Towards a linear algebra semantics for SQL

J.N. Oliveira

InfoBlender Seminar Series 2016

February 17th, 2016

INESC TEC & University of Minho

Grant FP7-ICT 619606

Context

FP7-ICT 619606

About project **LeanBigData**:

> *" (...)* **queries** *[identifying] facts of interest take hours, days, or weeks, whereas business processes demand today shorter cycles.*

Project motto: *lean* big data!

However — **what** are we actually **leaning**?

What is, after all, a **query**?

Back to basics

There are **jobs**:

create table *jobs* (
j_code **char** (15) **not null**,
j_desc **char** (50),
j_salary **decimal** $(15, 2)$ **not null**);

j_code	j_desc	j_salary
Pr	*Programmer*	*1000*
SA	*System Analyst*	*1100*
GL	*Group Leader*	*1333*

Back to basics

There are **jobs**:

create table *jobs* (
j_code **char** (15) **not null**,
j_desc **char** (50),
j_salary **decimal** (15, 2) **not null**);

j_code	j_desc	j_salary
Pr	*Programmer*	*1000*
SA	*System Analyst*	*1100*
GL	*Group Leader*	*1333*

Back to basics

There are **employees**:

create table *empl* (
e_id **integer not null**,
e_job **char** (15) **not null**,
e_name **char** (15),
e_branch **char** (15) **not null**,
e_country **char** (15) **not null**);

e_id	e_job	e_name	e_branch	e_country
1	Pr	Mary	Mobile	UK
2	Pr	John	Web	UK
3	GL	Charles	Mobile	UK
4	SA	Ana	Web	PT
5	Pr	Manuel	Web	PT

Back to basics

There are **employees**:

create table *empl* (
e_id **integer not null**,
e_job **char** (15) **not null**,
e_name **char** (15),
e_branch **char** (15) **not null**,
e_country **char** (15) **not null**);

e_id	e_job	e_name	e_branch	e_country
1	*Pr*	*Mary*	*Mobile*	*UK*
2	*Pr*	*John*	*Web*	*UK*
3	*GL*	*Charles*	*Mobile*	*UK*
4	*SA*	*Ana*	*Web*	*PT*
5	*Pr*	*Manuel*	*Web*	*PT*

Query

Monthly salary total per country / branch:

```
select e_country, e_branch, sum (j_salary)
  from empl, jobs
  where j_code = e_job
  group by e_country, e_branch
  order by e_country;
```

`sqlite3:`

```
PT|Web|2100
UK|Mobile|2333
UK|Web|1000
```

Query

Impact of

```
insert into "jobs" values ('SA','System Admin',1000);
```

that is, *j_code* no longer a key.

`sqlite3:`

```
PT|Web|3100
UK|Mobile|2333
UK|Web|1000
```

Fine — so *SA* is taken as a kind of "multi-job".

But — where are these **semantics** specified?

Standard semantics

Given in English:

> *"The result of evaluating a query-specification can be explained in terms of a multi-step algorithm. The order of [the 7] steps in this algorithm follows the mandatory order of the clauses (FROM, WHERE, and so on) of the SELECT statement"*

Cf. pages 71-73 of

> *X/Open CAE Specification Data Management: Structured Query Language (SQL) Version 2 March 1996, X/Open Company Limited*

7 steps

1. *For each table-reference that is a joined-table, conceptually join the tables (...) to form a single table*
2. *Form a Cartesian product of all the table-references (...)*
3. *Eliminate all rows that do not satisfy the search-condition in the WHERE clause.*
4. *Arrange the resulting rows into groups (...)*
   - *If there is a GROUP BY clause specifying grouping columns, then form groups so that all rows within each group have equal values for the grouping columns (...)*
5. *If there is a HAVING clause, eliminate all groups that do not satisfy its search- condition (...)*
6. *Generate result rows based on the result columns specified by the select-list (...)*
7. *In the case of SELECT DISTINCT, eliminate duplicate rows from the result (...)*

Background

Join operator — ok, well defined in Codd's relation algebra.

