Relational Algebra

CSE462 Database Concepts

Demian Lessa

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

January 26–28, 2011

Next...

1. Relatonal Algebra

Relational Algebra

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**.

- Modern systems do not use RA as their query language.
- 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.

Relational Algebra (cont.)

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?

Relational Algebra (cont.)

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.
  - **Striclty less powerful** than <your favorite PL>.
  - Easy to use, e.g., fewer and simpler syntactic constructs.
  - Simpler allows the DBMS to search for efficient query evaluations.
  - Still, expressible enough that it is practically useful.

Relational Algebra (cont.)

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.
  - **Striclty less powerful** than <your favorite PL>.
  - Easy to use, e.g., fewer and simpler syntactic constructs.
  - Simpler allows the DBMS to search for efficient query evaluations.
  - Still, expressible enough that it is practically useful.
- Limitations.
  - Finiteness of relations. Not all operations are closed.
  - Aggregates (e.g., MIN, MAX, AVG);
  - Recursion (e.g., transitive closure);
  - Ordering.

Relational Algebra (cont.)

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 must be finite: their arity and extension must be finite.
- RA introduces six primitive operations.
  - Set union and set difference.
  - Selection, projection, cartesian product, rename.
- Several derived operations may be defined.
  - However, they **do not add expressive power to RA**.

Relational Algebra (cont.)

Relational Algebra constituents.

- Operands.
  - Variables that stand for relations.
  - Constants, which are 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 ($\bowtie_\theta$).
  - Left, right, and full outer joins.
  - Quotient ($\div$).

Relational Algebra (cont.)

Some properties of relational 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, selection.
  - Rename modifies a relation schema without affecting its tuples.
  - Cartesian and joins output a relation with a “merged” schema.
- Arity.
  - Unary: selection, projection, rename.
  - Binary: cartesian, all join operations, all set operations.
- Monotonicity.
  - All primitive operations, except set difference.

Relational Operations

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 the rename operation to achieve schema-compatibility.

Relational Operations (cont.)

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 $S - R$?

Relational Operations (cont.)

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 index (position).
  - Note: do not combine names and indexes when specifying a projection!

Relational Operations (cont.)

Conceptual examples for projection (π).

- 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.

Relational Operations (cont.)

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

Table: `Movies`.

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

Relational Operations (cont.)

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

Relational Operations (cont.)

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

Relational Operations (cont.)

Selection (σ).

- Selection takes a relation R and a formula φ and removes all tuples from R that do not satisfy φ. The selection formula φ consists of:
  - Operands: constants and attribute names.
  - Arithmetic comparisons: $<$, $=$, $>$, $\leq$, $\neq$, $\geq$.
  - Logical operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$).
  - Logical precedence order applies: NOT, AND, OR.
- The selection of $R(A_1, \ldots, A_m)$ with formula φ is denoted $\sigma_\varphi(R)$.
  - 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.
- The schema of $\sigma_\varphi(R)$ is identical to the schema of R.

Relational Operations (cont.)

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

Table: `Movies`.

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

Relational Operations (cont.)

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

Table: `Movies`.

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

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

Relational Operations (cont.)

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

Table: `Movies`.

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

Relational Operations (cont.)

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

Table: `Movies`.

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

title	year	length	genre
Galaxy Quest	1999	104	comedy

Relational Operations (cont.)

Cartesian ($\times$).

- Cartesian product (also cross product) takes relations R and S and computes the set of all possible tuples obtained from pairing every tuple from R with every tuple from S.
- The cartesian of R and S is denoted $R \times S$.
  - 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.
- The 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. For a common attribute A, we use $R.A$ and $S.A$.

Relational Operations (cont.)

Relation: R

A	B
1	2
3	4

Relation: S

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

Compute: $R \times S$

Relational Operations (cont.)

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

Relational Operations (cont.)

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 $R \bowtie_\theta S$.
  - Semantics: $R \bowtie_\theta S$ is the result of $\sigma_\theta(R \times S)$.
- If θ only involves equalities, it is called an **equijoin**.

Relational Operations (cont.)

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 $R \bowtie S$.
  - Semantics: Given $R(A_1, \ldots, A_k, B_1, \ldots, B_m)$ and $S(A_1, \ldots, A_k, C_1, \ldots, C_n)$,

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

    is the result of $R \bowtie S$.
- The schema of $R \bowtie S$ is the union of the schemas of R and S: identical attributes are included only once.
- Note: if R and S have no common attributes, the natural join degrades to a cartesian product.

Relational Operations (cont.)

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

Relational Operations (cont.)

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

Relational Operations (cont.)

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: $U \bowtie V$

Relational Operations (cont.)

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

Relational Operations (cont.)

Quotient ($\div$).

- The quotient takes relations R and S and returns the largest relation T satisfying $S \times T \subseteq R$. The attribute sets of S and T must form a partition of the attribute set of R. Quotients are useful for expressing universal quantification.
- The quotient of R and S is denoted $R \div S$.
  - 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)

Relational Operations (cont.)

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`

Relational Operations (cont.)

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

Relational Operations (cont.)

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.

Relational Operations (cont.)

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

Relational Operations (cont.)

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

Relational Expressions

Beyond simple operations.

- RA allows us to create expressions by composing operations.
  - RA would not be useful if it only supported unary of binary operations.
  - Arbitrarily complex relations can be created by composing subexpressions.
  - Parenthesis are used to indicate grouping of operands or 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 $A_j, j < i$.

Relational Expressions (cont.)

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.

Relational Expressions (cont.)

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.
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{studioName = \text{`}Fox\text{'}}(Movies)$

Relational Expressions (cont.)

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.
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{studioName = \text{`}Fox\text{'}}(Movies)$
  - $T(t,y,l,g,s) := R \cap S$

Relational Expressions (cont.)

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.
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{studioName='Fox'}(Movies)$
  - $T(t,y,l,g,s) := R \cap S$
  - $Answer(title, year) := \pi_{t,y}(T)$
- There are many ways to do it. . .

Relational Expressions (cont.)

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.
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{studioName='Fox'}(Movies)$
  - $T(t,y,l,g,s) := R \cap S$
  - $Answer(title,year) := \pi_{t,y}(T)$
- There are many ways to do it...
  - $R(t,y,l,g,s) := \sigma_{length \geq 100}(Movies)$
  - $S(t,y,l,g,s) := \sigma_{s='Fox'}(R)$
  - $Answer(title,year) := \pi_{t,y}(S)$

Relational Expressions (cont.)

Equivalence.

- More than one RA expression may perform the same computation.
- These are 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.
  - $R \cap S \equiv R - (R - S)$

Set vs Bag Semantics

Practical issues.

- Applications may require duplication.
- Queries may produce (intermmediate) duplication.
- Certain bag operations are more efficient than their set counterparts.
  - For example, union and projection.
- Let t be a tuple occurring n times in R and m times in S. Then,
  - 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 $R-S$
- 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.

Required

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