IMDb (Part 2)
Relational Algebra

Big Data Engineering
(formerly Informationssysteme)

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

May 2, 2023

IMDb (Part 2)

1. Concrete application: IMDb

- We looked at this last time.

IMDb (Part 2)

2. What are the data management and analysis issues behind this?

last week:

Question 1

How is the data on films, actors, directors etc. modelled and stored in IMDb?

Question 2

How are links between these data modelled and stored in IMDb?

this week:

Question 3

How do we query this data?

...

IMDb (Part 2)

3. Basics to be able to solve these problems
    (a) Slides
    (b) Jupyter/Python/SQL Hands-on

- The relational algebra

Main Learning Objectives

Relational Algebra

Operators (unary and binary), Selection (not to be confused with SELECT in SQL), Projection, Union, Difference, Cross product, Intersection, Theta Join, Group by vs Aggregation, Co-Grouping

Relational Algebra (RA)

- modular query language for converting one or more input relations into an output relation
- comes in different variants/dialects
- but the core idea is always the same
- can be extended very easily
- there are many different implementations of relational algebra, currently the most well-known is Apache Spark
- RA is used internally by systems for query optimisation, e.g. in relational database systems
- or: RA is used directly by users to specify a data flow program, e.g. in Apache Spark

Relational Algebra: Operators

Operators

RA consists of **operators**. An operator transforms one or two input relations into one output relation, i.e. operators come in two flavors:

unary: op: $\mathcal{P}(R) \to \mathcal{P}(R)$

binary: op: $\mathcal{P}(R) \times \mathcal{P}(R) \to \mathcal{P}(R)$

Here, $\mathcal{P}(R)$ designates the set of all possible relations. For historical reasons, only unary and binary operators are allowed.
(This is actually an unnecessary restriction.)

Caution

Relational algebra only abstractly describes **WHAT** is to be calculated, but not **HOW** this calculation is implemented algorithmically!

Significance for data analysis and processing:

Comparable with the periodic table of elements in chemistry

Notes on Domains

We had already introduced domains (value ranges) for entity-relationship modelling, see IMDb (Part 1):

The following applies:

Domain (value range)

A domain is a set **atomic** values. These values must not be structured (i.e. further divisible). Domains are notated as D, e.g. integer, float, string, etc.

Careful

This restriction is outdated and leads to a lot of confusion, e.g. the so-called **first normal form**. In practice, this restriction is usually weakened. We will come back to this when we discuss (modern) SQL-99.

The Periodic Table of Relational Algebra Operators

Symbol	Name		Classification	
	German	English	Main operator class	unary/binary
σ	Selektion	selection	basic	unary
π	Projektion	projection		unary
$\cup$	Vereinigung	union		binary
$-$	Differenz	minus		binary
$\times$	kartesisches Produkt (oder Kreuzprodukt)	cross product		binary
ρ	Umbenennung	rename		unary
$\cap$	Schnitt	intersection	derived	binary
$\bowtie$	Verbund	join		
$\bowtie_\theta$	Theta-Verbund	theta join		
$\bowtie_{[L],[R]}$	Equi-Verbund	equi join		
$\ltimes$	Linker Pseudo-Verbund	left semi join		
$\rtimes$	Rechter Pseudo-Verbund	right semi join		
$\triangleright$	Linker Anti-Pseudo Verbund	left anti semi join		
$\triangleleft$	Rechter Anti-Pseudo Verbund	right anti semi join		
$⟕$	Linker äußerer Verbund	left outer join		
$⟖$	Rechter äußerer Verbund	right outer join		
$⟗$	Äußerer Verbund	full outer join		
γ	Gruppierung (mit Aggregation)	grouping (group by)	extensions	unary
Γ	Co-Gruppierung (**ohne** Aggregation)	co-grouping		binary
	...			

A Note on the Kemper Notation

Since it is usually clear from the context whether R denotes the relation $R = (\{R\}, [R])$, i.e. the instance and the schema definition, or just the instance, we often write R instead of $\{R\}$ in the following.

Example (see Notebook "Relational Algebra")

In [11]:
```
exp2 = Projection_ScanBased(newmovies, ['id', 'year'])
```

In [12]:
```
print(exp2)
```

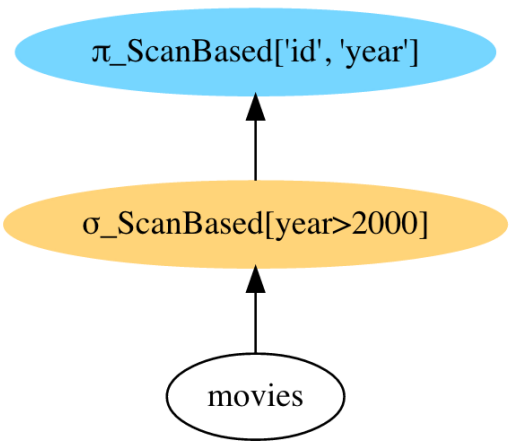
```
π_ScanBased_['id', 'year'](σ_ScanBased_[year>2000](movies))
```

In [13]:
```
exp2.evaluate().print_set()
```

```
[Result] : {[ id:int, year:int ]}
{
        (96779, 2001),
        (393538, 2003),
        (176712, 2004),
        (176711, 2003),
        (127297, 2003),
        (159665, 2006),
        (105938, 2002),
        (10934, 2005)
}
```

In [14]:
```
graph = exp2.get_graph()
Source(graph)
```

Out[14]:



github: Relational Algebra.ipynb

Operations on Relational Schemas

Since relational schemas are sequences (with unique attribute names) rather than sets, we need definitions to make statements about them.