However,

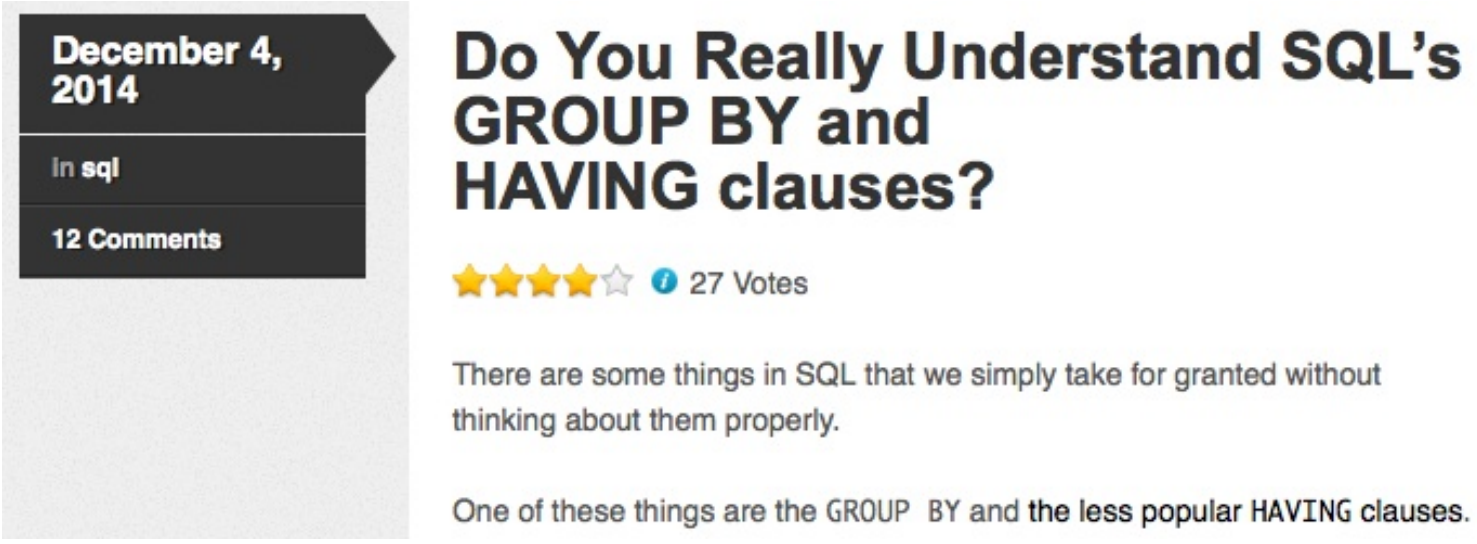
> *[...] relational DBMS were never intended to provide the very powerful functions for data synthesis, analysis and consolidation that is being defined as multi-dimensional data analysis.*
>
> *E.F.Codd* [1]

> *[...] expressing roll-up, and cross-tab queries with conventional SQL is daunting. [...] GROUP BY is an unusual relational operator [...]*
>
> *J. Gray et al* [2]

[1] Providing OLAP to User-Analysts: An IT Mandate (1998)

[2] Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals (1997)

Background



[http://blog.jooq.org/2014/12/04/
do-you-really-understand-sqls-group-by-and-having-clauses/]

Background

Why these shortcomings / questions ?

> *While* **relation algebra** *"à la Codd" [works] well for qualitative data science [it is] rather clumsy in handling the quantitative side [...] we propose to solve this problem by suggesting* **linear algebra** *(LA) as an alternative suiting both sides [...]*
>
> *H. Macedo, J. Oliveira* [3]

Linear algebra ...

[3] A linear algebra approach to OLAP (2015)

Formalizing SQL data aggregation

VLDB'87, among other research:

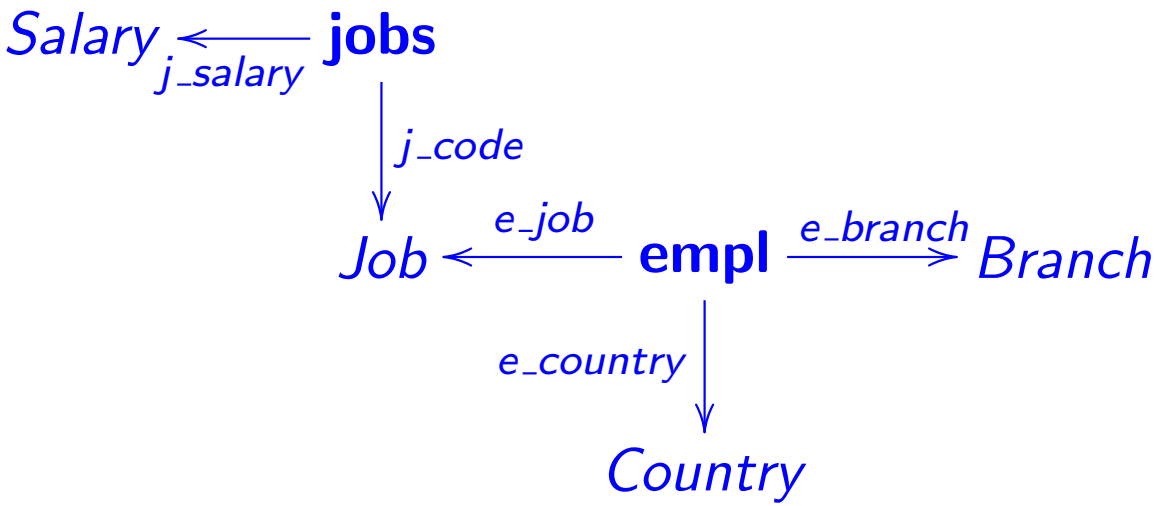
SQL Query		Calculus Expression
SELECT FROM WHERE	$f_1, \ldots, f_l$ $r_1(v_1), \ldots, r_n(v_n)$ P_w	$(f'_1, \ldots, f'_l)(\ * : r_1(v_1), \ldots, r_n(v_n) : P_w\ )$
SELECT FROM WHERE	$t_1, \ldots, t_l \ (\neq f)$ $r_1(v_1), \ldots, r_n(v_n)$ P_w	$(t_1, \ldots, t_l) : r_1(v_1), \ldots, r_n(v_n) : P_w$
SELECT FROM WHERE GROUP BY HAVING	$t_1, \ldots, t_l$ $r_1(v_1), \ldots, r_n(v_n)$ P_w $v_{i_1}[A_{i_1}], \ldots, v_{i_k}[A_{i_k}]$ P_h	$(t'_1, \ldots, t'_l) : \alpha(v) : P'_h$ $\alpha = (\phi_{<(A_{i_1}, \ldots, A_{i_k}), (f'_1, \ldots, f'_m)>}(\ * : r_1(v_1), \ldots, r_n(v_n) : P_w\ ))$; $(t'_1, \ldots, t'_l, P'_h) = (t_1, \ldots, t_l, P_h)[f_i/v[k+i], v_{i_j}[A_{i_j}]/v[j]]$; $(f_1, \ldots, f_m$ aggregate functions in $t_1, \ldots, t_l, P_h)$

