The Relational Algebra

Dirk Van Gucht[1]

[1] Indiana University

February 5, 2019

Outline

- Motivation
- Syntax of Relational Algebra (RA)
- Semantics of RA in SQL
- Expressing queries in RA

Relational Algebra (Motivation)

- SQL is a declarative language to specify queries over relational databases
- RA is a procedural language wherein algebraic expressions specify queries
- RA expressions provide procedures (algorithms) to evaluate queries
  - Examining these procedure provides insights about the time and space complexities of evaluating queries

Relational Algebra (Motivation)

- SQL[1] and RA express the same queries:
  - Each SQL query can be expressed by an RA expression
  - Each RA expression can be expressed by a SQL query.
- Query optimization: Rewrite rules can transform a RA expression into another equivalent RA expression that is more efficient to evaluate
- Query evaluation: Different algorithms and data structures can be associated with RA operations for efficient evaluations

[1] without aggregation functions

Syntax of RA

- The relational algebra is a typed language of expressions
- Starting from relations and constants, RA expressions are inductively built using the operators

Operator	Algebraic notation
cartesian product	$\times$
selection	$\sigma(.)$
projection	$\pi_{\ldots}(.)$
union	$\cup$
intersection	$\cap$
difference	$-$

- Each RA expression has a schema which is its type
- We will use $E(A_1, \ldots, A_m)$ to denote a RA expression E with attribute schema $(A_1, \ldots, A_m)$

RA basic expressions: relations

- Let $R(A_1, \ldots, A_m)$ be a relation
- Then R is a RA expression with schema $(A_1, \ldots, A_m)$
- Its value is the set of tuples from R, i.e., the relation instance associated with R
- R can be expressed by SQL query

SELECT $r.A_1, \ldots, r.A_m$
FROM $R\ r$

RA basic expressions: constants

- Let A be an attribute and let **a** be a constant
- Then (A:**a**) is a RA expression with schema (A)
- Its value is a unary relation containing the single tuple (**a**)
- (A : **a**) can be expressed by SQL query

SELECT **a** AS A

A
a

RA operators: the cartesian product $\times$

- Let $R(A_1, \ldots, A_m)$ and $S(B_1, \ldots, B_n)$ be two relations with non-overlapping schemas
- Then $R \times S$ is a RA expression with schema $(A_1, \ldots, A_m, B_1, \ldots, B_n)$
- Its value is the set of all tuples that can be obtained by pairing *each* tuple r of R with *each* tuple s of S
- $R \times S$ can be expressed by the SQL query

$$\begin{array}{ll} \text{SELECT} & r.A_1, \ldots, r.A_m, s.B_1, \ldots, s.B_n \\ \text{FROM} & R\,r,\ S\,s \end{array}$$

- Observe that $|R \times S| = |R||S|$

RA operators: the cartesian product $\times$ (example)

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S

D
1
2

$R \times S$

A	B	C	D
a_1	b_1	c_1	1
a_2	b_2	c_2	1
a_3	b_3	c_3	1
a_1	b_1	c_1	2
a_2	b_2	c_2	2
a_3	b_3	c_3	2

RA operators: the cartesian product $\times$ (building search space)

- $R \times S$ provides in **single** relation all the information that is present in R and S
- $R \times S$ can be interpreted as the search space
- It provides a mechanism to associate each tuple r of R with each tuple s of S into a tuple (r, s) who components can be compared
  - I.e., the components $r.A_1, \ldots, r.A_m$ of r can be compared with the components $s.B_1, \ldots, s.B_n$ of s

RA operators: the cartesian product $\times$ (the general case)

- More generally, the cartesian product applies to two RA expressions with non-overlapping schemas $E_1(A_1, \ldots, A_m)$ and $E_2(B_1, \ldots, B_n)$
- $E_1 \times E_2$ has schema $(A_1, \ldots, A_m, B_1, \ldots, B_n)$
- Its value is the set of all tuples that can be obtained by pairing *each* tuple e_1 in the value of E_1 with *each* tuple e_2 in the value of E_2
- If Q_{E_1} and Q_{E_2} denote the SQL queries corresponding to E_1 and E_2 then $E_1 \times E_2$ can be expressed by the SQL query

$$\begin{array}{ll} \text{SELECT} & e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n \\ \text{FROM} & (Q_{E_1})\, e_1, (Q_{E_2})\, e_2 \end{array}$$

