CMPUT391

Brief review of the Relational Algebra

Instructor: Denilson Barbosa

University of Alberta

October 10, 2023

Slides by D. Barbosa, with suggested corrections and improvements by (in alphabetical order) C. Bins, D. Caminhas, K. Guhzva, Q. Lautischer, E. Macdonald, M. A. Nascimento, K. Newbury, M. Strobl, D. Sunderman, K. Wang, and K. Wong.

The Relational Model of Data

Relational databases were introduced in the 1970's, became prevalent in the 1980's and remain the industry standard for applications that manage large volumes of data

In CMPUT291/391 we study two relational query languages:
- the algebra, which is clean and grounded on solid mathematical foundations
- SQL, which is a very complex language with many nuances and intricacies introduced to make programmers lives easier

The algebra and SQL are good friends

While the algebra is procedural and SQL is declarative they are "largely equivalent". In fact, most DBMSs translate SQL queries into algebraic expressions and evaluate them that way.

Relation

A relation is a **collection of** similar **facts** relevant to the application.
- **schema**: relation name, attribute types, constraints;
- **instance**: tuples.

In the context of the algebra, we ignore attribute types and constraints

Movie

title	year	imdb	director
Ghostbusters	1984	7.8	Ivan Reitman
Big	1988	7.3	Penny Marshall
Lost in Translation	2003	7.8	Sofia Coppola
Wadjda	2012	8.1	Haifaa al-Mansour
Ghostbusters	2016	5.3	Paul Feig

Relational Database

A database is a **collection of** relations, each with its schema and constraints.
- **schema**: the schemata of the relations *plus* (optionally) inter-relation constraints;
- **instance**: the instances of the relations.

Legal database instance

A database instance is **legal** if it satisfies all constraints in the schema.

Constraint

Logical condition over one or more tuples, derived from the application, that must be true at all times.

The relational algebra is taught as a theoretical tool that helps one understand and prove statements about relational databases and more complex languages such as SQL.

We thus ignore practical issues such as data types and constraints when talking about the algebra.

Movie

title	year	imdb	director
Ghostbusters	1984	7.8	Ivan Reitman
Big	1988	7.3	Penny Marshall
Lost in Translation	2003	7.8	Sofia Coppola
Wadjda	2012	8.1	Haifaa al-Mansour
Ghostbusters	2016	5.3	Paul Feig

Cast

title	year	actor	role
Ghostbusters	1984	Bill Murray	Dr. Venkman
Ghostbusters	1984	Sigourney Weaver	Dana Barret
Big	1988	Tom Hanks	Josh
Big	1988	Elisabeth Perkins	Susan
Lost in Translation	2003	Bill Murray	Bob Harris
Wadjda	2012	Waad Mohammed	Wadjda
Ghostbusters	2016	Dan Aykroyd	Cabbie
Ghostbusters	2016	Sigourney Weaver	Dana Barret

Cinema

name	address
Garneau	8712 109 St, Edmonton
Princess	10337 82 Ave, Edmonton
Landmark	10200 102 Ave, Edmonton

Guide

theater	film	year	start
Garneau	Ghostbusters	1984	1140
Garneau	Ghostbusters	2016	1290
Princess	Wadjda	2012	1260

The (Standard) Relational Algebra

The relational model is defined on solid mathematical grounds:
- relations are sets (or multisets) of tuples
- query languages manipulate tuples or sets of tuples

The Relational Algebra is an operator-based, procedural language that allows us to manipulate relational data.

Algebra operators result in new relations

Every operator in the algebra takes one or two relations as input and **produces a new relation** as output.

This is what allows operators to be combined to form complex queries.

The (horizontal) **selection** operator $\sigma_c(R)$:
- creates a new relation with all tuples from R that satisfy the boolean condition c
- the **schema** of the resulting relation is the same as R

title	year	actor	role
Ghostbusters	1984	Bill Murray	Dr. Venkman
Lost in Translation	2003	Bill Murray	Bob Harris