Operations on Relational Schemas

Attribute is included:
$A \in \{[A_1, \ldots, A_n]\} \Leftrightarrow$ there is a j with $1 \leq j \leq n$ and $A_j = A$.

Subschema:
$[R] \subseteq [S], \Leftrightarrow$ for all $A \in [R]$ holds $A \in [S]$.

Intersection of two schemas (common attributes):
$[R] \cap [S] = [T]$ with $A \in [T] \Leftrightarrow A \in [R] \wedge A \in [S]$.

Union of two schemas:
$[R] \cup [S] = [T]$ with $A \in [T] \Leftrightarrow A \in [R] \vee A \in [S]$.

Concatenated schema:
$[R] \circ [S] = [R] \cup [S]$ if $[R] \cap [S] = \varnothing$, otherwise undefined

Equality of schemas:
$[R] == [S] \Leftrightarrow [R] \subseteq [S] \wedge [S] \subseteq [R]$

Cardinality of a schema:
$\left|[R]\right| := n$ for $[R] = \{[A_1, \ldots, A_n]\}$.

Basic Operators: Selection σ

The following operators are called the **basic operators** of relational algebra. Many other operators build on these or are *syntactic sugar*.

Selection (filter) σ

The selection operator selects a subset $R' \subseteq R$ from the input relation R based on a predicate $P : [R] \rightarrow bool$. The attributes used in predicate P must be contained in $[R]$, i.e. $A_i \in [R] \; \forall \; A_i \in P$.
$R' = \sigma_P(R) = \{r \in R \mid P(r)\}$.

Examples: (keys in schema omitted)

$[R_1] : \{[\text{id:int}]\}$, $R_1 = \{(1), (2)\}$,
$[R_2] : \{[\text{id:int}]\}$, $R_2 = \{(2), (3), (4)\}$

P:= id ≤ 2

$\sigma_P(R_1) = \{(1), (2)\}$, $\sigma_P(R_2) = \{(2)\}$

directors		
id	**first_name**	**last_name**
78273	Quentin	Tarantino
43095	Stanley	Kubrick
11652	James (I)	Cameron

$\sigma_{id>45000}$(directors):

id	first_name	last_name
78273	Quentin	Tarantino

$\sigma_{first\_name='Stanley'}$(directors):

id	first_name	last_name
43095	Stanley	Kubrick

Predicates and Atomic Predicates

Predicates

A predicate is a function $P : [R] \rightarrow bool$. It is well-defined on the schema $[R]$ if all attributes used in P are defined in $[R]$ with appropriate domains.

Atomic Predicates

Atomic predicates are of the form $A_1 == A_2$ or $A_1 == c$, where A_1, A_2 are attributes and c is a constant. The atomic predicates can be linked with boolean operators. The following operators exist for specific domains: $\leq, \geq, <, >$ (int, float) and is_substr (string). These represent comparisons and the substring property.

Examples:

[R] : {[id:int, a:int, b:int]}, $P_i : [R] \rightarrow bool$

P_1:= id $\leq$ 42 (semantically not meaningful as a range of symbols does not make sense)
P_2:= a == 42 (aka **point query**, i.e. we query for one point in the domain of attribute a)
P_3:= b $\geq$ 42 (aka **range query**, i.e. we query for a range in the domain of attribute b)
P_4:= c == 42 (not allowed, as attribute c is not defined in [R])

Projection π

Projection π

The projection operator reduces the schema of input relation R to a subschema of that schema and returns the projected tuples.
Let $[R'] \subseteq [R]$ be any non-empty subschema of the attributes of R. Then the result of the projection is $R' := \pi_{[R']}(R) := \{r_{[R']} \mid r \in R\}$.
$r_{[R']}$ is a tuple containing only the attributes defined in subschema $[R']$.

Syntactic sugar: $\pi_{A_1,\ldots,A_n}(R) = \pi_{\{[A_1:D_1,\ldots,A_n:D_n]\}}(R)$

Examples:

[R] : {[a:int, b:int]}, R = {(1,3), (2,4), (2,7), (3,3), (4,2)}

subschema: $[R'] = \{[a:int]\}$

$\pi_{[R']}(R) = \pi_a(R) = \{ (1), (2), (3), (4) \}$

subschema: $[R''] = \{[b:int]\}$

$\pi_{[R'']}(R) = \pi_b(R) = \{ (3), (4), (7), (2) \}$

Cardinality of an Operator, Projection and Duplicates

Cardinality of a Relation

Let R be a relation. Then, $|R| \mapsto \text{int}$ is called the cardinality of R. It returns the number of tuples contained in R.

Projection and duplicates

It holds that $|\pi_{[R']}(R)| \leq |R|$.

Why?

Duplicates can arise through the projection. Since a relation is a set, that set does not represent those duplicates.

Example:

genre
Comedy
Mystery
Action
Romance
Fantasy
Horror
Music
Film-Noir
Sci-Fi
Drama
Short
Documentary
Adventure
Crime
War
Family
Thriller

$|\pi_{\text{genre}}(\text{movies\_genres})| = 17$

but:

$|\text{movies\_genres}| = 102$

Union ∪

Union ∪:

The union operator combines the tuples from two relations R_1 and R_2 to form a new relation R'. The following precondition must hold: $[R_1] == [R_2]$. Then the result of the union is $R' := R_1 \cup R_2$.
The number of tuples in R' is limited by $max(|R_1|, |R_2|) \leq |R'| \leq |R_1| + |R_2|$.