RA operators: the cartesian product $\times$ (multiple cartesian product factors)

- The cartesian product applies to at least two RA expressions with pairwise non-overlapping schemas $E_1, E_2, \ldots, E_k$
- $E_1 \times E_2 \times \cdots \times E_k$ has as schema the union of the schemas of E_1 through E_k
- Its value is the set of all tuples that can be obtained by pairing *each* tuple e_1 in the value of E_1 with *each* tuple e_2 in the value of E_2, etc, with each tuple e_k in the value of E_k.
- If $Q_{E_1}, Q_{E_2}, \ldots, Q_{E_k}$ denote the SQL queries corresponding to $E_1, E_2, \ldots, E_k$ then $E_1 \times E_2 \times \cdots \times E_k$ can be expressed by the SQL query

$$\begin{array}{ll} \text{SELECT} & e_1.*, e_2.*, \ldots e_k.* \\ \text{FROM} & (Q_{E_1})\, e_1, (Q_{E_2}) e_2, \ldots, (Q_{E_k})\, e_k \end{array}$$

RA operators: the cartesian product $\times$ (using CROSS JOIN or WITH)

- $E_1 \times E_2$ using the SQL CROSS JOIN operator:

$$\begin{array}{ll} \text{SELECT} & e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n \\ \text{FROM} & (Q_{E_1})\ e_1\ \text{CROSS JOIN}\ (Q_{E_2})\ e_2 \end{array}$$

- $E_1 \times E_2$ using the SQL WITH statement:

$$\begin{array}{ll} \text{WITH} & e_1\ \text{AS}\ (Q_{E_1}), \\ & e_2\ \text{AS}\ (Q_{E_2}) \\ \text{SELECT} & e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n \\ \text{FROM} & e_1, e_2 \end{array}$$

RA operators: the cartesian product $\times$ (special cases)

- In the special case where $E_1 = R$, $E_1 \times E_2$ is expressed in SQL by

$$\begin{array}{ll} \text{SELECT} & r.A_1, \ldots, r.A_m, e_2.B_1, \ldots, e_2.B_n \\ \text{FROM} & R\ r, (Q_{E_2})\ e_2 \end{array}$$

- In the special case where $E_1 = R$ and $E_2 = S$, $E_1 \times E_2$ is expressed in SQL by

$$\begin{array}{ll} \text{SELECT} & r.A_1, \ldots, r.A_m, s.B_1, \ldots, s.B_n \\ \text{FROM} & R\ r, S\ s \end{array}$$

RA operators: the cartesian product $\times$ (special cases)

- In the special case where $E_1 = (A : \mathbf{a})$, $E_1 \times E_2$ is expressed in SQL query by

$$\begin{array}{ll} \text{SELECT} & \mathbf{a} \text{ AS A}, e_2.B_1, \ldots, e_2.B_n \\ \text{FROM} & (Q_{E_2})\, e_2 \end{array}$$

- In the special case where $E_1 = (A : \mathbf{a})$ and $E_2 = (B : \mathbf{b})$, $E_1 \times E_2$ is expressed in SQL query by

$$\text{SELECT} \quad \mathbf{a} \text{ AS A}, \mathbf{b} \text{ AS B}$$

RA operators: selection $\sigma_{A\,\theta\,\mathbf{a}}$ (Introduction)

- Let $R(A_1, \ldots, A_m)$ be a relation
- Let **a** be a constant value
- Let θ be one of the comparison operators $=, \neq, <, \leq, >, \geq$
- Then $\sigma_{A_i\,\theta\,\mathbf{a}}(R)$ is a RA expression with schema $(A_1, \ldots, A_m)$
- Its value consists of the tuples r in R such that $r.A_i\,\theta\,\mathbf{a}$ is true
- $\sigma_{A_i\,\theta\,\mathbf{a}}(R)$ is expressed in SQL by

$$\begin{array}{ll} \text{SELECT} & r.A_1, \ldots, r.A_m \\ \text{FROM} & R\ r \\ \text{WHERE} & r.A_i\,\theta\,\mathbf{a} \end{array}$$

