUD

Jef Wijsen

Université de Mons (UMONS)

October 1, 2024

Tabular Representation

The table

A	B	C
1	3	2
1	4	1
2	4	2
2	3	1

is shorthand for the following set of tuples:

$$\left\{\begin{array}{l}\{A:1, B:3, C:2\},\\ \{A:1, B:4, C:1\},\\ \{A:2, B:4, C:2\},\\ \{A:2, B:3, C:1\}\end{array}\right\}.$$

Notation

- Every tuple is a total function from a set of attributes to the set of constants.
- Therefore, if $t = \{A : 1, B : 3, C : 2\}$,
  then $t(B) = 3$ and $t[\{A, C\}] = \{A : 1, C : 2\}$.
- Note that $t(B)$ is a constant, and $t[\{A, C\}]$ a tuple.
- In database theory, we often omit curly brackets ($\{\}$) and union symbols ($\cup$) in the notation of sets. For example, if A, B, C, D are attributes and $X = \{A, B\}$, then XCD denotes the set $\{A, B, C, D\}$.

Algebraic Operators

- Unary operators: Select, Project, Rename
- Binary operators: Join, Union, Difference
- Unary operators take in a single relation; binary operators take in two relations.
- Every operator returns a single relation.

Select

R	A	B	C
	1	3	2
	1	4	1
	2	4	2
	2	3	1

$\sigma_{A=\text{“}1\text{”}}(R)$	A	B	C
	1	3	2
	1	4	1

$\sigma_{A=C}(R)$	A	B	C
	1	4	1
	2	4	2

- This is like `SELECT * FROM R WHERE A="1"` in SQL.

Project

R	A	B	C
	1	3	1
	1	3	2
	1	4	3
	1	4	4
	2	3	5

$\pi_{\{A,B\}}(R)$	A	B
	1	3
	1	4
	2	3

- Since relations are sets, duplicates are removed.
- Note that `SELECT A, B FROM R` in SQL does not remove duplicates.

Join

R	A	B	C
	1	3	5
	1	4	5
	2	4	5
	2	3	6

S	B	C	D
	3	5	2
	4	5	2
	4	5	1
	4	6	1

$R \bowtie S$	A	B	C	D
	1	3	5	2
	1	4	5	2
	1	4	5	1
	2	4	5	2
	2	4	5	1

- $R \bowtie S$ contains all tuples t such that $t[ABC]$ is in R, and $t[BCD]$ in S.
- This is different from the cross product `SELECT * FROM R, S` in SQL.

Rename

R	A	B	C
	1	3	2
	1	4	1
	2	4	2
	2	3	1

$\rho_{C \mapsto D}(R)$	A	B	D
	1	3	2
	1	4	1
	2	4	2
	2	3	1

Union

R	A	B	C
	1	3	5
	1	4	5

S	A	B	C
	1	4	5
	2	3	6

$R \cup S$	A	B	C
	1	3	5
	1	4	5
	2	3	6

- $R \cup S$ is only allowed if R and S have exactly the same attributes.
- In SQL, `(SELECT * FROM R) UNION (SELECT * FROM S)` only requires that R and S have the same number of attributes. The result takes the attributes of R.

Difference

R	A	B	C
	1	3	5
	1	4	5

S	A	B	C
	1	4	5
	2	3	6

$R - S$	A	B	C
	1	3	5

- $R - S$ is only allowed if R and S have exactly the same attributes.
- In SQL, `(SELECT * FROM R) MINUS (SELECT * FROM S)` only requires that R and S have the same number of attributes. The result takes the attributes of R.

Examples

VINS

Cru	*Millesime*	*Qualite*
Chablis	1992	excellent
Chablis	1993	bon
Rothschild	1993	bon
Rothschild	1994	imbuvable

ABUS

Nom	*Cru*	*Annee*
Ed	Chablis	1992
Ed	Chablis	1993
An	Chablis	1993
An	Rothschild	1993

Qui n'a jamais bu un vin excellent ?

Soit

$$\begin{aligned} E_1 &= \rho_{Annee \mapsto Millesime}(ABUS) \\ E_2 &= \sigma_{Qualite = \text{“excellent”}}(VINS) \\ E_3 &= \pi_{\{Nom\}}(E_1 \bowtie E_2) \end{aligned}$$

E_3 renvoie les personnes ayant déjà bu un vin excellent. La requête demandée est donc :

$$\pi_{\{Nom\}}(ABUS) - E_3$$

Quels crus ont varié en qualité?