Example:

$[R_1]$: {[a:int]}, $R_1 = \{(1), (2)\}$,
$[R_2]$: {[a:int]}, $R_2 = \{(2), (3), (4)\}$

$[R_1] == [R_2]$

$R_1 \cup R_2 = \{(1), (2), (3), (4)\}$

cardinalities: $|R_1| = 2$, $|R_2| = 3$

$|R_1 \cup R_2| = 4 \leq (|R_1| + |R_2|)$.

Example:

both new as well as good films combined in a single relation

$\sigma_{year>2000}(movies) \cup \sigma_{rank \geq 7.5}(movies)$

id	name	year	rank
10934	Aliens of the Deep	2005	6.5
92616	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb	1964	8.7
105938	Expedition: Bismarck	2002	7.5
387728	ER	1994	7.7
96779	Earthship.TV	2001	5.6
393538	Jimmy Kimmel Live!	2003	6.7
267038	Pulp Fiction	1994	8.7
..	..	..	..

Difference –

Difference (minus/except) –

The difference operator removes the tuples of relation R_2 from relation R_1. The following precondition must hold: $[R_1] == [R_2]$. Then the result of the difference is $R' := R_1 - R_2$.
The number of tuples in R' is limited by $0 \leq |R'| \leq |R_1|$.

Examples:

$[R_1]$: {[a:int]}, $R_1 = \{(1), (2)\}$,
$[R_2]$: {[a:int]}, $R_2 = \{(2), (3), (4)\}$

$[R_1] == [R_2]$

$R_1 - R_2 = \{(1)\}$
$R_2 - R_1 = \{(3), (4)\}$

cardinalities: $|R_1| = 2$, $|R_2| = 3$

$|R_1 - R_2| = 1 \leq |R_1|$.
$|R_2 - R_1| = 2 \leq |R_2|$.

Example:

only new films that do not have a bad ranking in one relation; in other words: only new, good films

$\sigma_{\text{year}>2000}(\text{movies}) - \sigma_{\text{rank}<7.5}(\text{movies})$

id	name	year	rank
105938	Expedition: Bismarck	2002	7.5
176711	Kill Bill: Vol. 1	2003	8.4
176712	Kill Bill: Vol. 2	2004	8.2
159665	Inglorious Bastards	2006	8.3

Cross Product $\times$

Cross Product (cartesian product, cross join) $\times$

The cross product operator forms the cartesian product[1] of two relations, i.e. each tuple from the left input relation is combined with each tuple from the right input relation to form a new concatenated tuple.
If $[R_1] \cap [R_2] \neq \emptyset$, see uniqueness of attribute names. It holds: $[R'] = [R_1] \circ [R_2]$. The result of the cross product is $R' := R_1 \times R_2 = \{t_1 \circ t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$, where $\circ$ is the concatenation of two tuples. The number of tuples in the cross product is $|R'| = |R_1 \times R_2| = |R_1| \cdot |R_2|$.

Example:

$[R_1]$: {[a:int]}, $R_1 = \{(1), (2)\}$,
$[R_2]$: {[b:int]}, $R_2 = \{(2), (3), (4)\}$; $[R_1] \cap [R_2] = \emptyset$

$R_1 \times R_2 = \{ (1,2), (1,3), (1,4), (2,2), (2,3), (2,4)\}$

cardinalities: $|R_1| = 2$, $|R_2| = 3$, $|R_1 \times R_2| = |R_1| \cdot |R_2| = 2 \cdot 3 = 6$

[1] The terms cartesian and cross product are used synonymously for relational algebra.

Rename ρ

Rename ρ

The renaming operator changes the name of a relation **or** the name of an attribute of a relation.

1. For renaming a relation from R to R', it holds:
   $[R'] = [R], \{R'\} = \{R\}$. Then, $R' := \rho_{R'}(R) = \big(\{\text{R'}\}, [\text{R'}]\big)$.
2. Renaming an attribute $\text{A} \in [R]$ to A' yields a modified schema [R'] with the same domains and the same number of attributes:
   $R' := \rho_{\text{A}' \leftarrow \text{A}}(R)$.

Examples:

see Python Notebook Relational Algebra

The Problem with Duplicate Attribute Names

Example:
$[R_1]$: {[a:int]}, $R_1 = \{(1), (2)\}$,
$[R_2]$: {[b:int]}, $R_2 = \{(2), (3), (4)\}$

Concatenated schema:
$[R_1] \circ [R_2] = [R_1] \cup [R_2]$ if $[R_1] \cap [R_2] = \varnothing$, otherwise undefined,
i.e., in this case **undefined**

The following definition is **not correct**:
$[CP] := [R_1] \circ [R_2] = \{[\text{id:int}, \text{id}:\text{int}]\}$

$CP = R_1 \times R_2 = \{(1,2), (1,3), (1,4), (2,2), (2,3), (2,4)\}$

- How would we reference the attribute ‘id‘ in the result of the cross product?
- That reference would not be unique. And therefore this is not allowed!
- That’s why you have to rename the attributes first to make them unique across the input relations.

Derived Operators: Intersection ∩

All subsequent operators can be expressed by the basic operators and are consequently only "syntactic sugar".

