Relational Algebra

CSE462 Database Concepts

Demian Lessa/Jan Chomicki

Department of Computer Science and Engineering
State University of New York, Buffalo

Fall 2013

Introduction

Relational Algebra (RA) is an algebra of relations that provides simple yet powerful ways to construct new relations from existing ones. It is related both to first-order logic and set algebra.

- RA is fundamentally an abstract query language.
- Hence, modern database systems do not use RA.
- Instead, they use a concrete language such as SQL.
- It is important to note, however, that RA is at the core of SQL.
- DBMSs translate queries into RA (or variant) during query processing.

Introduction

Why RA?

- I can do anything with <your favorite PL>!
  - Yes, but only in principle. In practice. . .
  - How do you represent tuples in <your favorite PL>?
  - How do multiple users share, query, and updated their data?
  - How do you achieve this while keeping, e.g., physical data independence?
- Practical importance.
  - RA is striclty less powerful than <your favorite PL>.
  - Easy to use, e.g., fewer and simpler syntactic constructs.
  - This allows the DBMS to search for efficient query evaluation plans.
  - RA is still expressible enough to be practically useful.
- Limitations.
  - Finite relations only (this is usually not a problem).
  - Set semantics (i.e., tuple duplication not allowed).
  - No aggregate functions (e.g., MIN, MAX, AVG);
  - No recursion (e.g., transitive closure);
  - No ordering (i.e., tuples returned in non-deterministic order).

Introduction

An algebra consists of one or more sets closed under one or more operations, satisfying some axioms.

- RA deals with sets of relations closed under certain operations.
- A relation is a set of k-tuples where tuple components are named.
- Relations are finite: their arity and extension are both finite.
- RA introduces six primitive operations.
  - Set union and set difference.
  - Selection, projection, cartesian product, and renaming.
- Additional operations may be included, for convenience.
  - However, they do not add expressive power to RA.
  - That is, they can be defined in terms of the primitive operations.

Relational Algebra Constituents

- Operands.
  - Variables, that stand for relations.
  - Constants, that stand for fixed, finite relations.
- Primitive operations.
  - Set union ($\cup$), set difference ($-$).
  - Selection (σ), projection (π), cartesian ($\times$).
  - Rename (ρ).
- Derived operations (not extensive).
  - Set intersection ($\cap$).
  - Natural join ($\bowtie$), theta join ($\bowtie_\theta$).
  - Quotient ($\div$).

Relational Algebra Operations

- Removing parts of a relation.
  - Selection eliminates rows, projection eliminates columns.
- Combining tuples.
  - Cartesian product pairs tuples of two relations in all possible ways.
  - Join pairs tuples of two relations selectively.
- Schema-preserving.
  - All set operations and the selection operation.
  - Rename modifies a relation schema without affecting its tuples.
  - Cartesian and joins output a relation with a “merged” schema.
- Operator Arity.
  - Unary: selection, projection, rename.
  - Binary: cartesian, all join operations, all set operations.
- Monotonicity.
  - All primitive operations, except set difference.

Set Operations ($\cup, \cap, -$)

- Schema compatibility requirement.
  - $R \; op \; S$, R and S relations with the same arity, $op \in \{\cup, \cap, -\}$.
  - Attribute names and types must match based on presentation order.
- Set union ($\cup$).
  - $R \cup S$ is the set of tuples that are in R or S or both.
  - Is $R \cup S = S \cup R$?
- Set intersection ($\cap$).
  - $R \cap S$ is the set of tuples that are in both R and S.
  - Is $R \cap S = S \cap R$?
- Set difference ($-$).
  - $R - S$ is the set of tuples that are in R but not in S.
  - Is $R - S = S - R$?
- In all set operations, a tuple may only appear once in the result.
- Hint: use renaming to achieve schema-compatibility.

Set Operations: Example

name	address	gender	birthday
Carrie Fisher	123 Maple St., Holywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88

Relation : Contacts owned by George Lucas (R).

name	address	gender	birthday
Carrie Fisher	123 Maple St., Holywood	F	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

Relation : Contacts owned by Steven Spielberg (S).

Answer:

- What are the schemas of R and S?
- What are the results of: $R \cup S$, $R \cap S$, $R - S$, and $S - R$?