- Observe that $|\sigma_{A_i\,\theta\,\mathbf{a}}(R)| \leq |R|$

RA operators: selection $\sigma_{A\theta a}$ (Example)

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_2	b_1	c_2
a_3	b_3	c_2

$\sigma_{B=b_1}(R)$

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

$\sigma_{A\neq a_3}(R)$

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_2	b_1	c_2

RA operators: selection $\sigma_{A\,\theta\,B}$ (Introduction)

- Let $R(A_1, \ldots, A_m)$ be a relation
- Let θ be one of the comparison operators $=, \neq, <, \leq, >, \geq$
- Then $\sigma_{A_i\,\theta\,A_j}(R)$ is a RA expression with schema $(A_1, \ldots, A_n)$
- Its value is the set of tuples r of R such that $r.A_i\,\theta\,r.A_j$ is true
- $\sigma_{A_i\,\theta\,A_j}(R)$ is expressed in SQL by

$$\begin{array}{ll} \text{SELECT} & r.A_1, \ldots, r.A_m \\ \text{FROM} & R\ r \\ \text{WHERE} & r.A_i\,\theta\,r.A_j \end{array}$$

- Observe that $|\sigma_{A_i\,\theta\,A_j}(R)| \leq |R|$

RA operators: selection $\sigma_{A=B}$ (Example)

R

A	B	C
a_1	b_1	a_1
a_2	b_2	c_2
a_2	b_3	a_2

$\sigma_{A=C}(R)$

A	B	C
a_1	b_1	a_1
a_2	b_3	a_2

$\sigma_{A\neq C}(R)$

A	B	C
a_2	b_2	c_2

RA operators: selection σ

- A selection acts as a filter on R
- It provides a horizontal slice of R

RA operators: selection σ (the general case)

- More generally, selection applies to a RA expression $E(A_1, \ldots, A_m)$
- $\sigma_{A_i\,\theta\,\mathbf{a}}(E)$ has schema $(A_1, \ldots, A_m)$
- Its value is the set of tuples e in the value of E that satisfy the selection condition
- If Q_E denotes the SQL query corresponding to E then $\sigma_{A_i\,\theta\,\mathbf{a}}(E)$ is expressed in SQL by[2]

$$\begin{array}{ll} \text{SELECT} & e.A_1, \ldots, e.A_m \\ \text{FROM} & (Q_E)\ e \\ \text{WHERE} & e.A_i\,\theta\,\mathbf{a} \end{array}$$

- Analogously for $\sigma_{A_i\,\theta\,A_j}(E)$, but instead of $e.A_i\,\theta\,\mathbf{a}$ we use $e.A_i\,\theta\,e.A_j$

[2] If E is a relation R, we can replace (Q_E) by R

RA operators: selection σ (the general case)

- Alternatively, assume Q_E has the form

  SELECT ...
  FROM ...
  WHERE condition

  Then $\sigma_{A_i\,\theta\,\mathbf{a}}(E)$ is expressed in SQL by

  SELECT ...
  FROM ...
  WHERE condition AND $A_i\,\theta\,\mathbf{a}$

- For $\sigma_{A_i\,\theta\,A_j}(E)$ is expressed in SQL by

  SELECT ...
  FROM ...
  WHERE condition AND $A_i\,\theta\,A_j$

RA operators: projection π (Introduction)

- Let $R(A_1, \ldots, A_m)$ be a relation and let $(B_1, \ldots, B_k)$ be a non-empty list of k attributes of R, i.e. $(B_1, \ldots, B_k) = (A_{j_1}, \ldots, A_{j_k})$
- Then $\pi_{B_1,\ldots,B_k}(R)$ is a RA expression with schema $(B_1, \ldots, B_k)$
- Its value is the relation $\{(r.B_1, \ldots, r.B_k) \mid r \in R\}$
- In other words, each tuple r of R is "projected" on its $(B_1, \ldots B_k)$ components.
- $\pi_{B_1,\ldots,B_k}(R)$ is expressed in SQL by

```
SELECT DISTINCT   r.B1, ..., r.Bk
FROM              R r
```