Intersection ∩

The intersection forms the intersection of the tuple sets of two relations R_1 and R_2. The following precondition must hold: $[R_1] == [R_2]$. Then the result of the intersection is $R' := R_1 \cap R_2 = R_1 - (R_1 - R_2)$. The number of tuples in the intersection is $|R_1 \cap R_2| \leq \max\big(|R_1|, |R_2|\big)$.

Example:

$[R_1]$: {[a:int]}, $R_1 = \{(1), (2)\}$,
$[R_2]$: {[a:int]}, $R_2 = \{(2), (3), (4)\}$

$[R_1] == [R_2]$

$R_1 \cap R_2 = \{(2)\}$

cardinalities: $|R_1| = 2$, $|R_2| = 3$

$|R_1 \cap R_2| = 1 \leq \max\big(|R_1|, |R_2|\big)$.

Example:

only new good films in one relation:

$\sigma_{\text{year}>2000}(\text{movies}) \cap \sigma_{\text{rank}\geq 7.5}(\text{movies})$

id	name	year	rank
176711	Kill Bill: Vol. 1	2003	8.4
105938	Expedition: Bismarck	2002	7.5
176712	Kill Bill: Vol. 2	2004	8.2
159665	Inglorious Bastards	2006	8.3

Theta Join $\bowtie_\theta$

Theta Join $\bowtie_\theta$

The theta join forms the compound of two relations R_1 and R_2 under the general join predicate θ. From this, a new relation TJ is generated. It holds: $[TJ] = [R_1] \circ [R_2]$. $\theta : [TJ] \rightarrow$ bool.
$TJ := R_1 \bowtie_\theta R_2 = \sigma_\theta(R_1 \times R_2) \subseteq R_1 \times R_2$. The number of tuples in the join result is $0 \leq |TJ| \leq |R_1 \times R_2| = |R_1| \cdot |R_2|$.
If $[R_1] \cap [R_2] \neq \emptyset$, the same rules apply as for the uniqueness of attribute names in the cross product.

Example:

[R] : {[a:int, b:int]}, R = {(3,1), (4,2), (7,2), (3,3), (7,6)}
[S] : {[c:int, d:int]}, S ={(2,3), (1,4), (5,4), (3,8), (2,5)}

[TJ] = [R] ∪ [S] = {[a:int, b:int, c:int, d:int]}

$R \bowtie_{b=c} S = \{ (3,\underline{1},\underline{1},4), (4,\underline{2},\underline{2},3), (4,\underline{2},\underline{2},5), (7,\underline{2},\underline{2},3), (7,\underline{2},\underline{2},5), (3,\underline{3},\underline{3},8) \}$

$R \bowtie_{a=d} S = \{ (\underline{3},1,2,\underline{3}), (\underline{4},2,1,\underline{4}), (\underline{4},2,5,\underline{4}), (\underline{3},3,2,\underline{3}) \}$

Theta Join Example for IMDb

directors $\bowtie_{\text{id=director\_id}}$ movies_directors

id	first_name	last_name	director_id	movie_id
43095	Stanley	Kubrick	43095	176891
43095	Stanley	Kubrick	43095	177019
43095	Stanley	Kubrick	43095	250612
11652	James (I)	Cameron	11652	328277
43095	Stanley	Kubrick	43095	30431
43095	Stanley	Kubrick	43095	106666
78273	Quentin	Tarantino	78273	267038
43095	Stanley	Kubrick	43095	65764
78273	Quentin	Tarantino	78273	118367
..	..	..	..	..

Equi Join

Equi Join

The equi join forms the compound of two relations R_1 and R_2 under an equi join predicate, i.e. instead of specifying the join predicate directly, as with the theta join, subschemas of R_1 and R_2 are specified whose attribute values must be equal.

Let $[R_1'] \subseteq [R_1]$ and $[R_2'] \subseteq [R_2]$, $\left|[R_1']\right| == \left|[R_2']\right|$.

Then $[R_1']$ and $[R_2']$ define the following theta join predicate:

$\theta(r_1 \circ r_2) := \pi_{R_1'}(r_1 \circ r_2) == \pi_{R_2'}(r_1 \circ r_2)$.

Example:

assume the following subschemas:

$[\mathrm{R}_1'] = \{[\mathrm{ID}, \mathrm{salary}]\}$ and $[\mathrm{R}_2'] = \{[\mathrm{SID}, \mathrm{bonus}]\}$

$\Rightarrow \theta(r_1 \circ r_2) : \mathrm{ID} == \mathrm{SID} \wedge \mathrm{salary} == \mathrm{bonus}.$

$\mathrm{R}_1 \bowtie_{[\mathrm{R}_1'],[\mathrm{R}_2']} \mathrm{R}_2 = \mathrm{R}_1 \bowtie_{\{[\mathrm{ID},\mathrm{salary}]\},\{[\mathrm{SID},\mathrm{bonus}]\}} \mathrm{R}_2 = \mathrm{R}_1 \bowtie_{\theta} \mathrm{R}_2$

Equi Join: Formal Definition

Equi Join

$R_1 \bowtie_{\{[A_1:D_1,\ldots,A_n:D_n]\},\{[B_1:D_1,\ldots,B_n:D_n]\}} R_2 :=$
$R_1 \bowtie_{A_1=B_1 \wedge \ldots \wedge A_n=B_n} R_2$

Syntactic sugar:
$R_1 \bowtie_{A_1,\ldots,A_n;B_1,\ldots,B_n} R_2 =$
$R_1 \bowtie_{\{[A_1,\ldots,A_n]\},\{[B_1,\ldots,B_n]\}} R_2 =$
$R_1 \bowtie_{\{[A_1:D_1,\ldots,A_n:D_n]\},\{[B_1:D_1,\ldots,B_n:D_n]\}} R_2$

Extensions: Group by γ

Group by γ

Group by is a unary operator. It partitions the input relation R based on a subset of R's attributes (using **horizontal partitioning**). The operator outputs one tuple for each resulting group (using **aggregation**). Each of those tuples is projected to a user-defined format (using **projection**).