G. Bultzingsloewen [4]

[4] Translating and optimizing SQL queries having aggregates (1987)

"Star" diagrams

Entities (cf. tables) surrounded (placed at the center of) by their **attributes**:



Entities marked in bold.

Attribute types made explicit, linking entities to each other.

"Star" diagrams

What is the (formal) meaning of the **arrows** in the diagram?

There is one arrow per **attribute** — **column** in the database table.

Assigning meanings to the arrows amounts to formalizing a **columnar** approach to SQL.[5]

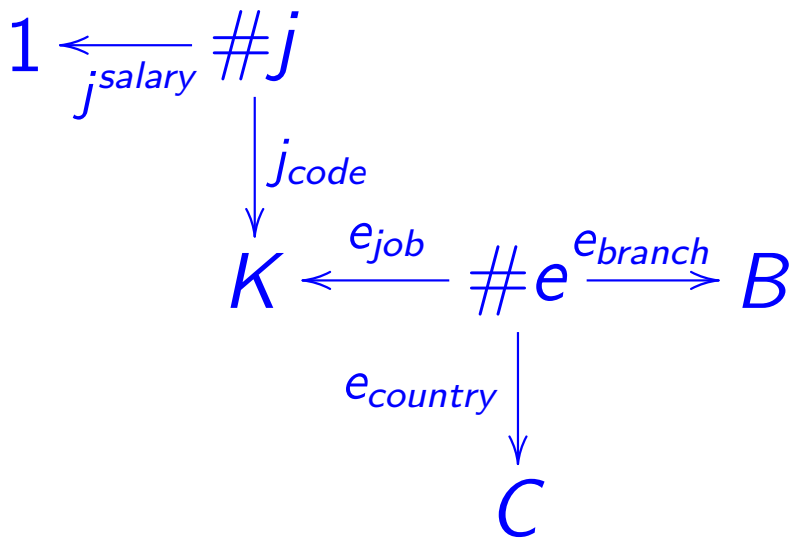
Let us do so using the **linear algebra of programming** (LAoP).[6]

---

[5] D. Abadi et al, *The Design and Implementation of Modern Column-Oriented Database Systems* (2012).

[6] J. Oliveira, *Towards a Linear Algebra of Programming* (2012).

Formal star-diagram in (**typed**) LAoP



Legend:

- **Types**:
  K — Job code
  C — Country
  B — Branch
  $\#e$ — *empl* record nrs
  $\#j$ — *jobs* record nrs
- **Dimensions**:
  - *branch*
  - *code*
  - *country*
  - *job*
- **Measures**:
  - *salary*