Projection (π)

- Projection takes a relation R, removes some of its attributes and/or rearranges its (remaining) attributes. It implicitly performs duplicate elimination as necessary.
- The projection of $R(A_1, \ldots, A_m)$ onto components $A_{i_1}, \ldots, A_{i_k}$, where every i_j is an integer in the range 1 to m, is denoted $\pi_{A_{i_1}, \ldots, A_{i_k}}(R)$.
- Semantics: for every tuple $(b_1, \ldots, b_k)$ in $\pi_{A_{i_1}, \ldots, A_{i_k}}(R)$, there exists a tuple $(a_1, \ldots, a_m)$ in R for which $b_j = a_{i_j}$ for all $1 \leq j \leq k$.
- Projection may also specify attributes by position. Note: do not combine names and positions when specifying a projection!

Projection: Conceptual Examples

- Compute $\pi_{C,A,E}(R)$ for $R(A, B, C, D, E)$ using the definition.
  - From the definition, $(C, A, E) = (A_{i_1}, A_{i_2}, A_{i_3}) = (A_3, A_1, A_5)$.
  - Assume $(b_1, b_2, b_3) \in \pi_{C,A,E}(R)$.
  - Then, $(a_1, \ldots, a_m) \in R$ such that $(b_1, b_2, b_3) = (a_{i_1}, a_{i_2}, a_{i_3})$.
  - Using the values of the indexed subscripts, $(b_1, b_2, b_3) = (a_3, a_1, a_5)$.
  - But we know that $(A_3, A_1, A_5) = (C, A, E)$.
  - Thus, (b_1, b_2, b_3) are precisely the (C, A, E) components of $(a_1, \ldots, a_m)$.
- Equivalent projections for $R(A, B, C, D, E)$ using names and indexes.
  - Relations $\pi_{B,C,D}(R)$ and $\pi_{2,3,4}(R)$ are equivalent.
  - Relations $\pi_{C,A,E}(R)$ and $\pi_{3,1,5}(R)$ are equivalent.

Projection: Example

title	year	length	genre
Star Wars	1977	124	scifi
Galaxy Quest	1999	104	comedy
Wayne's World	1992	95	comedy

Table : Movies.

Compute : $\pi_{\texttt{title,year,length}}(\texttt{Movies})$

title	year	length
Star Wars	1977	124
Galaxy Quest	1999	104
Wayne's World	1992	95

Compute : $\pi_{\texttt{genre}}(\texttt{Movies})$

genre
scifi
comedy

Selection (σ)

- Selection takes a relation R and a formula φ and removes all tuples from R that do not satisfy φ. The formula φ consists of:
  - Operands: constants and attribute names.
  - Comparison: $<$, $=$, $>$, $\leq$, $\neq$, $\geq$.
  - Logical: AND ($\wedge$), OR ($\vee$), NOT ($\neg$) and the usual precedence: $\neg > \wedge > \vee$.
- The selection of $R(A_1, \ldots, A_m)$ with formula φ is denoted $\sigma_\varphi(R)$.
- The output schema of $\sigma_\varphi(R)$ is identical to the schema of R.
- Semantics: a tuple $(a_1, \ldots, a_m)$ in R is also in $\sigma_\varphi(R)$ if, for all $1 \leq i \leq m$, when we substitute every occurrence of A_i in φ for a_i, φ becomes true.

Selection: Example

title	year	length	genre
Star Wars	1977	124	scifi
Galaxy Quest	1999	104	comedy
Wayne's World	1992	95	comedy

Table : `Movies`.

Compute : $\sigma_{\texttt{length} \geq \texttt{100}}(\texttt{Movies})$

title	year	length	genre
Star Wars	1977	124	scifi
Galaxy Quest	1999	104	comedy

Selection: Example

title	year	length	genre
Star Wars	1977	124	scifi
Galaxy Quest	1999	104	comedy
Wayne's World	1992	95	comedy

Table : Movies.

Compute : $\sigma_{\texttt{length} \geq \texttt{100 AND genre} = \texttt{'comedy'}}(\texttt{Movies})$

title	year	length	genre
Galaxy Quest	1999	104	comedy

Selection: Example

title	year	length	genre
Star Wars	1977	124	scifi
Galaxy Quest	1999	104	comedy
Wayne's World	1992	95	comedy

Table : Movies.

