These lecture notes include some material from Professors Bertossi, Kolaitis, Guagliardo and Libkin

Relational Algebra

Lecture Handout

Dr Eugenia Ternovska

Simon Fraser University

Relational algebra

Relational algebra strikes a good balance between expressive power and efficiency.

Codd's key contribution was to identify a small set of basic operations on relations and to demonstrate that useful and interesting queries can be expressed by combining these operations.

- Thus, relational algebra is a rich enough language, even though, as we will see later on, it suffers from certain limitations in terms of expressive power.

The first RDBMS prototype implementations (System R and Ingres) demonstrated that the relational algebra operations can be implemented efficiently.

Basic Notions from Discrete Mathematics

A **Domain** is a finite set of objects, e.g., the set of people in our class, the set of all possible grades

A **k-tuple** is an ordered sequence of k objects (need not be distinct and can be from different domains)
$(2, 0, 1)$ is a 3-tuple; (a, b, a, a, c) is a 5-tuple, and so on.

If $D_1, D_2, \ldots, D_k$ are finite sets (domains), then
the **cartesian product** $D_1 \times D_2 \cdots \times D_k$ of these sets is the set of all k-tuples $(d_1, d_2, \ldots, d_k)$ such that $d_i \in D_i$, for $1 \leq i \leq k$.

Basic Notions from Discrete Mathematics

Example
If $D_1 = \{0, 1\}$ and $D_2 = \{a, b, c, d\}$, then
$D_1 \times D_2 = \{(0, a), (0, b), (0, c), (0, d), (1, a), \ldots\}$
(usually written as a table)

Warning: Computing Cartesian products is an expensive operation!

Fact: Let $|D|$ denote the cardinality ($=$ # of elements) of a set D.
Then $|D_1 \times D_2 \times \cdots \times D_k| = |D_1| \times |D_2| \times \cdots \times |D_k|$.

In the example above, $|D_1| \times |D_2| = 8$.

Basic Notions from Discrete Mathematics

A k-ary relation R is a subset of a cartesian product of k sets, i.e., $R \subseteq D_1 \times D_2 \times \cdots \times D_k$.

Example

Unary $R = \{0, 2, 4, \ldots, 100\}$ $(R \subseteq D)$
Binary $T = \{(a, b) : a$ and b have the same birthday$\}$
Ternary $S = \{(m, n, s) : s = m + n\}$
$\ldots$

In Relational Data Model, relations are recorded as tables that have names for their columns, called **attributes**

Example of a table

Customer

CustID	**Name**	**City**	**Age**
cust1	Renton	Edinburgh	24
cust2	Watson	London	32
cust3	Holmes	London	35

What are the attributes in this table?

What are the tuples in this table?

The Basic Operations of Relational Algebra

Group I: Three standard set-theoretic binary operations:

- Union
- Difference
- Cartesian Product.

Group II. Two unary operations on relations:

- Projection
- Selection.

Group III. One special operation:

- Renaming

Relational Algebra consists of all expressions obtained by combining these basic operations in syntactically correct ways.

Relational algebra

Procedural query language

A relational algebra expression

- takes as input one or more relations
- applies a **sequence of operations**
- returns a relation as output

Operations:

Projection	(π)	Union	$(\cup)$
Selection	(σ)	Intersection	$(\cap)$
Product	$(\times)$	Difference	$(-)$
Renaming	(ρ)		

The application of each operation results in a new relation that can be used as input to other operations

Projection

- **Vertical operation**: choose some of the **columns**
- Syntax: $\pi_{\text{sequence of attributes}}(\text{relation})$
- $\pi_{A_1,\ldots,A_n}(R)$ takes only the values of attributes $A_1, \ldots, A_n$ for each tuple in R