Dimensions

Dimension attribute columns are captured by **bitmap** matrices:

e_{branch}	1	2	3	4	5
Mobile	1	0	1	0	0
Web	0	1	0	1	1

e_{job}	1	2	3	4	5
GL	0	0	1	0	0
Pr	1	1	0	0	1
SA	0	0	0	1	0

$e_{country}$	1	2	3	4	5
PT	0	0	0	1	1
UK	1	1	1	0	0

j_{desc}	1	2	3
Group Leader	0	0	1
Programmer	1	0	0
System Analyst	0	1	0

j_{code}	1	2	3
GL	0	0	1
Pr	1	0	0
SA	0	1	0

Meaning of bitmap **matrix** t_d, for d a dimension of table t:

$$v\ t_d\ i = 1 \quad \Leftrightarrow \quad t[i].d = v \tag{1}$$

Measures

However — main difference wrt. **relation algebra** — we won't build

j_{salary}	1	2	3
1000	1	0	0
1100	0	1	0
1333	0	0	1

but rather the **row vector** $j^{salary} : \#j \longrightarrow 1$ which "internalizes" the quantitative information:

j^{salary}	1	2	3
1	1000	1100	1333

Summary:

Measures are *vectors*, **dimensions** are *matrices*.

Linear algebra

Matrix **multiplication**, given matrices $B \xleftarrow{M} C \xleftarrow{N} A$:

$$b\ (M \cdot N)\ a\ =\ \langle \sum\ c\ ::\ (b\ M\ c) \times (c\ N\ a) \rangle \tag{2}$$

Matrix **converse**:

$$c\ M^{\circ}\ b\ =\ b\ M\ c \tag{3}$$

Functions are (special cases of Boolean) matrices:

$$y\ f\ x\ =\ \begin{cases} 1\ \textbf{if}\ y = f\ x \\ 0\ \textit{otherwise} \end{cases} \tag{4}$$

Examples

$$1 \xleftarrow{j^{salary}} \#j \xleftarrow{j^{\circ}_{code}} K \;\Big|\; Pr \quad SA \quad GL$$

1	1000	1100	1333

Calculation:

$$1\ (j^{salary} \cdot j^{\circ}_{code})\ k$$

$\Leftrightarrow$ { multiplication (2) }

$$\langle \sum y \ ::\ (1\ j^{salary}\ y) \times (y\ j^{\circ}_{code}\ k) \rangle$$

$\Leftrightarrow$ { converse (3) ; vector j^{salary} }

$$\langle \sum y \ ::\ (k\ j_{code}\ y) \times (j[y].salary) \rangle$$

$\Leftrightarrow$ { functions (4) ; quantifier notation (details soon) }

$$\langle \sum y \ :\ k = j[y].code :\ j[y].salary \rangle$$

□

Examples

In case of the addition of

insert into "jobs" **values** ('SA', 'System Admin', 1000);

we get non-injective bitmap

j_{code}	1	2	3	4
GL	0	0	1	0
Pr	1	0	0	0
SA	0	**1**	0	**1**

and

j^{salary}	1	2	3	4
1	1000	1100	1333	1000

Therefore:

$1 \xleftarrow{j^{salary}} \#j \xleftarrow{j^{\circ}_{code}} K$	Pr	SA	GL
1	1000	**2100**	1333

Pointwise LAoP calculus

Quantifier notation follows the Eindhoven style,

$$\langle \sum x \ : \ R : \ T \rangle$$

where R is a predicate (**range**) and T is a numeric term.

In case $T = B \times M$ where Boolean $B = [\![P]\!]$ encodes predicate P, we have the **trading rule**:

$$\langle \sum x \ : \ R : \ [\![P]\!] \times M \rangle \ = \ \langle \sum x \ : \ R \wedge P : \ M \rangle \tag{5}$$

Thus

$$y(f \cdot N)x \quad = \quad \langle \sum z \ : \ y = f\, z : \ z\ N\ x \rangle \tag{6}$$

$$y(g^{\circ} \cdot N \cdot f)x \quad = \quad (g\ y)\ N\ (f\ x) \tag{7}$$

hold.

Pointwise LAoP calculus

Given a binary predicate $p : B \times A \to Bool$, we denote by $[\![p]\!] : B \leftarrow A$ the Boolean matrix which encodes p, that is,

$$b\ [\![p]\!]\ a = \textbf{if}\ p\ (b, a)\ \textbf{then}\ 1\ \textbf{else}\ 0 \tag{8}$$

In case of a unary predicate $q : A \to Bool$, $[\![q]\!] : 1 \leftarrow A$ is the Boolean vector such that:

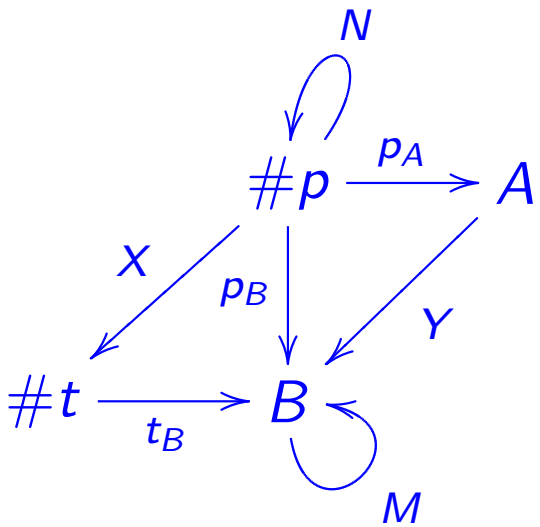
$$1\ [\![q]\!]\ a = [\![q]\!]\ [a] = \textbf{if}\ q\ a\ \textbf{then}\ 1\ \textbf{else}\ 0 \tag{9}$$

Joins and tabulations

SQL querying amounts to **following paths** in star diagrams.