Soit

$$\begin{aligned} E_1 &= \pi_{\{Cru, Qualite\}}(VINS) \\ E_2 &= E_1 \bowtie \rho_{Qualite \mapsto Valeur}(E_1) \end{aligned}$$

E_2 renvoie $\{Cru : c, Qualite : q, Valeur : v\}$ ssi c est un cru qui a été de qualité q et de qualité v (il est possible que $q = v$). Soit

$$E_3 = E_2 - \sigma_{Qualite=Valeur}(E_2)$$

E_3 renvoie $\{Cru : c, Qualite : q, Valeur : v\}$ ssi c est un cru qui a été de qualité q et de qualité v avec $q \neq v$. La requête demandée est donc :

$$\pi_{\{Cru\}}(E_3)$$

Qui a bu tous les crus?

Soit

$$\begin{aligned} E_1 &= \pi_{\{Nom\}}(ABUS) \bowtie \pi_{\{Cru\}}(VINS) \\ E_2 &= \pi_{\{Nom,Cru\}}(ABUS) \\ E_3 &= E_1 - E_2 \end{aligned}$$

E_3 renvoie $\{Nom : n, Cru : c\}$ ssi c est un cru qui n'a pas été bu par la personne n. La requête demandée est donc :

$$\pi_{\{Nom\}}(ABUS) - \pi_{\{Nom\}}(E_3)$$

Alphabet

- Relation names. Each relation name R is associated with a fixed set of attributes, denoted $sort(R)$.
- Constants.
- Typically, R, S, T are relation names; A, B, C are attributes; a, b, c are constants.

Syntax I

Relation names Every relation name is an algebra expression.

Selection If E is an algebra expression, $A \in sort(E)$, and c is a constant, then $\sigma_{A=c}(E)$ is an algebra expression with $sort(\sigma_{A=c}(E)) = sort(E)$.

Selection If E is an algebra expression and $A, B \in sort(E)$, then $\sigma_{A=B}(E)$ is an algebra expression with $sort(\sigma_{A=B}(E)) = sort(E)$.

Projection If E is an algebra expression and $X \subseteq sort(E)$, then $\pi_X(E)$ is an algebra expression with $sort(\pi_X(E)) = X$.

Join If E_1 and E_2 are algebra expressions, then $E_1 \bowtie E_2$ is an algebra expression with $sort(E_1 \bowtie E_2) = sort(E_1) \cup sort(E_2)$.

Rename If E is an algebra expression, $A \in sort(E)$, and B is an attribute not in $sort(E)$, then $\rho_{A \mapsto B}(E)$ is an algebra expression with $sort(\rho_{A \mapsto B}(E)) = (sort(E) \setminus \{A\}) \cup \{B\}$.

Syntax II

Union If E_1 and E_2 are algebra expressions with $sort(E_1) = sort(E_2)$, then $E_1 \cup E_2$ is an algebra expression with $sort(E_1 \cup E_2) = sort(E_1)$.

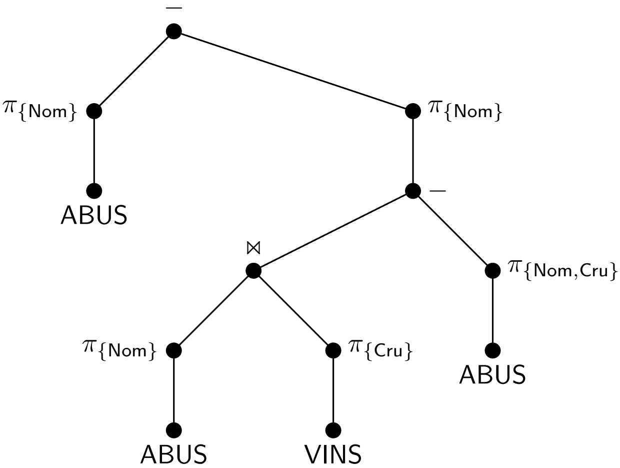
Difference If E_1 and E_2 are algebra expressions with $sort(E_1) = sort(E_2)$, then $E_1 - E_2$ is an algebra expression with $sort(E_1 - E_2) = sort(E_1)$.

Semantics I

A database [instance] $\mathcal{I}$ is a total function that maps every relation name R to a relation over $sort(R)$. We denote by $R^{\mathcal{I}}$ the relation to which R is mapped by $\mathcal{I}$.