- Observe that $|\pi_{B_1,\ldots,B_k}(R)| \leq |R|$

RA operators: projection π (Example)

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_1	b_3	c_1
a_2	b_2	c_1

$\pi_{A,C}(R)$

A	C
a_1	c_1
a_2	c_2
a_2	c_1

$\pi_{B,A}(R)$

B	A
b_1	a_1
b_2	a_2
b_3	a_1

Notice that duplicates are eliminated and that projection permits attributes to be permuted

RA operators: projection π

- A projection provides a mechanism to get certain columns from R
- It provides a vertical slice of *R*

RA operators: projection π (the general case)

- More generally, projection applies to a RA expression $E(A_1, \ldots, A_m)$
- $\pi_{B_1,\ldots,B_k}(E)$ has schema $(B_1, \ldots, B_k)$
- Its value is the set all tuples obtained from the tuples in the value of E projected on their $B_1, \ldots, B_k$ components
- If Q_E denotes the SQL query corresponding to E then $\pi_{B_1,\ldots,B_k}(E)$ is expressed in SQL by[3]

```
SELECT DISTINCT   e.B1, ..., e.Bk
FROM              (QE) e
```

- The DISTINCT clause is required to ensure that SQL returns a set. I.e., duplicates are eliminated.

[3] If E is a relation R, we can replace (Q_E) by R

The projection operator π (the general case)

- Alternatively, assume Q_E has the form

  SELECT $A_1, \ldots, A_m$
  FROM ...
  WHERE ...

  then the translation of $\pi_{B_1,\ldots,B_k}(E)$ is

  SELECT DISTINCT $B_1, \ldots, B_k$
  FROM ...
  WHERE ...

Example query in RA and SQL

- Consider the relations Student(sid,sname,age) and Enroll(sno,cno)[4]
- "Find the sid and age of each enrolled student whose name is Ann."
- This query can be expressed by the RA expression

$$\pi_{\text{sid, age}}(\sigma_{\text{sid=sno}}(\sigma_{\text{sname='Ann'}}(\text{Student} \times \text{Enroll})))$$

- In SQL

```
SELECT   DISTINCT s.sid, s.age
FROM     Student s, Enroll e
WHERE    s.sname = 'Ann' AND s.sid = e.sno
```

[4]Notice that we have used two different names sid and sno to refer uniquely to a student. We did this to ensure that the relations have no commonly named attributes.

Example query in RA and SQL

- "Find the sid and age of each enrolled student whose name is Ann."
- This query can also be expressed by the RA expression

$$\pi_{\text{sid, age}}(\sigma_{\text{sid}=\text{sno}}(\pi_{\text{sid,sage}}(\sigma_{\text{sname}=\text{'Ann'}}(\text{Student})) \times \pi_{\text{sno}}(\text{Enroll}))))$$

- In SQL

```
SELECT   DISTINCT s_Ann.sid, s_Ann.age
FROM     (SELECT DISTINCT s.sid, s.age
          FROM    Student s
          WHERE   s.sname = 'A') s_Ann CROSS JOIN
         (SELECT DISTINCT e.sno FROM Enroll e) e_s
WHERE    s_Ann.sid = e_s.sno
```

Boolean conditions in selection operations

- It is possible to extend the selection operations by permitting boolean combinations of basic conditions "$A_j \, \theta \, a$" and "$A_i \, \theta \, A_j$" using the boolean connectors $\wedge$, $\vee$, and $\neg$.
- "Find the sid and age of each enrolled student whose name is Ann."
- This query can then be expressed using the following RA expression

$$\pi_{\text{sid, age}}(\sigma_{\text{sname}=\text{'Ann'} \wedge \text{sid}=\text{sno}}(\text{Student} \times \text{Enroll}))$$

- Notice that the structure of this query is nearly the same as that of its SQL equivalent

Some comments about attribute names (renaming)

- Cartesian products require that the attributes names of the participating expressions do not overlap
- There are various ways to overcome this by renaming attributes or using some conventions
- For example, let $R(A, B)$ and $S(B, C)$ be two relations. We can not write $R \times S$ since attribute B occurs in the schemas of R and S. Nonetheless we will permit this and agree that the output schema of $R \times S$ is $(A, R.B, S.B, C)$ where we have used the relation names R and S to differentiate the B attribute in R from the B attribute in S