The **meaning of a path** is obtained by composing (multiplying) the matrices involved.

Two particular such compositions deserve special reference, as they correspond to well-known operations in data processing:



- **Join**: $X = t_B^\circ \cdot M \cdot p_B$
- **Tabulation**: $Y = p_B \cdot N \cdot p_A^\circ$

M and N are whatever matrices of their type.

Simple Examples

Equi-join ($M = id$):

$j^{\circ}_{code} \cdot e_{job}$	1	2	3	4	5
1	1	1	0	0	1
2	0	0	0	1	0
3	0	0	1	0	0

Pointwise meaning: $j[y].code = e[x].job$ recall (7).

Counting tabulation ($N = id$):

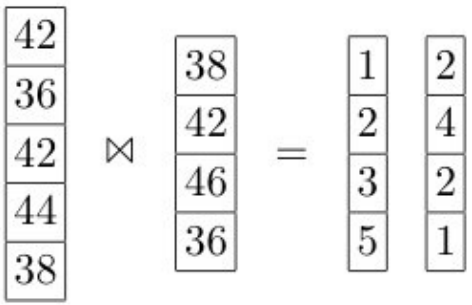
$e_{country} \cdot e^{\circ}_{branch}$	Mobile	Web
PT	0	2
UK	2	1

Pointwise meaning: $\langle \sum\ k\ :\ y = e[k].country \wedge x = e[k].branch :\ 1 \rangle$
recall (6), for y a country, x a branch.

Columnar joins

Excerpt from Abadi et al[7]

> For example, the figure below shows the results of a join of a column of size 5 with a column of size 4:



shows **columnar-join** "isomorphic" to our matrix joins:

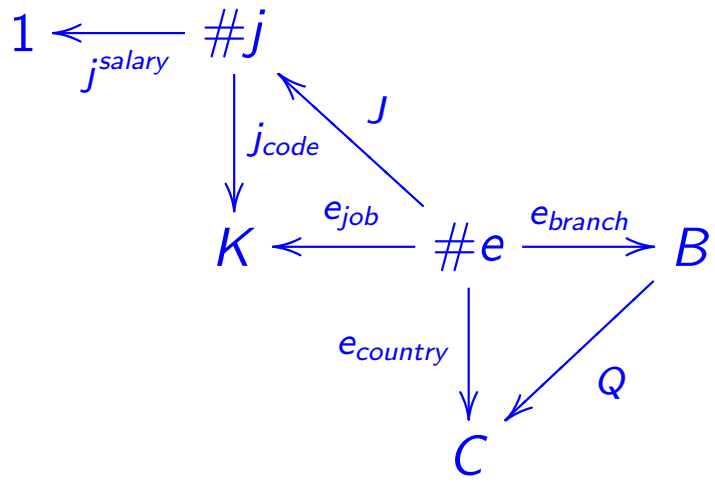
	1	*2*	*3*	*4*	*5*
1	*0*	*0*	*0*	*0*	*1*
2	*1*	*0*	*1*	*0*	*0*
3	*0*	*0*	*0*	*0*	*0*
4	*0*	*1*	*0*	*0*	*0*

[7] *The Design (..) of Modern Column-Oriented Database Systems* (2012).

Back to the starting SQL query

```
select
    e_branch,
    e_country,
    sum (j_salary)
    from empl, jobs
        where j_code = e_job
    group by
        e_country,
        e_branch
    order by
        e_country;
```

Minimal diagram accommodating query:



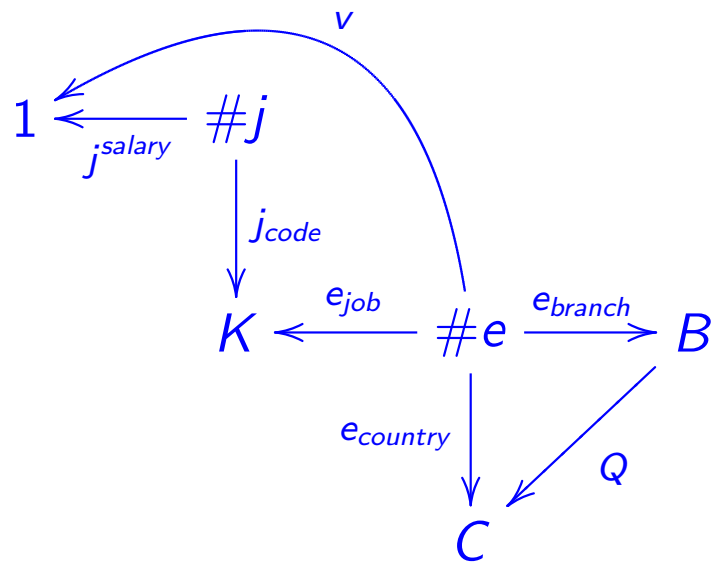
Clearly,

group by $\Rightarrow$ *tabulation* Q
where $\Rightarrow$ *join* J

Back to the starting SQL query

```
select
    e_branch,
    e_country,
    sum (j_salary)
    from empl, jobs
        where j_code = e_job
    group by
        e_country,
        e_branch
    order by
        e_country;
```

How do **salaries** get involved? We need a direct path from employees to (their) salaries,



involving the **where**-clause join:

$$v = j^{salary} \cdot j^{\circ}_{code} \cdot e_{job} \qquad (10)$$

Query = Group by + Join

The **group by** clause calls for a tabulation — but, how does vector

$j^{salary} \cdot j^{\circ}_{code} \cdot e_{job}$	1	2	3	4	5
1	1000	1000	1333	1100	1000

get into the place of N in the generic scheme?

Easy: every vector v can be turned into a **diagonal** matrix, e.g.

$v \triangledown id$	1	2	3	4	5
1	1000	0	0	0	0
2	0	1000	0	0	0
3	0	0	1333	0	0
4	0	0	0	1100	0
5	0	0	0	0	1000

and vice versa.

Khatri-Rao product

This diagonalization resorts to another LA operator, termed Khatri-Rao product $(M \triangledown N)$ defined by

$$(b, c)\ (M \triangledown N)\ a \quad = \quad (b\ M\ a) \times (c\ N\ a) \tag{11}$$

Then:

$$\begin{array}{ll} & b\ (v \triangledown id)\ c \ = \ v\ [c] \times (b\ id\ c) \\ \Leftrightarrow & \qquad \{\ \text{Khatri-Rao (11) ; function } id\ \} \\ & b\ (v \triangledown id)\ c \ = \ v\ [c] \times (b = c) \\ \Leftrightarrow & \qquad \{\ \text{pointwise LAoP (8)}\ \} \\ & b\ (v \triangledown id)\ c \ = \ \textbf{if}\ b = c\ \textbf{then}\ v\ [c]\ \textbf{else}\ 0 \end{array}$$

i.e. non-zeros can only be found in the **diagonal**.

Linear algebra

Property of diagonal matrices:

$$(v \triangledown id) \cdot (u \triangledown id) = (v \times u) \triangledown id \tag{12}$$

where $M \times N$ is the Hadamard product:

$$b\ (M \times N)\ a = (b\ M\ a) \times (b\ N\ a) \tag{13}$$

Moreover, for f a function, rule

$$f \triangledown v = f \cdot (v \triangledown id) \tag{14}$$

is easy to derive:

$$\begin{array}{ll} & b\ (f \cdot (v \triangledown id))\ a \\ \Leftrightarrow & \quad \{\ \text{composition ; Khatri-Rao}\ \} \\ & \langle \sum c\ ::\ (b\ f\ c) \times (v\ [a] \times (c\ id\ a)) \rangle \\ \Leftrightarrow & \quad \{\ \text{trading (5) ; cancel } \textstyle\sum \text{ cf. } c = a\ \} \\ & (b\ f\ a) \times v\ [a] \\ \Leftrightarrow & \quad \{\ \text{Khatri-Rao}\ \} \\ & b\ (f \triangledown v)\ a \end{array}$$

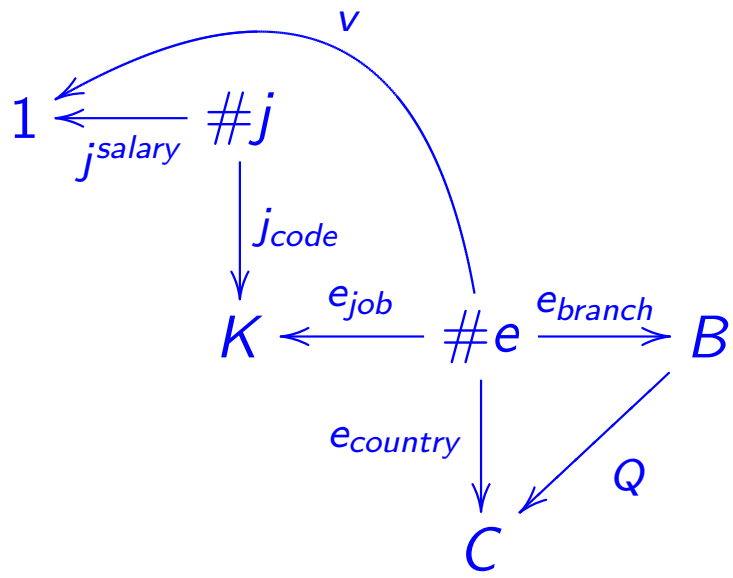
□

Query = Group by + Join

Query:

```
select
  e_branch,
  e_country,
  sum (j_salary)
  from empl, jobs
    where j_code = e_job
  group by
    e_country,
    e_branch
  order by
    e_country;
```