Let $[G] \subseteq [R]$ and $[G_1] \subseteq [R], \ldots, [G_n] \subseteq [R]$ be arbitrary (possibly empty) subschemas of $[R]$. Furthermore, assume we have a sequence of aggregate functions $\mathrm{f}_1 : [G_1] \to [\mathrm{D}_1], \ldots, \mathrm{f}_n : [G_n] \to [\mathrm{D}_n]$.
Then the result of group by is $R' := \gamma_{[G],\mathrm{f}_1([G_1]),\ldots,\mathrm{f}_n([G_n])}(R)$.

Syntactic sugar: Schemas are abbreviated as in projection.

Examples:

[R] : {[a:int, b:int]}, R = {(1,3), (2,4), (2,7), (3,3), (4,2)}

$\gamma_{\mathrm{a,\ count(*)}}(\mathrm{R})$ = { (1,1), (2,2), (3,1), (4,1) }

$\gamma_{\mathrm{b,\ count(*)}}(\mathrm{R})$ = { (3,2), (4,1), (7,1), (2,1) }

IMDb Examples

$\gamma_{\text{gender, count(*)}}(\text{actors})$

gender	count(*)
F	289
M	802

$\gamma_{\text{count(*)}}(\text{actors}) = |\text{actors}|$

count(*)
1091

$\gamma_{\text{first\_name, last\_name, count(*)}}(\text{directors} \bowtie_{\text{id=director\_id}} \text{movies\_directors})$

last_name	first_name	count(*)
Tarantino	Quentin	10
Kubrick	Stanley	16
Cameron	James (I)	14

The most important Aggregate Functions

sum(A)

Calculates the sum of the values of attribute A.

avg(A)

Calculates the average (arithmetic mean) of the values of attribute A.

min(A)

Calculates the minimum of the values of attribute A.

max(A)

Calculates the maximum of the values of attribute A.

count(*)

Calculates the number of tuples (independent of the attribute values).
Note that $|R| = \gamma_{\text{count}(*)}(R)$.

Group By γ

The name 'group by' is actually misleading for this operator. This is because **three different operations** are performed during 'group by':

(1.) **Grouping:** all tuples in the input relation are grouped according to the attributes in $[G]$ (in other words **horizontally partitioned**). All tuples that have the same values with respect to $[G]$ are assigned to the same group/horizontal partition.

(2.) **Aggregation:** for each group/horizontal partition created in (1.), aggregate functions $\mathrm{f}_1([G_1]), \ldots, \mathrm{f}_n([G_n])$ are applied, i.e. **for each group/horizontal partition, each of these functions is applied independently**.

(3.) **Projection:** for each group/horizontal partition created in (1.), a tuple is created with the associated aggregates from (2.). The schema of this tuple is $[G] \circ [\mathrm{D}_1, \ldots, \mathrm{D}_n]$.

Grouping: Formal Definition (1/2)

Horizontal Partition(ing) (Definition using equality of grouping keys)

Let $[G] = \{[A_1 : D_1, \ldots, A_n : D_n]\} \subseteq [R]$.
Let for each tuple $t = (a_1, \ldots, a_n) \in \pi_{[G]}(R)$ the relation R_t be defined as follows: $R_t = \sigma_{A_1=a_1 \wedge \ldots \wedge A_n=a_n}(R)$

Then R_t is a horizontal partition (HP).
The set $HPT_{[G]}(R) = \left\{R_t \mid t \in \pi_{[G]}(R)\right\}$ is called horizontal partitioning (HPT) of R.

It holds:
(1.) $R_i \cap R_j = \varnothing \; \forall_{i,j \in \pi_{[G]}(R) \wedge i \neq j}$ (disjointness)
(2.) $\bigcup_{t \in \pi_{[G]}(R)} R_t = R$ (completeness)

Group by: Formal Definition (2/2)

Group by

Let $[G] = \{[A_1 : D_1, \ldots, A_n : D_n]\} \subseteq [R]$.

Let $GB := \gamma_{[G],\mathrm{f}_1([G_1]),\ldots,\mathrm{f}_m([G_m])}(R)$

Then $GB := \left\{ t \circ f_1\left(\pi_{[G_1]}(R_t)\right) \circ \ldots \circ f_m\left(\pi_{[G_m]}(R_t)\right) \mid R_t \in HPT_{[G]}(R) \right\}$.

$[GB] = [G] \circ [\mathrm{D}_1, \ldots, \mathrm{D}_n]$

Horizontal Partitioning (in general)

Horizontal Partitioning (in general)

Let R be a relation. Any assignment of tuples from R to relations $R_1, \ldots, R_k$ is called *horizontal partitioning of* R (HPT) if $\forall_{t \in R} \exists R_i, 1 \leq i \leq k$ with $t \in R_i$. The relations R_i are called the *horizontal partitions* (HP) of R.