Customer

CustID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

$\pi_{\text{Name,City}}(\text{Customer})$

Name	City
Renton	Edinburgh
Watson	London
Holmes	London

Selection

- **Horizontal operation**: choose **rows** satisfying some condition
- Syntax: $\sigma_{\Theta}(\text{relation})$, where Θ is a condition
- A *family* of unary operations, one for each condition Θ
- $\sigma_{\theta}(R)$ takes only the tuples in R for which θ is satisfied

$$
\begin{aligned}
\text{term} &:= \text{attribute} \mid \text{constant} \\
\theta &:= \text{term } \mathbf{op} \text{ term with } \mathbf{op} \in \{=, \neq, >, <, \geqslant, \leqslant\} \\
&\quad \mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta
\end{aligned}
$$

Example of selection

Customer

CustID	Name	City	Age
cust1	Renton	Edinburgh	24
cust2	Watson	London	32
cust3	Holmes	London	35

$\sigma_{\text{City} \neq \text{'Edinburgh'} \wedge \text{Age} < 33}(\text{Customer})$

CustID	Name	City	Age
cust2	Watson	London	32

More on the Selection Operator

Note: The use of the comparison operators $<$, $>$, $\leq$, $\geq$ assumes that the underlying domain of values is **totally ordered**.

If the domain is not totally ordered, then only $=$ and $\neq$ are allowed.

If we do not have attribute names (hence, we can only reference columns via their column number), then we need to have a special symbol, say \$, in front of a column number.

Thus, $\$4 > 100$ is a meaningful basic clause $\$1 =$"Apto" is a meaningful basic clause, and so on.

Efficiency (1)

Consecutive selections can be combined into a single one:

$$\sigma_{\theta_1}\big(\sigma_{\theta_2}(R)\big) = \sigma_{\theta_1 \wedge \theta_2}(R)$$

Example

$Q_1 = \sigma_{\text{City}\neq\text{'Edinburgh'}}\big(\sigma_{\text{Age}<33}(\text{Customer})\big)$

$Q_2 = \sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}(\text{Customer})$

$Q_1 = Q_2$ but Q_2 **faster** than Q_1 in general

Efficiency (2)

Projection can be pushed inside selection

$$\pi_\alpha\big(\sigma_\theta(R)\big) = \sigma_\theta\big(\pi_\alpha(R)\big)$$

only if all attributes mentioned in θ appear in α

Example

$Q_1 = \pi_{\text{Name,City,Age}}\big(\sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}(\text{Customer})\big)$

$Q_2 = \sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}\big(\pi_{\text{Name,City,Age}}(\text{Customer})\big)$

Question: Which one is more efficient?

Combine Operations: Cartesian product

$R \times S$ **concatenates** each tuple of R with all the tuples of S

Example

R	A	B
	1	2
	3	4

$\times$

S	C	D
	1	a
	2	b
	3	c

$=$

$R \times S$	A	B	C	D
	1	2	1	a
	1	2	2	b
	1	2	3	c
	3	4	1	a
	3	4	2	b
	3	4	3	c

Note: all attributes must be different
If attribute A in common, R.A and S.A used to disambiguate

Expensive operation:

- $\text{card}(R \times S) = \text{card}(R) \times \text{card}(S)$
- $\text{arity}(R \times S) = \text{arity}(R) + \text{arity}(S)$

Resulting relation schema is the union of R and S schemas

Join

Combining Cartesian product and selection

Customer: ID, Name, City, Address
Account: Number, Branch, CustID, Balance

We can join customers with the accounts they own as follows

$$\sigma_{\text{ID}=\text{CustID}}(\text{Customer} \times \text{Account})$$

Renaming

Gives a new name to some of the attributes of a relation

Syntax: $\rho_{\text{replacements}}(\text{relation})$,
where a replacement has the form $C \to D$, $A \to A'$, etc.

$$\rho_{A\to A', C\to D}\left(\begin{array}{ccc} \textbf{A} & \textbf{B} & \textbf{C} \\ \hline \text{a} & \text{b} & \text{c} \\ 1 & 2 & 3 \end{array}\right) = \begin{array}{ccc} \textbf{A}' & \textbf{B} & \textbf{D} \\ \hline \text{a} & \text{b} & \text{c} \\ 1 & 2 & 3 \end{array}$$