- For every *relation name* R, $[\![R]\!]^{\mathcal{I}} = R^{\mathcal{I}}$.
- $[\![\sigma_{A=c}(E)]\!]^{\mathcal{I}} = \{t \in [\![E]\!]^{\mathcal{I}} \mid t(A) = c\}$.
- $[\![\sigma_{A=B}(E)]\!]^{\mathcal{I}} = \{t \in [\![E]\!]^{\mathcal{I}} \mid t(A) = t(B)\}$.
- $[\![\pi_X(E)]\!]^{\mathcal{I}} = \{t[X] \mid t \in [\![E]\!]^{\mathcal{I}}\}$.
- Assume $sort(E_1) = X$ and $sort(E_2) = Y$. Then, $[\![E_1 \bowtie E_2]\!]^{\mathcal{I}} = \{t \mid t[X] \in [\![E_1]\!]^{\mathcal{I}} \text{ and } t[Y] \in [\![E_2]\!]^{\mathcal{I}}\}$.
- $[\![\rho_{A \mapsto B}(E)]\!]^{\mathcal{I}}$ contains every tuple that can be obtained by replacing A with B in some tuple of $[\![E]\!]^{\mathcal{I}}$.
- $[\![E_1 \cup E_2]\!]^{\mathcal{I}} = [\![E_1]\!]^{\mathcal{I}} \cup [\![E_2]\!]^{\mathcal{I}}$, where $\cup$ is the standard union of sets.
- $[\![E_1 - E_2]\!]^{\mathcal{I}} = [\![E_1]\!]^{\mathcal{I}} \setminus [\![E_2]\!]^{\mathcal{I}}$.

Syntax Tree for *"Qui a bu tous les crus?"*



Graphical Interpretation

- The syntax defines which trees are correct.
- The leaf nodes of the tree are always labeled with relation names.
- The semantics essentially tells us that a tree can be evaluated bottom-up once relations (i.e., finite sets of tuples) have been associated with the leaf nodes.

Equivalence

Two algebra expressions E_1 and E_2 are said to be equivalent, denoted $E_1 \equiv E_2$, if for every database $\mathcal{I}$, we have $[\![E_1]\!]^{\mathcal{I}} = [\![E_2]\!]^{\mathcal{I}}$.

Example 1

$$\pi_{\{Cru\}}(\sigma_{Qualite=\text{“excellent”}}(VINS)) \\ \equiv \\ \pi_{\{Cru\}}(\sigma_{Qualite=\text{“excellent”}}(\pi_{\{Cru,Qualite\}}(VINS)))$$

Example 2

Let R and S be two relation names such that $A \in sort(R) = sort(S)$.

$$\pi_{\{A\}}(\sigma_{A=\text{“7”}}(R \cup S)) \equiv (\sigma_{A=\text{“7”}}(\pi_{\{A\}}(R))) \cup (\sigma_{A=\text{“7”}}(\pi_{\{A\}}(S)))$$

Theorem (Trakhtenbrot 1950)

There exists no algorithm for the following problem:

INPUT: Two expressions E_1, E_2 in SPJRUD.

QUESTION: Are E_1 and E_2 equivalent?

Boris Trakhtenbrot (1921–2016)

An Algorithm for Testing $E_1 \equiv E_2$?

Assume that $C = \{c_1, \dots, c_n\}$ are all constants occurring in E_1 or E_2, in selections of the form $\sigma_{A=\text{“}c_i\text{”}}(\cdot)$. Assume w.l.o.g. that no c_i is a natural number.

for $j = 0, 1, 2, 3, \dots$ **do**
  for each database $\mathcal{I}$ that uses only constants in $\{1, 2, \dots, j\} \cup C$ (and uses only relation names occurring in E_1 or E_2) **do**
    if $[\![E_1]\!]^{\mathcal{I}} \neq [\![E_2]\!]^{\mathcal{I}}$,
    then return $\mathcal{I}$ and **halt**

Claim:

- If $E_1 \not\equiv E_2$, then this piece of code will halt in finitely many steps and return a database on which E_1 and E_2 disagree.
- If $E_1 \equiv E_2$, then the code will run forever without halting.

It is crucial here that, by definition, databases are finite.

Two Important Sub-languages of SPJRUD

SPJR	Conjunctive queries
SPJRU	Unions of conjunctive queries

Theorem

There exists an algorithm for the following problem:

INPUT: *Two expressions E_1, E_2 in SPJRU.*

QUESTION: *Are E_1 and E_2 equivalent?*

Take-home message: "Less can be more beautiful."

Discussion

- Can we express all "interesting" queries in SPJRUD, or should we add more operators?
- Why don't we have intersection? Why don't we have $\sigma_{A \neq B} R$?
- Show the following: if we leave out one of the 6 operators, then we can express strictly less queries.

  Hint: For union, consider two relations R and S:

R: A
1

  and

S: A
0

  Show that if E is an algebraic expression that does not contain $\cup$, then E will not return

A
0
1

  on input R and S (and therefore, E does not express $R \cup S$).