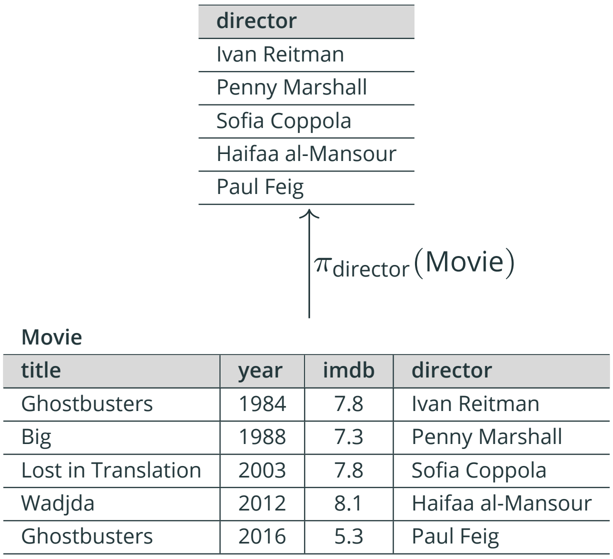
$\uparrow$ $\sigma_{\text{actor='Bill Murray'}}(\text{Cast})$

Cast

title	year	actor	role
Ghostbusters	1984	Bill Murray	Dr. Venkman
Ghostbusters	1984	Sigourney Weaver	Dana Barret
Big	1988	Tom Hanks	Josh
Big	1988	Elisabeth Perkins	Susan
Lost in Translation	2003	Bill Murray	Bob Harris
Wadjda	2012	Waad Mohammed	Wadjda
Ghostbusters	2016	Dan Aykroyd	Cabbie
Ghostbusters	2016	Sigourney Weaver	Dana Barret

The **projection** operator $\pi_{a_1,\ldots,a_n}(R)$:

- creates a new relation with attributes $a_1, \ldots, a_n$ of tuples from R
- the **schema** of the resulting relation is $a_1, \ldots, a_n$



director
Ivan Reitman
Penny Marshall
Sofia Coppola
Haifaa al-Mansour
Paul Feig

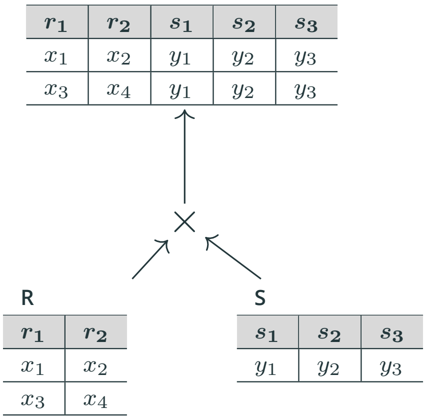
$\pi_{\text{director}}(\text{Movie})$

Movie

title	year	imdb	director
Ghostbusters	1984	7.8	Ivan Reitman
Big	1988	7.3	Penny Marshall
Lost in Translation	2003	7.8	Sofia Coppola
Wadjda	2012	8.1	Haifaa al-Mansour
Ghostbusters	2016	5.3	Paul Feig

The **cross product** operator $R \times S$:

- creates a new relation with the concatenation of every tuple in R with every tuple in S
- the **schema** of the resulting relation has every attribute in R followed by every attribute in S



r_1	r_2	s_1	s_2	s_3
x_1	x_2	y_1	y_2	y_3
x_3	x_4	y_1	y_2	y_3

$\times$

R

r_1	r_2
x_1	x_2
x_3	x_4

S

s_1	s_2	s_3
y_1	y_2	y_3

The **join** of two relations $R \bowtie_c S$ concatenates *matching* tuples from R and S and is defined as follows:

$$R \bowtie_c S \equiv \sigma_c (R \times S)$$

Equi-joins and Natural Joins

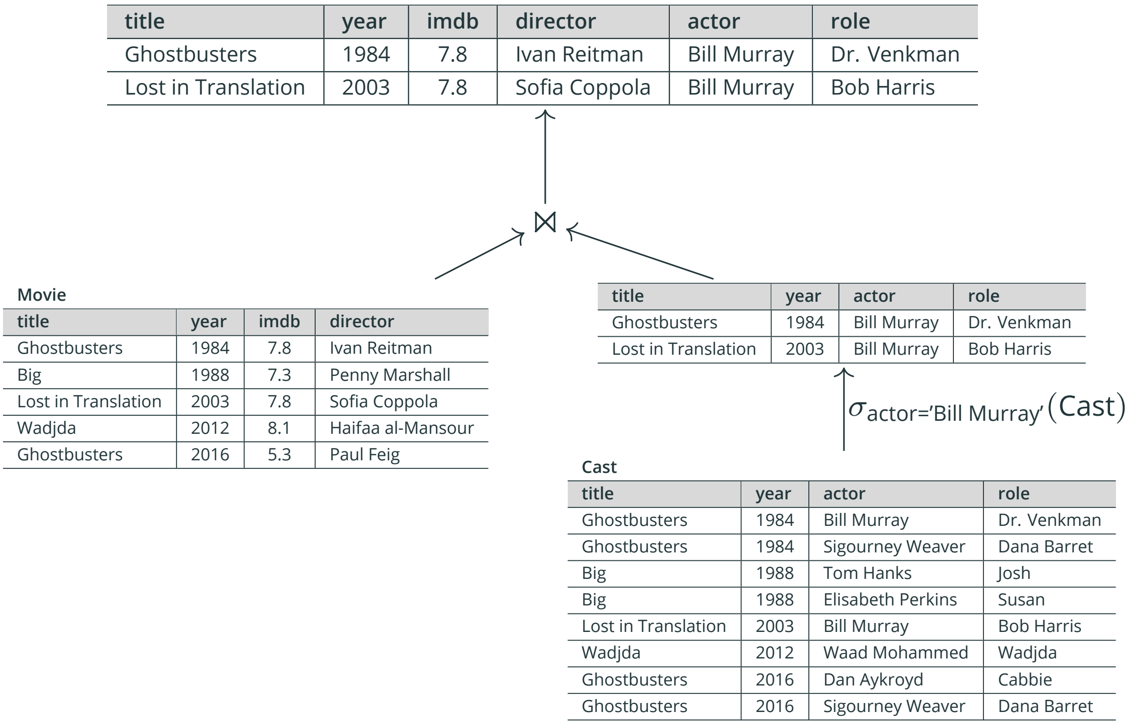
The join condition c normally involves attributes from both relations.

An **equijoin** is a join where all conditions are equalities.

The **natural join**[1] of R and S written just $R \bowtie S$ is an equijoin defined over all attributes with the same name in R and S.

[1] Some textbooks refer to the natural join simply as a “join”, calling the join with a condition a theta-join: $\bowtie_\theta$

Example: Movie $\bowtie$ $\left(\sigma_{\text{actor='Bill Murray'}}(\text{Cast})\right)$



title	year	imdb	director	actor	role
Ghostbusters	1984	7.8	Ivan Reitman	Bill Murray	Dr. Venkman
Lost in Translation	2003	7.8	Sofia Coppola	Bill Murray	Bob Harris

$\bowtie$

Movie

title	year	imdb	director
Ghostbusters	1984	7.8	Ivan Reitman
Big	1988	7.3	Penny Marshall
Lost in Translation	2003	7.8	Sofia Coppola
Wadjda	2012	8.1	Haifaa al-Mansour
Ghostbusters	2016	5.3	Paul Feig

title	year	actor	role
Ghostbusters	1984	Bill Murray	Dr. Venkman
Lost in Translation	2003	Bill Murray	Bob Harris

$\sigma_{\text{actor='Bill Murray'}}(\text{Cast})$

Cast

title	year	actor	role
Ghostbusters	1984	Bill Murray	Dr. Venkman
Ghostbusters	1984	Sigourney Weaver	Dana Barret
Big	1988	Tom Hanks	Josh
Big	1988	Elisabeth Perkins	Susan
Lost in Translation	2003	Bill Murray	Bob Harris
Wadjda	2012	Waad Mohammed	Wadjda
Ghostbusters	2016	Dan Aykroyd	Cabbie
Ghostbusters	2016	Sigourney Weaver	Dana Barret

The **schema** of the relation resulting from a join has all attributes in the incoming relations.

For convenience, in these slides, we omit repeated attributes for the sake of clarity, as in:

$$\text{Movie} \bowtie \left(\sigma_{\text{actor='Bill Murray'}}(\text{Cast})\right)$$

title	year	imdb	director	actor	role
Ghostbusters	1984	7.8	Ivan Reitman	Bill Murray	Dr. Venkman
Lost in Translation	2003	7.8	Sofia Coppola	Bill Murray	Bob Harris

Different DBMSs handle these situations differently. Try an example like this on SQLite!

The join operator ignores all tuples from either relation that do not match with any tuple in the other relation. To keep non-matching tuples we can use an **outerjoin**.[2]

The **left outerjoin** $R ⟕ S$ keeps unmatched tuples in R with NULL values for the missing tuple in S.