Example

Customer: **CustID**, Name, City, Address
Account: Number, Branch, **CustID**, Balance

$$\sigma_{\text{CustID}=\text{CustID}'}\left(\text{Customer} \times \rho_{\text{CustID}\to\text{CustID}'}(\text{Account})\right)$$

Set operations

Union

$$\begin{array}{c|cc} \text{R} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a2} & \text{b2} \end{array} \cup \begin{array}{c|cc} \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a3} & \text{b3} \end{array} = \begin{array}{c|cc} \text{R} \cup \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a2} & \text{b2} \\ & \text{a3} & \text{b3} \end{array}$$

Intersection

$$\begin{array}{c|cc} \text{R} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a2} & \text{b2} \end{array} \cap \begin{array}{c|cc} \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a3} & \text{b3} \end{array} = \begin{array}{c|cc} \text{R} \cap \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \end{array}$$

Difference

$$\begin{array}{c|cc} \text{R} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a2} & \text{b2} \end{array} - \begin{array}{c|cc} \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a1} & \text{b1} \\ & \text{a3} & \text{b3} \end{array} = \begin{array}{c|cc} \text{R} - \text{S} & \textbf{A} & \textbf{B} \\ \hline & \text{a2} & \text{b2} \end{array}$$

The relations must have the same set of attributes

Union and renaming

R	Father	Child
	George	Elizabeth
	Philip	Charles
	Charles	William

S	Mother	Child
	Elizabeth	Charles
	Elizabeth	Andrew

We want to find the relation **parent-child**

$$\rho_{\text{Father}\to\text{Parent}}(\text{R}) \cup \rho_{\text{Mother}\to\text{Parent}}(\text{S}) =$$

Parent	Child
George	Elizabeth
Philip	Charles
Charles	William
Elizabeth	Charles
Elizabeth	Andrew

Full relational algebra

Primitive operations: π , σ , $\times$, ρ , $\cup$, $-$

Removing any of these results in a **loss of expressive power**

Schemas For Results

• Union, intersection, and difference: The schemas of the two operands must be the same, so use that schema for the result

• Selection: Schema of the result is the same as the schema of the operand

• Projection: List of attributes tells us the schema

• Renaming: The operator tells the schema

• Product: Schema is the attributes of both relations
Use R.A, etc., to distinguish two attributesnamed A

Derived operations

some common examples

$\cap$ can be expressed in terms of difference:

$$R \cap S = R - (R - S)$$

$\bowtie$ can be expressed in terms of π , σ , $\times$, ρ (next slide)

There are many more examples

Derived relational algebra operations are operations on relations that are expressible via an expression built from the basic operators.

Derived Operation: Natural Join

$R \bowtie S$ – pairing two tables on **common attributes**

• Pair only those tuples of R and S that agree in whatever attributes are common to the schemas of R and S

• A tuple r and a tuple s are matched if and only if r and s agree on value on each of attributes A1, A2, . . . , An that are **in both schemas**

• The resulting tuple is called joined tuple

Derived Operation: Natural Join

Example

R	A	B
	1	2
	4	2
	3	4

$\bowtie$

S	B	C
	1	a
	2	b
	3	c

$=$

$R \bowtie S$	A	B	C
	1	2	b
	4	2	b

joined tuples: $(1, 2, b)$, $(4, 2, b)$

Dangling Tuple

- A tuple that fails to pair with any tuple of the other relation in a join
- Has no effect on the result of a natural join

dangling tuples: $(3, 4)$, $(1, a)$, $(3, c)$

Derived Operation: Natural Join

Express $\bowtie$ using basic operations of RA:

Example

Customer: **CustID**, Name, City, Address

Account: Number, Branch, **CustID**, Balance

Customer $\bowtie$ Account $=$

$$\pi_{X \cup Y}\left(\sigma_{\mathsf{CustID}=\mathsf{CustID}'}\left(\mathsf{Customer} \times \rho_{\mathsf{CustID} \to \mathsf{CustID}'}(\mathsf{Account})\right)\right)$$

where X = { all attributes of Customer }
Y = { all attributes of Account }

The schema of $\bowtie$ is the union of schemas, $X \cup Y$
Notice: no repeated attributes at all in the resulting schema

Other derived operations: Theta-join $\bowtie_\theta$

Example

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

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

=

$R \bowtie_{A<D} S$	A	R.B	R.C	S.B	S.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

$R \bowtie_{A<D \text{ AND } R.B \neq S.B} S$	A	R.B	R.C	S.B	S.C	D
	1	2	3	7	8	10

Other derived operations: Theta-join $\bowtie_\theta$

- Take the product of R and S
- Select from the product only those tuples that satisfy the condition Θ
- Schema of $R \bowtie_\theta S$ is the union of the (modified) schemas of R and S, after renaming the repeated attributes, say B in R to R.B, and B in S to S.B, as in the previous slide

(This rule for schema is the same as for Product's schema)

More derived operations

Theta-join	$R \bowtie_\theta S = \sigma_\theta(R \times S)$
Equijoin	$\bowtie_\theta$ where θ is a **conjunction of equalities**
Semijoin	$R \ltimes_\theta S = \pi_X\big(R \bowtie_\theta S\big)$ where X is the set of attributes of R
Antijoin	$R \,\overline{\ltimes}_\theta\, S = R - (R \ltimes_\theta S)$

Why use derived operations?

- to write things more succinctly
- they can be optimized independently

Joins

Inner Join

Includes only those tuples with matching attributes and the rest are discarded in the resulting relation

- Natural Join
- Theta Join
- Equijoin

Outer Join

Includes unmatched tuples from the participating relations

- Full outer join
- Left outer join
- Right outer join

(Semijoin and Antijoin are separate, not in these categories)

Outer Join

- The outer join (full outer join) $R \bowtie_{\theta}^{\text{outer}} S$

- Schema contains union of the (possibly renamed) attributes in R and S

- Tuples consist of
  - The tuples in the regular join
  - The tuples of R that do not join with any tuple in S, and padded with NULL instead
  - The tuples of S that do not join with any tuple in R, and padded with NULL instead

Recall that those tuples that are not joined are called dangling tuples

Left Outer Join

- The outer join (full outer join) $R \bowtie_{\theta}^{\text{left}} S$

- Schema contains union of the (possibly renamed) attributes in R and S

- Tuples consist of
  - The tuples in the regular join
  - The tuples of R that do not join with any tuple in S (dangling tuples), and padded with NULL instead

Right Outer Join

- The outer join (full outer join) $R \bowtie_{\theta}^{\text{right}} S$
- Schema contains union of the (possibly renamed) attributes in R and S
- Tuples consist of
  - The tuples in the regular join
  - The tuples of S that do not join with any tuple in R (dangling tuples), and padded with NULL instead

Outer Join: Example

Example

R	**A**	**B**
	1	2
	3	4

S	**B**	**C**	**D**
	2	3	4
	5	6	7

$R \bowtie_{R.B=S.B}^{\text{outer}} S$	**A**	**R.B**	**S.B**	**C**	**D**
	1	2	2	3	4
	2	4	NULL	NULL	NULL
	NULL	NULL	5	6	7

All dangling tuples are padded with NULLs

Left Outer Join: Example

Example

R	A	B
	1	2
	3	4

S	B	C	D
	2	3	4
	5	6	7

$R \bowtie^{\text{left}}_{R.B=S.B} S$	A	R.B	S.B	C	D
	1	2	2	3	4
	2	4	NULL	NULL	NULL
	NULL	NULL	5	6	7