Diagram:



LA semantics:

$$Q = e_{country} \cdot (v \triangledown id) \cdot e^{\circ}_{branch} \tag{15}$$

where $v = j^{salary} \cdot j^{\circ}_{code} \cdot e_{job}$

Pointwise semantics

Of vector v first:

$$
\begin{array}{ll}
 & v\,[k] \\
= & \quad \{ \text{ definition (10) } \} \\
 & 1\,(j^{salary} \cdot j^{\circ}_{code} \cdot e_{job})\;k \\
= & \quad \{ \text{ matrix multiplication (2) } \} \\
 & \langle \sum i \;::\; (1\;j^{salary}\;i) \times (i\;(j^{\circ}_{code} \cdot e_{job})\;k) \rangle \\
= & \quad \{ \text{ trading rules (7) and (5) } \} \\
 & \langle \sum i \;:\; j_{code}\;i = e_{job}\;k :\; (1\;j^{salary}\;i) \rangle \\
= & \quad \{ \text{ pointwise notation conventions } \} \\
 & \langle \sum i \;:\; j[i].code = e[k].job :\; j[i].salary \rangle
\end{array}
$$

□

Pointwise semantics

Of the whole query:

$$
\begin{array}{ll}
 & c\ Q\ b \\
= & \quad \{\ \text{definition (15) ; diagonal } v \triangledown id\ \} \\
 & \langle \sum k\ ::\ (c\ e_{country}\ k) \times (k\ (v \triangledown id)\ k) \times (k\ e^{\circ}_{branch}\ b) \rangle \\
\Leftrightarrow & \quad \{\ \text{trading rule (5)}\ \} \\
 & c\ Q\ b = \langle \sum k\ :\ c = e_{country}\ k \wedge b = e_{branch}\ k :\ v\ [k] \rangle
\end{array}
$$

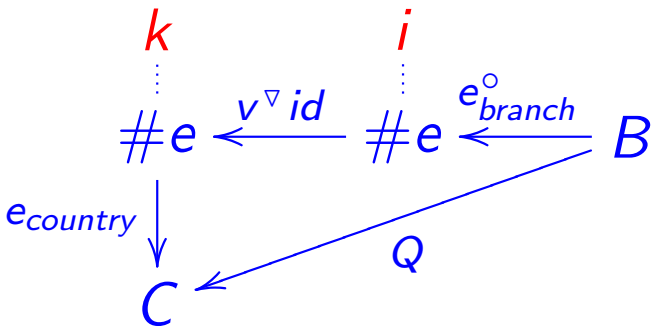
Putting both together:

$$
\begin{array}{l}
query\ (c, b) = \sum k, i : \\
\qquad c = e[k].country \wedge b = e[k].branch \wedge j[i].code = e[k].job : \\
\qquad j[i].salary
\end{array}
$$

Rest point :-)

Clearly:

- SQL is a **path-language**
- SQL is **pointfree** — see how the surface language **hides** the double-cursor k, i pointwise **for**-loop.



SQL tries to be as **pointfree** as **natural** language is so, compare

"dogs are mammals"

to the (boring!)

$$\langle \forall\ d\ :\ d \in Dog\ :\ d \in Mammal \rangle$$

We don't **speak** using *cursors*!

Simplification

LA script (15)

$$Q = e_{country} \cdot (v \triangledown id) \cdot e^{\circ}_{branch} \textbf{ where } v = j^{salary} \cdot j^{\circ}_{code} \cdot e_{job}$$