Examples:

$R = \{(2, A), (7, B), (1, B), (6, C)\}$

- $R_1 = \{(2, A), (1, B)\}$, $R_2 = \{(7, B), (6, C)\}$ is a HPT.
- $R_1 = \{(1, B)\}$, $R_2 = \{(7, B), (2, A), (6, C)\}$ is a HPT.
- $R_1 = \{(2, A), (1, B)\}$, $R_2 = \{(2, A), (6, C)\}$ is **not** a HPT.

Disjoint Horizontal Partitioning

A horizontal partitioning is called *disjoint*, if $R_i \cap R_j = \varnothing \ \forall i, j \neq i$.

$R_1 = \{(2, A), (1, B)$, $R_2 = \{(7, B), (2, A), (6, C)\}$ is a HPT but **not** disjoint.

Partitioning Function

Partitioning Function

Let D be an arbitrary domain. Any function $p : [R] \to D$ is called *partitioning function* of R.

Examples:

$[R] = \{[a : int, b : char]\}, R = \{(2, A), (7, B), (1, B), (6, C)\}$

$p_0 : [R] \to int, p_0(t) := t.a$ modulo 2

- $p_0((2, A)) = 0$
- $p_0((7, B)) = 1$
- $p_0((1, B)) = 1$
- $p_0((6, C)) = 0$

$p_1 : [R] \to char, p_1(t) := t.b$

- $p_1((2, A)) = A$
- $p_1((7, B)) = B$
- $p_1((1, B)) = B$
- $p_1((6, C)) = C$

Induced Horizontal Partitioning

Induced Horizontal Partitioning

Let $p : [R] \to D$ be a partitioning function. The horizontal partitioning of R into partitions R_i such that $\forall_{t \in R} t \in R_{p(t)}$ is called *induced horizontal partitioning*.

Examples:

$p_0 : [R] \to int, p_0(t) := t.a$ modulo 2

- $p_0((2, A)) = 0$
- $p_0((7, B)) = 1$
- $p_0((1, B)) = 1$
- $p_0((6, C)) = 0$

Induced horizontal partitioning:
$R_0 = \{(2, A), (6, C)\}$,
$R_1 = \{(7, B), (1, B)\}$

$p_1 : [R] \to char, p_1(t) := t.b$

- $p_1((2, A)) = A$
- $p_1((7, B)) = B$
- $p_1((1, B)) = B$
- $p_1((6, C)) = C$

Induced horizontal partitioning:
$R_A = \{(2, A)\}$, $R_B = \{(7, B), (1, B)\}$,
$R_C = \{(6, C)\}$

Co-Grouping Γ: Intuition

Horizontal Co-Partition(ing), aka Co-Grouping Γ

Co-grouping partitions two input relations R and S using two partitioning functions $p_R : [R] \rightarrow D$ and $p_S : [S] \rightarrow D$ that have the same target domain D. The output contains a tuple for each tuple in $\left\{d \mid d \in p_R(r) \forall r \in R \ \vee \ d \in p_S(s) \forall s \in S\right\}$.
The output of the co-grouping has the schema $\left[\text{d:D, R:[R], S:[S]}\right]$.

Example:
[R] : {[a:int, b:int]}, R = {(3,1), (4,2), (7,2), (3,3), (7,6)}
[S] : {[c:int, d:int]}, S ={(2,3), (1,4), (5,4), (3,8), (2,5)}

$p_R : [R] \rightarrow int, p_R(r) := r.b,\ p_S : [S] \rightarrow int, p_S(s) := s.c$

$$[\text{HCPT}]_{p_R,p_S} = \Big\{ \underbrace{\Big(1, \{(3,1)\}, \{(1,4)\}\Big)}_{d=1}, \underbrace{\Big(2, \{(4,2),(7,2)\}, \{(2,3),(2,5)\}\Big)}_{d=2},$$
$$\underbrace{\Big(3, \{(3,3)\}, \{(3,8)\}\Big)}_{d=3}, \underbrace{\Big(6, \{(7,6)\}, \{\}\Big)}_{d=6}, \underbrace{\Big(5, \{\}, \{(5,4)\}\Big)}_{d=5} \Big\}$$

Co-Grouping Γ: Formal Definition

Horizontal Co-Partition(ing), aka Co-Grouping Γ

Let:
$HPT_{p_R}(R) = \left\{R_{p_R(t)} \mid t \in R\right\}$ an HPT of R and
$HPT_{p_S}(S) = \left\{S_{p_S(t)} \in S\right\}$ an HPT of S.

Then:
$HCPT_{p_R,p_S}(R,S) = \left\{(d, R_d, S_d) \mid R_d \in HPT_{p_R}(R), S_d \in HPT_{p_S}(S))\right\}$
is the co-grouping (HCPT) of R and S.

Relationship of Co-Grouping and Equi Join: Formal Definition

Equivalence of Co-Grouping (plus Cross Product) and Equi Join

Let:
$HCPT_{p_S,p_T}(S,T)$ be an HCPT of S and T.

$[S' \subseteq S], [T' \subseteq T], \left|[S']\right| = \left|[T']\right|$ are the desired subschemas of an equi join.