Left dangling tuples are padded with NULLs

Right Outer Join: Example

Example

R	A	B
	1	2
	3	4

S	B	C	D
	2	3	4
	5	6	7

$R \bowtie^{\text{right}}_{R.B=S.B} S$	A	R.B	S.B	C	D
	1	2	2	3	4
	NULL	NULL	5	6	7

Right dangling tuples are padded with NULLs

Another Derived Operation: Division (Example)

Find the names of students who have taken exams in all CS courses

Exams

Student	Course
John	Databases
John	Chemistry
Mary	Programming
Mary	Math
Mary	Databases

CS

CourseName
Databases
Programming

Exams $\div$ CS $=$

Student
Mary

$$= \pi_{\textbf{Student}}(\text{Exams}) - \pi_{\textbf{Student}}\big((\pi_{\textbf{Student}}(\text{Exams}) \times \text{CS}) - \text{Exams}\big)$$

Division

R over set of attributes X

S over set of attributes $Y \subset X$

Let $Z = X - Y$

$$R \div S = \{\ \bar{r} \in \pi_Z(R) \mid \forall \bar{s} \in S\ (\bar{r}\bar{s} \in R\ )\}$$

$$= \{\ \bar{r} \in \pi_Z(R) \mid \{\bar{r}\} \times S \subseteq R\ \}$$

$$= \pi_Z(R) - \pi_Z\big(\pi_Z(R) \times S - R\big)$$

Relational Algebra Expression

Definition: A relational algebra **expression** is a string obtained from relation schemas using union, difference, cartesian product, projection, selection and renaming.

Context-free grammar for relational algebra expressions:

$$E := R, S, \ldots \mid (E_1 \cup E_2) \mid (E_1 - E_2) \mid (E_1 \times E_2) \mid \pi_L(E) \mid \sigma_\Theta(E) \mid \rho\, E,$$

where
$R, S, \ldots$ are relation schemas
L is a list of attributes
Θ is a condition.

Strength from Unity and Combination

By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.

When used in combination, however, the basic relational algebra operations can express interesting and, quite often, rather complex queries.

Independence of the Basic Relational Algebra Operations

Question: Are all the basic relational algebra operations really needed? Can one of them be expressed in terms of the other five?

Theorem: Each of the basic relational algebra operations is independent of the other five, that is, it **cannot** be expressed by a relational algebra expression that involves only the other five.

Proof Idea: For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other five operations.

Independence of the Basic Relational Algebra Operations

Proof Sketch: (projection and cartesian product only)

Property of projection: It is the only operation whose output may have arity smaller than its input.

Show, by induction, that the output of every relational algebra expression in the other five basic relational algebra is of arity at least as big as the maximum arity of its arguments.

Property of cartesian product: It is the only operation whose output has arity bigger than its inputs.

Show, by induction, that the output of every relational algebra expression in the other five basic relational algebra is of arity at most as big as the maximum arity of its arguments.

Exercise: Complete this proof.

Relational Algebra: Summary

When combined with each other, the basic relational algebra operations can express interesting and complex queries (natural join, quotient (division), ...)

The basic relational algebra operations are independent of each other: none can be expressed in terms of the other.

So, in conclusion, Codd's choice of the basic relational algebra operations has been very judicious.

Relational Completeness

Definition (Codd – 1972): A database query language L is **relationally complete** if it is at least as expressive as relational algebra, i.e., every relational algebra expression E has an equivalent expression F in L.

Relational completeness provides a benchmark for the expressive power of a database query language.

Every commercial database query language should be at least as expressive as relational algebra.