Compute : $\sigma_{\texttt{title} = \text{`}\texttt{E.T.}\text{'}}(\texttt{Movies})$

title	year	length	genre

Cartesian Product ($\times$)

- Cartesian product (also cross product) takes relations R and S and computes the set of all possible tuples obtained from pairing every tuple in R with every tuple in S.
- The Cartesian product of R and S is denoted $R \times S$.
- The output schema of $R \times S$ contains all attributes from both R and S.
  - If R and S have common attributes, new names are assigned to at least one (but usually both) of each pair of identical attributes.
  - By convention, we disambiguate by qualifying the attribute names with their relation names. E.g., for a common attribute A, we use $R.A$ and $S.A$.
- Semantics: Let R and S have arities k_1 and k_2, respectively. $R \times S$ is the set of all $(k_1 + k_2)$-tuples whose first k_1 components come from a tuple in R and whose last k_2 components come from a tuple in S. If R and S have, respectively, n_1 and n_2 tuples, then $R \times S$ is a set of $n_1 \cdot n_2$ tuples.

Cartesian Product: Example

Relation : R

A	B
1	2
3	4

Relation : S

B	C	D
2	5	6
4	7	8
9	10	11

Compute : $R \times S$

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Cartesian Product: Example

Relation : R

A	B
1	2
3	4

Relation : S

B	C	D

Compute : $R \times S$

A	R.B	S.B	C	D

Theta Join ($\bowtie_\theta$)

- Theta join is a derived operation that takes relations R and S, a formula θ consisting of arithmetic comparisons between R and S attributes, and returns all tuples in $R \times S$ satisfying the formula θ. References to common attributes in R and S must be qualified in θ.
- The theta join of R and S with formula θ is denoted $R \bowtie_\theta S$.
- The output schema of $R \bowtie_\theta S$ is the same as the schema of $R \times S$.
- Semantics: $R \bowtie_\theta S$ is the result of $\sigma_\theta(R \times S)$.
- If θ only involves equalities, it is called an equijoin.

Theta Join: Example

Relation : U

A	B	C
1	2	3
6	7	8
9	7	8

Relation : V

B	C	D
2	3	4
2	3	5
7	8	10

Compute : $U \bowtie_{A<D} V$

A	U.B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Rename (ρ)

- The rename operations takes a relation R and returns a relation with the same set of tuples but a different schema. Rename can modify the name of the input relation as well as any of its attributes.
- To rename relation $R(A_1, \ldots, A_k)$ to $S(B_1, \ldots, B_k)$, use $\rho_{S(B_1,\ldots,B_k)}(R)$.
  - Semantics: The result of $\rho_{S(B_1,\ldots,B_k)}(R)$ is a relation named S, attributes named $B_1, \ldots, B_k$, and the same set of tuples as R.
- To rename relation $R(A_1, \ldots, A_k)$ to $S(A_1, \ldots, A_k)$, use $\rho_S(R)$.
  - Semantics: The result of $\rho_S(R)$ is a relation named S, attributes named $A_1, \ldots, A_k$, and the same set of tuples as R.
- To rename relation $R(A_1, \ldots, A_k)$ to $R(B_1, \ldots, B_k)$, use $R(B_1, \ldots, B_k)$.
  - Semantics: The result of $R(B_1, \ldots, B_k)$ is a relation named R, attributes named $B_1, \ldots, B_k$, and the same set of tuples as R.

Rename: Example

Relation : U

A	B	C
1	2	3
6	7	8
9	7	8

Relation : V

B	C	D
2	3	4
2	3	5
7	8	10

Compute : $U \bowtie_{A<D} \rho_{T(C,D,E)}(V)$

A	B	U.C	T.C	D	E
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10

Natural Join ($\bowtie$)

- Natural join is an equijoin that takes relations R and S and returns all tuples in $R \times S$ that agree on the values of their common attributes.
- The natural join of R and S is denoted $R \bowtie S$.
- The schema of $R \bowtie S$ is the union of the schemas of R and S: identical attributes are unqualified and included only once.
- Semantics:

  Given $R(A_1, \ldots, A_k, B_1, \ldots, B_m)$ and $S(A_1, \ldots, A_k, C_1, \ldots, C_n)$, the result of $R \bowtie S$ is:

$$\pi_{A_1,\ldots,A_k,B_1,\ldots,B_m,C_1,\ldots,C_n}(R \bowtie_{A_1=D_1 \wedge \cdots \wedge A_k=D_k} \rho_{S(D_1,\ldots,D_k,C_1,\ldots,C_n)}(S))$$

- Note: if R and S have no common attributes, the natural join reduces to a cartesian product.

Natural Join: Example

Relation : U

A	B	C
1	2	3
6	7	8
9	7	8

Relation : V

B	C	D
2	3	4
2	3	5
7	8	10

Compute : $U \bowtie V$

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Other kinds of joins

- Semijoin:

$$R \ltimes_\theta S = \pi_R(R \bowtie_\theta S).$$

- Anti-semijoin: $R - (R \ltimes_\theta S)$.