Some comments about attribute names

- We can also not write $R \times R$. To overcome this problem, we assume that for each relation name R, there is a series of relations $R_1, R_2, \cdots$ that are "copies" of R.
- Then instead of writing $R \times R$, we write $R_1 \times R_2$ and the output schema will be $(R_1.A, R_1.B, R_2.A, R_2.B)$
- In SQL, we can always use the AS clause to deal with attribute renaming. We will not give the details of attribute renaming in general but assume that it can always be done appropriately.

Some comments about attribute names (Example)

- "Find the sids of students who take at least two courses."

$$\pi_{E_1.sid}(\sigma_{E_1.sid = E_2.sid \wedge E_1.cno \neq E_2.cno}(Enroll_1 \times Enroll_2))$$

where E_1 and E_2 are abbreviations for $Enroll_1$ and $Enroll_2$, respectively

Boolean set operations (union, intersection, and set difference)

- Let E_1 and E_2 be two RA expressions with the same schema then
  - $E_1 \cup E_2$,
  - $E_1 \cap E_2$, and
  - $E_1 - E_2$

  are RA expressions with the same schema
- If Q_{E_1} and Q_{E_2} are the SQL queries corresponding to E_1 and E_2, respectively, then the expressions are expressed in SQL by
  - Q_{E_1} *UNION* Q_{E_2},
  - Q_{E_1} *INTERSECT* Q_{E_2}, and
  - Q_{E_1} *EXCEPT* Q_{E_2}

Examples

- In this example, assume relation $Enroll(sid, cno, grade)$
- "Find the sid of each student who received an 'A' in some course."

$$\pi_{sid}(\sigma_{grade=\text{'A'}}(Enroll))$$

- "Find the sid of each student who received an 'A' in some course and a 'B' in some course."

$$\pi_{sid}(\sigma_{grade=\text{'A'}}(Enroll)) \cap \pi_{sid}(\sigma_{grade=\text{'B'}}(Enroll))$$

- "Find the sid of each student who did not receive an 'A' in any course."

$$\pi_{sid}(Student) - \pi_{sid}(\sigma_{grade=\text{'A'}}(Enroll))$$

- "Find the sids of students who are enrolled in at least one course"

$$\pi_{sid}(Enroll)$$

- "Find the sids of students who take at least two courses"

$$\pi_{E_1.sid}(\sigma_{E_1.sid = E_2.sid \wedge E.cno \neq E_2.cno}(Enroll_1 \times Enroll_2))$$

- "Find the sids of students who take exactly one course"

$$\pi_{sid}(Enroll) - \pi_{E_1.sid}(\sigma_{E_1.sid = E_2.sid \wedge E.cno \neq E_2.cno}(Enroll_1 \times Enroll_2))$$

Examples (NOT ONLY and ONLY)

- "Find the cnos of CS courses."

$$\pi_{cno}(\sigma_{dept='CS'}(Course))$$

Denote this expression by CS

- "Find sids of students who do not only take CS courses."

$$\pi_{sid}(Enroll - (\pi_{sid}(Student) \times CS))$$

- "Find sids of students who only take CS courses."

$$\pi_{sid}(Student) - \pi_{sid}(Enroll - (\pi_{sid}(Student) \times CS))$$

Examples (NOT ALL and ALL)

- "Find sids of students who do not take all CS courses"

$$\pi_{sid}((\pi_{sid}(Student) \times CS) - Enroll)$$

- "Find sids of students who take all CS courses"

$$\pi_{sid}(Student) - \pi_{sid}(\pi_{sid}(Student) \times CS) - Enroll)$$

Examples

Consider the relation $Person(pid, age)$

- "Find the pids of persons who are not the youngest."

$$\pi_{P_1.pid}(\sigma_{P_1.age > P_2.age}(Person_1 \times Person_2))$$

- "Find the pids of the youngest persons."

$$\pi_{pid}(Person) - \pi_{P_1.pid}(\sigma_{P_1.age > P_2.age}(Person_1 \times Person_2))$$