Exercise: Explain why SQL is relationally complete

SQL vs. Relational Algebra

SQL	**Relational Algebra**
SELECT	Projection π
FROM	Cartesian Product $\times$
WHERE	Selection σ

Semantics of SQL via interpretation to Relational Algebra

SELECT $R_{i_1}.A_1, \ldots, R_{i_m}.A_m$
FROM $R_1, \ldots, R_k$
WHERE Ψ

$$= \pi R_{i_1}.A_1, \ldots, R_{i_m}.A_m(\sigma \Psi(R_1 \times \cdots \times R_k))$$

Additional Operators

There are additional relational algebra operators

- Usually used in the context of query optimization

Duplicate elimination – ϵ (or δ)

- Used to turn a bag (multiset) into a set

Aggregation operators

- e.g. sum, average

Grouping – γ

- Used to partition tuples into groups
- Typically used with aggregation

Combining Operators to Form Queries

Expressions by applying operations to the result of other operations
- Parentheses and precedence rules

Three notations
- Expression trees
- Expressions with several operators
- Sequences of assignment statements

Expressions

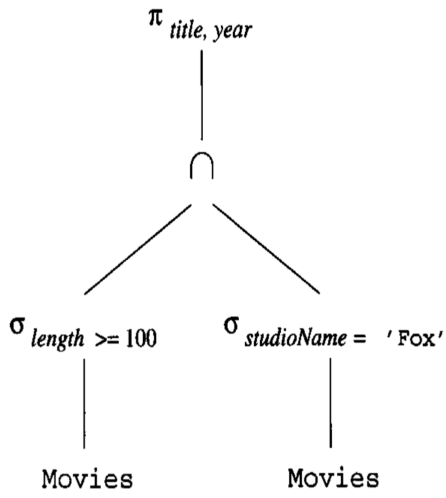
Example
What are the titles and years of movies made by Fox that are at least 100 minutes long?

1. Select those Movies tuples that have $length >= 100$
2. Select those Movies tuples that have $studioName = `Fox'$
3. Compute the intersection of (1) and (2)
4. Project the relation from (3) onto attributes title and year

Tree Expression

- Evaluated bottom-up
- Applying the operator at an interior node to the arguments, which are the results of its children

- The two selection nodes: steps (1) and (2)
- The intersection node: step (3)
- The projection node is step (4)



Linear Expression

More than one relational algebra expression that represents the same computation

$$\pi_{title,year}(\sigma_{lengh \geq 100}(Movies) \cap \sigma_{studioName=\text{'}Fox\text{'}}(Movies))$$

$$\pi_{title,year}(\sigma_{lengh \geq 100 \text{ AND } studioName=\text{'}Fox\text{'}}(Movies))$$

Linear Notation

- Names for temporary relations corresponding to interior nodes of the tree

- Sequence of assignments that create value for each temporary relation

- Flexible order as long as value is ready

Linear Notation

- A relation name and parenthesized list of attributes for that relation

The name answer will be used conventionally for the result of the final step

- i.e. the name of the relation at the root of the expression tree

- The assignment symbol ($:=$)

- Any algebraic expression on the right

Linear Notation

$$R(t, y, l, i, s, p) := \sigma_{lengh \geq 100}(Movies)$$

$$S(t, y, l, i, s, p) := \sigma_{studioName=`Fox'}(Movies)$$

$$Answer(title, year) := \pi_{t,y}(R \cap S)$$

Equivalent Expressions & Query Optimization

- Queries are based on expressions similar to relational algebra
- Query asked may have many equivalent expressions
- Some are much more quickly evaluated
- Query optimization: replace one expression of relational algebra by an equivalent expression that is more efficiently evaluated

Summary

- Relational Algebra operators
  - Five core operators: selection, projection, cross-product, union and set difference (plus renaming)
  - Additional operators are defined in terms of the core operators E.g. Intersection, join
- SQL and Relational Algebra can express the same class of queries
- Multiple Relational Algebra queries can be equivalent
  - Same semantics but different performance
  - Form basis for optimizations

Acknowledgements

[1] Database Systems: The Complete Book, 2nd EditionHector Garcia-Molina, Jeffrey D. Ullman, Jennifer WidomPrentice Hall, 2009

[2] Database System Concepts, Seventh EditionAvi Silberschatz, Henry F. Korth, S. SudarshanMcGraw-Hill, March 2019www.db-book.com

Additional references and resources used in preparation of this course are listed on the course webpage or mentioned in slides.