$p_S : [S] \to D, p_S(s) := \pi_{[S']}(s)$
$p_T : [T] \to D, p_T(t) := \pi_{[T']}(t)$

Then it holds:
$\left\{v \mid v \in S_d \times T_d \text{ mit } (d, S_d, T_d) \in HCPT_{p_S,p_T}(S,T)\right\} = S \bowtie_{[S'],[T']} T.$

Example for Equivalence of Co-Grouping (plus Cross Product) and Equi Join

[R] : {[a:int, b:int]}, R = {(3,1), (4,2), (7,2), (3,3), (7,6)}
[S] : {[c:int, d:int]}, S ={(2,3), (1,4), (5,4), (3,8), (2,5)}

$p_R : [R] \rightarrow int, p_R(r) := r.b,\ p_S : [S] \rightarrow int, p_S(s) := s.c$

$$[\text{HCPT}]_{p_R,p_S} = \Big\{ \underbrace{\big(1, \{(3,1)\}, \{(1,4)\}\big)}_{d=1,\{(3,\underline{1},\underline{1},4)\}}, \underbrace{\big(2, \{(4,2),(7,2)\}, \{(2,3),(2,5)\}\big)}_{d=2,\{(4,\underline{2},\underline{2},3),(4,\underline{2},\underline{2},5),(7,\underline{2},\underline{2},3),(7,\underline{2},\underline{2},5)\}},$$

$$\underbrace{\big(3, \{(3,3)\}, \{(3,8)\}\big)}_{d=3,\{(3,\underline{3},\underline{3},8)\}}, \underbrace{\big(6, \{(7,6)\}, \{\}\big)}_{d=6,\{\}}, \underbrace{\big(5, \{\}, \{(5,4)\}\big)}_{d=5,\{\}} \Big\}$$

IMDb (Part 2)

and thus back to:

2. What are the data management and analysis issues behind this?

Question 3

How do we query this data?

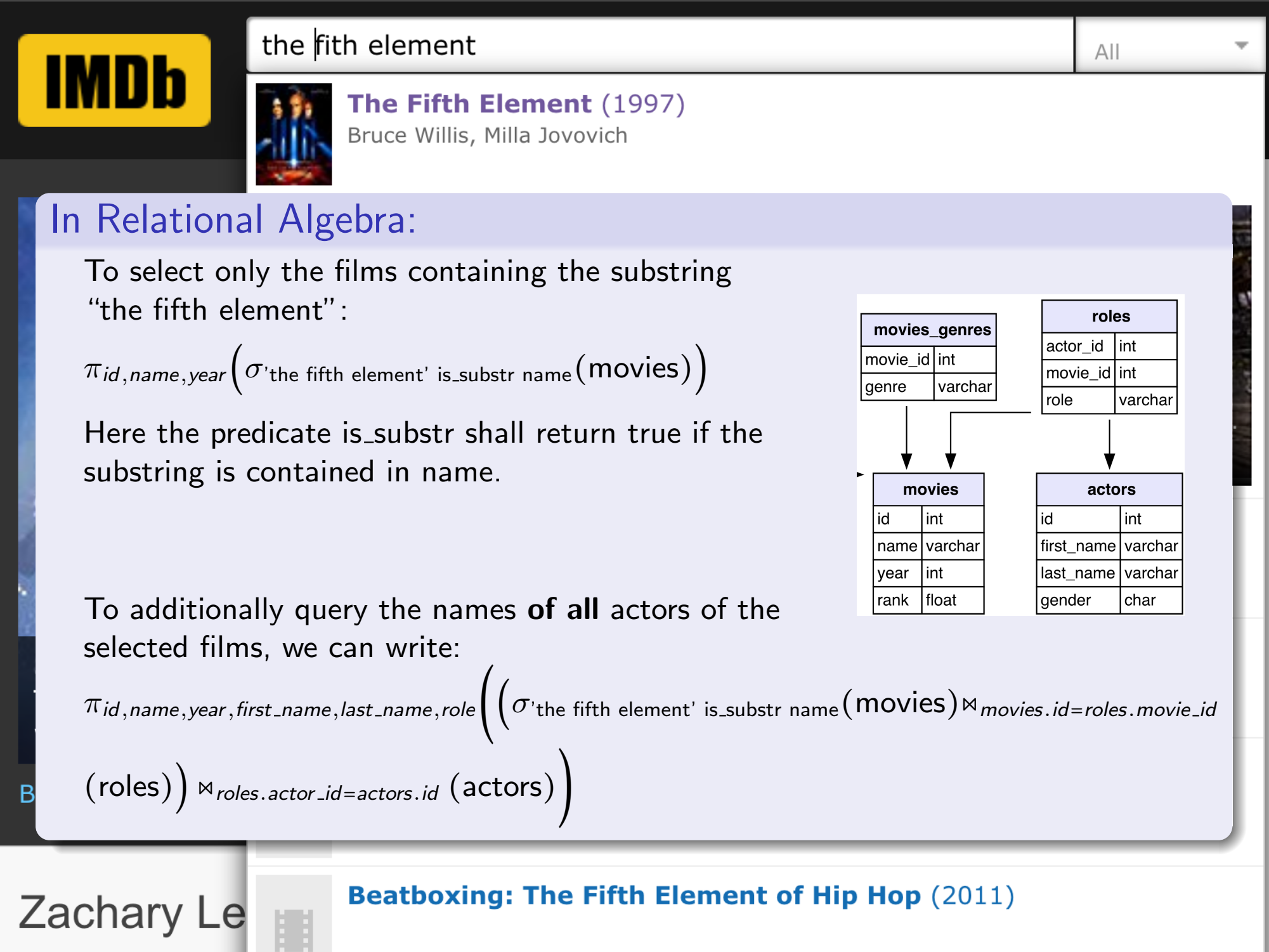
With relational algebra!