Quotient ($\div$)

- Quotients are useful for expressing universal quantification.
- The quotient takes relations R and S and returns the largest relation T satisfying $T \times S \subseteq R$. The attribute sets of S and T must form a partition of the attribute set of R.
- The quotient of R and S is denoted $R \div S$.
- The output schema of $R \div S$ consists of the attributes in R but not in S.
- Semantics: Given $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$ and $S(B_1, \ldots, B_m)$, $R \div S$ returns a set of tuples over the attributes $A_1, \ldots, A_n$ such that, for every tuple $(a_1, \ldots, a_n)$ in $R \div S$ and every tuple $(b_1, \ldots, b_m)$ in S, the tuple $(a_1, \ldots, a_n, b_1, \ldots, b_m)$ is in R.
- Quotient is a derived operation:

$\pi_{A_1,\ldots,A_n}(R) \times S$	possible
$\pi_{A_1,\ldots,A_n}(R) \times S - R$	possible - actual
$\pi_{A_1,\ldots,A_n}(\pi_{A_1,\ldots,A_n}(R) \times S - R)$	π(possible - actual)
$\pi_{A_1,\ldots,A_n}(R) - \pi_{A_1,\ldots,A_n}(\pi_{A_1,\ldots,A_n}(R) \times S - R)$	π(actual) - π(possible - actual)

Quotient: Example

Relation : `Account`

Customer	BranchName
Hewitt	Buffalo
Blake	Amherst
Blake	Buffalo
Blake	Depew
Fox	Amherst
Fox	Buffalo
Smith	Lockport

Relation : `Branch`

BranchName
Amherst
Buffalo

Compute : `Account` $\div$ `Branch`

Customer
Blake
Fox

Beyond Basic Operations

- RA allows us to create expressions by composing operations.
  - This property holds because RA is closed under the defined operations.
  - Arbitrarily complex relations can be created by composing subexpressions.
  - Parenthesis are used to group subexpressions, for clarity.
- RA expressions may be represented as trees.
  - Leaf nodes are stored relations.
  - Internal nodes are operators.
  - Subtrees are subexpressions.
- RA expressions can also be represented using a linear notation.
  - List $A_1, \ldots, A_k$ of assignments.
  - Each A_i has the form $R(v_1, \ldots, v_n) := expr$
    - `lhs` is a new relation name and a list of attributes.
    - `rhs` is a RA expression referencing stored relations or any A_j, $j < i$.

Linear Notation: Example

- Schema: `Movies(title, year, length, genre, studioName)`.
- List the title and year of Fox movies that run for at least 100 minutes.
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{studioName=\text{`Fox'}}(Movies)$
  - $T(t,y,l,g,s) := R \cap S$
  - $Answer(title, year) := \pi_{t,y}(T)$
- There are many ways to do it...
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{s=\text{`Fox'}}(R)$
  - $Answer(title, year) := \pi_{t,y}(S)$

Expression Equivalence

- Different RA expressions may perform the same computation.
- These are called equivalent expressions.
  - $E_1 \equiv E_2 \Leftrightarrow E_1(D) = E_2(D)$ for every database instance D.
  - Remember, $E_1(D) = E_2(D) \Leftrightarrow E_1(D) \subseteq E_2(D) \wedge E_2(D) \subseteq E_1(D)$.
- This fact is often explored by query optimizers in DBMSs.
  - A user query may have many equivalent expressions.
  - Some (sub)expressions are much faster to evaluate.
  - The DBMS may replace one (sub)expression for an equivalent one that is more efficiently evaluated.
- For instance, let R and S be schema-compatible.
  - $R \cap S \equiv R - (R - S) \equiv S - (S - R)$

Set vs Bag Semantics

- Queries may produce (intermediate) duplicate tuples that need to be eliminated under set semantics, e.g., union and projection.
- Most bag operations are more efficient than their set counterparts.
- Real applications need both set and bag semantics.
- Under bag semantics,
  - projection may create duplicate tuples;
  - selection is applied to each tuple independently;
  - cartesian is applied to each pair of tuples independently;
  - join matches pairs of tuples independently.
- Let tuple t occur n times in R and m times in S, respectively.
  - t appears $n+m$ times in $R \cup S$
  - t appears $min(n,m)$ times in $R \cap S$
  - t appears $max(0, n-m)$ times in $R - S$
- Let tuples r and s occur n times in R and m times in S, respectively.
  - rs appears $n \cdot m$ times in $R \times S$

Required

- Read sections 2.4 of chapter #2 and 5.1 of chapter #5.