Example: Cinema $⟕_{\text{name=theater}}$ Guide:

name	address	theater	film	year	start
Garneau	8712 109 St, Edmonton	Garneau	Ghostbusters	1984	1140
Garneau	8712 109 St, Edmonton	Garneau	Ghostbusters	2016	1290
Princess	10337 82 Ave, Edmonton	Princess	Wadjda	2012	1260
Landmark	10200 102 Ave, Edmonton	NULL	NULL	NULL	NULL

The **right outerjoin** $R ⟖ S$ and the **full outerjoin** $R ⟗ S$ are defined analogously.

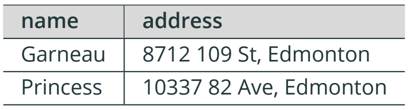
[2] The Garcia-Molina, Ullman, Widom book uses a different notation: $R \overset{\circ}{\bowtie} S$

Another variant of the join operator is the **semijoin**.

The **left semijoin** $R \ltimes_c S$ produces a relation with tuples from R that have a matching tuple in S.

Example[3]:

Cinema $\ltimes_{\text{name=theater}}$ Guide



name	address
Garneau	8712 109 St, Edmonton
Princess	10337 82 Ave, Edmonton

The **right semijoin** $R \rtimes_c S$ is defined analogously.

[3] Note that the result does not have duplicates, even though there are two matches in Guide for the Garneu tuple in Cinema.

Set operations

In the standard relational model and algebra, **relations are sets** and thus one can use the other set operators (besides cross product) on them.

In this case, however, we require the relations to be *compatible* (i.e., have the same schema).

- $R \cup S$: all tuples in R or S (or both)

- $R \cap S$: all tuples common to R and S

- $R - S$: all tuples in R but not in S

Renaming

Consider finding actors who starred in a movie and a remake of the movie. From slide 6 we gather actress Sigourney Weaver would be an answer, as she acted on the original Ghostbusters of 1984 and the remake in 2016[4]

The query we seek has to concatenate *two tuples from Cast* (one for the original move, and the other for the remake).

We can find the concatenations with the Cartesian product of Cast with *itself*, but, we need to **rename** the attributes in one of the copies to express a selection condition that the years are different

title	year	actor	role	title2	year2	actor2	role2
Ghostbusters	1984	Sigourney Weaver	Dana Barret	Ghostbusters	2016	Sigourney Weaver	Dana Barret

[4] Although missing from the example, Dan Aykroyd would be another answer.

The renaming operator $\rho_{S(b_1,\ldots,b_n)}(R)$ creates a "copy" of a relation $R(a_1, \ldots, a_n)$ that has a different schema but the same instance.

Thus, the query we are looking for is

$$\pi_{\text{actor}}\left(\text{Cast} \bowtie_{\substack{\text{title=title2} \\ \wedge\ \text{actor=actor2} \\ \wedge\ \text{year} \neq \text{year2}}} \rho_{\text{Cast2(title2,year2,actor2,role2)}}(\text{Cast})\right)$$

The $\wedge$ in the join condition stands for the conjunction (or logical "and") of two Boolean expression.

Conjunctive queries

A query is said to be conjunctive if it uses only σ, π, ρ and $\bowtie$ operators and all selection/join conditions c involve only single predicates or conjunctions of predicates.

Positive queries

A query is said to be positive if it does not use the set difference (-) operator.

Positive queries are monotonic

A monotonic query is such that *inserting* new tuples in the database cannot *reduce* the number of tuples in the answer to the query.

Rewriting algebra expressions

Conjunctive predicates can be rewritten as the intersection of queries... and vice-versa!

Example: movies with Sigourney Weaver *and* from year 1984:

$$\pi_{\text{title,year}}\left(\sigma_{\text{actor='Sigourney Weaver'} \wedge \text{ year}=1984}\left(\text{Cast}\right)\right)$$

is equivalent to

$$\pi_{\text{title,year}}\left(\sigma_{\text{actor='Sigourney Weaver'}}(\text{Cast}) \cap \sigma_{\text{year}=1984}(\text{Cast})\right)$$

This property is actually useful for a DBMS developer, as it allows the query optimizer to modify the original query provided by the user in a way to achieve better performance.

Disjunctive predicates can be rewritten as the union of queries... and vice-versa!