Transfer to IMDb

4. Transfer of the basics to the concrete application

We have already done this 'inline' this week and last week for ER, the relational model and relational algebra in numerous examples.

But, in addition to that, let's wrap up by going back to our initial example:

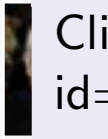

IMDb
the fith element
All
The Fifth Element (1997)
Bruce Willis, Milla Jovovich
In Relational Algebra:
To select only the films containing the substring "the fifth element":
$\pi_{id,name,year}\left(\sigma_{\text{'the fifth element' is\_substr name}}(\text{movies})\right)$
Here the predicate is_substr shall return true if the substring is contained in name.
movies_genres
movie_id int
genre varchar
roles
actor_id int
movie_id int
role varchar
movies
id int
name varchar
year int
rank float
actors
id int
first_name varchar
last_name varchar
gender char
To additionally query the names **of all** actors of the selected films, we can write:
$\pi_{id,name,year,first\_name,last\_name,role}\left(\left(\sigma_{\text{'the fifth element' is\_substr name}}(\text{movies}) \bowtie_{movies.id=roles.movie\_id} (\text{roles})\right) \bowtie_{roles.actor\_id=actors.id} (\text{actors})\right)$
Zachary Le
Beatboxing: The Fifth Element of Hip Hop (2011)

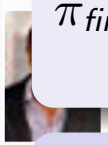
Cast

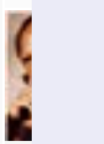
Edit

Cast overview, first billed only:

Bruce Willis ... Korben Dallas







Lee Evans ... Fog

In Relational Algebra:

Clicking on "The Fifth Element" in the web browser selects the film with id=112205 from the list generated above.

With this, we can formulate the query like this:

$$\pi_{first\_name,last\_name,role}\left(\sigma_{movies.id=112205}(\text{roles}) \bowtie_{roles.actor\_id=actors.id} (\text{actors})\right)$$

Problem:

What we **could not** express in relational algebra:

1. not all main actors but only k-many
2. only the greatest, most important actors (first billed only), i.e. order of tuples

Looking ahead to Next Week

Unfortunately, expressions in relational algebra are sometimes a bit hard to read. Therefore, another option is to hide relational algebra expressions under another language, i.e. design another query language and then translate that language automatically to the relational algebra.

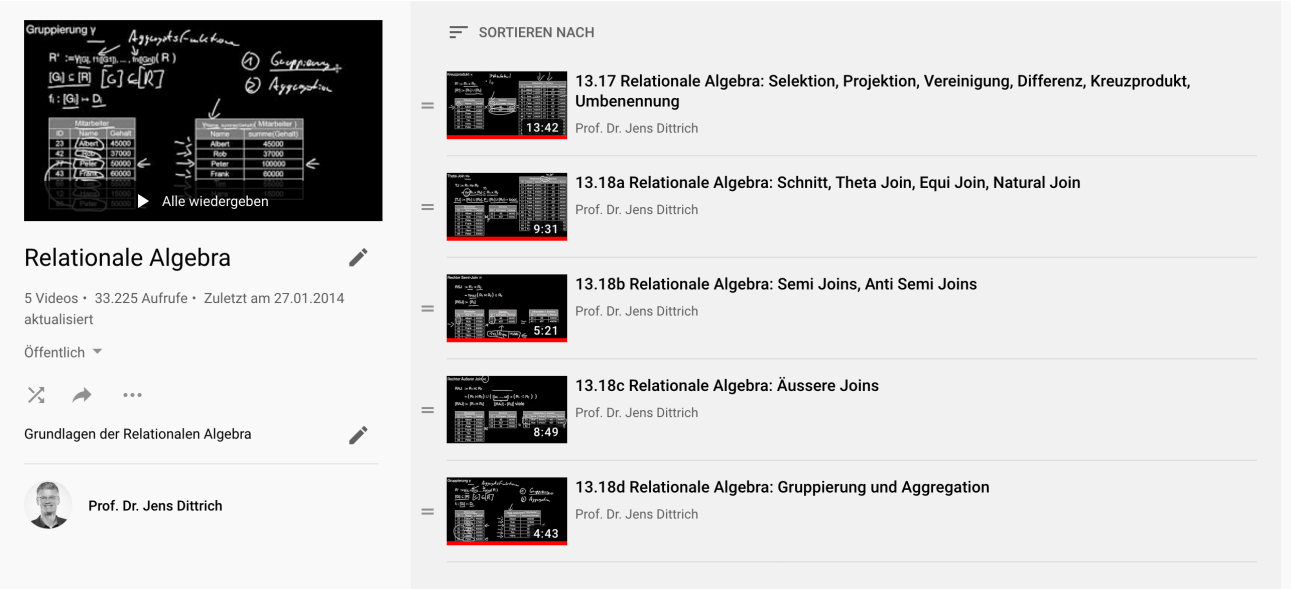
Fundamental Theorem of Software Engineering (FTSE):

"We can solve any problem by introducing an extra level of indirection . . . except for the problem of too many levels of indirection."

[David Wheeler]

And that's exactly what we will be doing next week....

Further material (in German and English)



Youtube Videos of Prof. Dittrich on Relational Algebra in German

- Chapters 3.4 in Kemper&Eickler in German
- Chapters 5.4–5.6 in Elmasri&Navathe in German or in English
- RelaX - relational algebra calculator contains the simplified IMDb schema used on the slides as well as the FotoDB schema, allows you to query that data using either relational algebra or SQL
  - gist for IMDb_sample
  - gist for fotodb