Example: movies with Sigourney Weaver *or* Bill Murray:

$$\pi_{\text{title,year}}\left(\sigma_{\text{actor='Sigourney Weaver'} \vee \text{ actor='Bill Murray'}}\left(\text{Cast}\right)\right)$$

is equivalent to

$$\pi_{\text{title,year}}\left(\sigma_{\text{actor='Sigourney Weaver'}}(\text{Cast}) \cup \sigma_{\text{actor='Bill Murray'}}(\text{Cast})\right)$$

When we say $\quad Q_1 : \quad \sigma_{a_1=x_1 \wedge \cdots \wedge a_n=x_n}(R)$

is equivalent to $\quad Q_2 : \quad \sigma_{a_1=x_1}(R) \cap \cdots \cap \sigma_{a_n=x_n}(R)$

We mean that, for any instance of R, every tuple t in the result of Q_1 **must also** be in the result of Q_2

Statements like this are useless unless **proven** correct.[5]

Sample proof by *contradiction*. Assume there is an instance of R for which there is a tuple t that is in the answer of Q_1 but not in the answer of Q_2. However, if t is in the answer of Q_1 it must satisfy every predicate $a_i = x_i$; meaning it must be in the answer of Q_2. This contradiction proves the queries *are* equivalent.

[5] Although CMPUT391 does not cover *database theory*, the good student should understand and try to prove equivalences mentioned in the course.

Other equivalence-based rewrite rules

Many algebra operators are both associative and commutative[6]:
- $R \times S = S \times R$; $\quad (R \times S) \times T = R \times (S \times T)$
- $R \bowtie S = S \bowtie R$; $\quad (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $R \cap S = S \cap R$; $\quad (R \cap S) \cap T = R \cap (S \cap T)$
- $R \cup S = S \cup R$; $\quad (R \cup S) \cup T = R \cup (S \cup T)$

The standard join is commutative $R \bowtie_c S = S \bowtie_c R$ but generally not associative (as the attributes in the condition c may not exist in all relations).

[6] Note that there is no “order” for the attribute names in a relation schema

More equivalences involving selections

Cascading selections

$$\sigma_{c_1 \wedge c_2}(R) \ = \ \sigma_{c_1}(\sigma_{c_2}(R)) \ = \ \sigma_{c_2}(\sigma_{c_1}(R))$$

“Pushing down” through union

$$\sigma_c(R \cup S) \ = \ \sigma_c(R) \cup \sigma_c(S)$$

“Pushing down” through intersection

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

“Pushing down” through set difference

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

Pushing down selections through joins

The general rule is to push the selection condition “down” to the relation participating in the join that contains the attributes in the selection condition. If both relations have the attributes, the selection can be pushed both ways.

This rule applies to Cartesian products as well:

$$
\begin{aligned}
\sigma_c(R \bowtie S) &= \sigma_{c^R}(R) \bowtie \sigma_{c^S} S \\
\sigma_c(R \bowtie_\theta S) &= \sigma_{c^R}(R) \bowtie_\theta \sigma_{c^S} S \\
\sigma_c(R \times S) &= \sigma_{c^R}(R) \times \sigma_{c^S} S
\end{aligned}
$$

c^R are the conditions involving only attributes of R; similarly for c^S.

There aren’t as many useful equivalence rules for **projections** as there are for selections, however.

The main reason is that whenever we “push” a projection down, we must bring along all attributes used in other operators. For joins, we can rewrite

$$\pi_{\mathbf{a}}(R \bowtie_{\mathbf{b}} S) = \pi_{\mathbf{a}}\left(\pi_{\mathbf{c}}(R) \bowtie_{\mathbf{b}} \pi_{\mathbf{d}}(S)\right)$$

Where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{d}$ are **sets of attributes**.

Note that the expressions above are equivalent only if both $\mathbf{c}$ and $\mathbf{d}$ contain **all** attributes in both $\mathbf{a}$ and $\mathbf{b}$.

Also, the outermost projection on $\mathbf{a}$ must remain unless $\mathbf{b}, \mathbf{c}$, and $\mathbf{d}$ form a partition $\mathbf{a}$.

The Algebra is a procedural language

Expressions in the relational algebra can be parsed (obviously and unambiguously) into trees, where the leaf nodes contain base tables and inner nodes contain algebra operators.

This gives an immediate algorithm for computing the result of queries: starting from the root, perform a (depth-first) traversal of the tree computing the relation resulting from the expression in every node in the tree, computing the inputs recursively.

The simple procedure is the basis of the query execution engine in most relational DBMSs out there!