can be simplified into

$$Q = (e_{country} \triangledown v) \cdot e^{\circ}_{branch}$$

thanks to Khatri-Rao law (14). Note how matrix

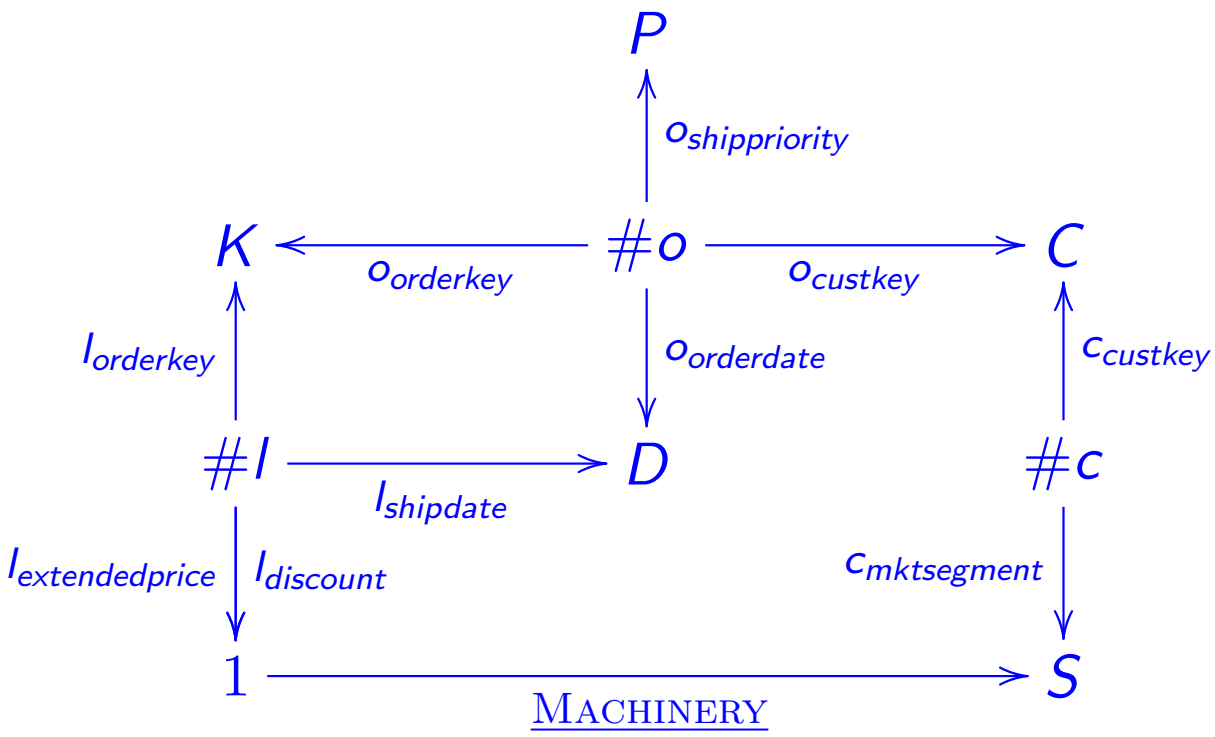
$e_{country} \triangledown v$	1	2	3	4	5
PT	0	0	0	1100	1000
UK	1000	1000	1333	0	0

nicely combines **qualitative** (functional) with **quantitative** information.

LA script for TPC-H query3


```
query3 =
  select
    l_orderkey, o_orderdate, o_shippriority;
    sum (l_extendedprice * (1 - l_discount)) as revenue
  from
    orders, customer, lineitem
  where
    c_mktsegment = 'MACHINERY'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-10'
    and l_shipdate > date '1995-03-10'
  group by
    l_orderkey, o_orderdate, o_shippriority
  order by
    revenue desc, o_orderdate;
```

Diagram for TPC-H query3



"Big-plan" **tabulation** again dictated by the **group by** clause:

$$Q = K \xleftarrow{l_{orderkey}} \#l \xleftarrow{\chi} \#o \xleftarrow{(o_{shippriority} \triangledown o_{shipdate})^{\circ}} P \times D$$

LA semantics for TPC-H query3

Data aggregation is performed over a derived vector

$$revenue = l_{extendedprice} \times (! - l_{discount}) \tag{16}$$

where $! : \#l \longrightarrow 1$ is the unique (constant) function of its type — a row vector wholly filled with ones.

We move on:

$$Q = \quad \#o \xleftarrow{(o_{shippriority} \triangledown o_{shipdate})^\circ} P \times D$$

$$\#o \xrightarrow{X} \#l \xrightarrow{l_{orderkey}} K$$

$$\#o \xrightarrow{Y} \#l \xrightarrow{revenue \triangledown id} \#l$$

LA semantics for TPC-H query3

As expected, the link Y between the two tables is the join in the **where** clause:

$$\begin{array}{ccccc} & & \#o & \xleftarrow{(o_{shippriority} \triangledown o_{shipdate})^\circ} & P \times D \\ & & \Big\downarrow{\scriptstyle Y=(l_{orderkey})^\circ \cdot o_{orderkey}} & & \\ K & \xleftarrow{l_{orderkey}} & \#l & \xleftarrow[revenue \triangledown id]{} & \#l \end{array}$$

LA semantics for TPC-H query3

Moving on, clauses

```
o_orderdate < date '1995-03-10'
and l_shipdate > date '1995-03-10'
```

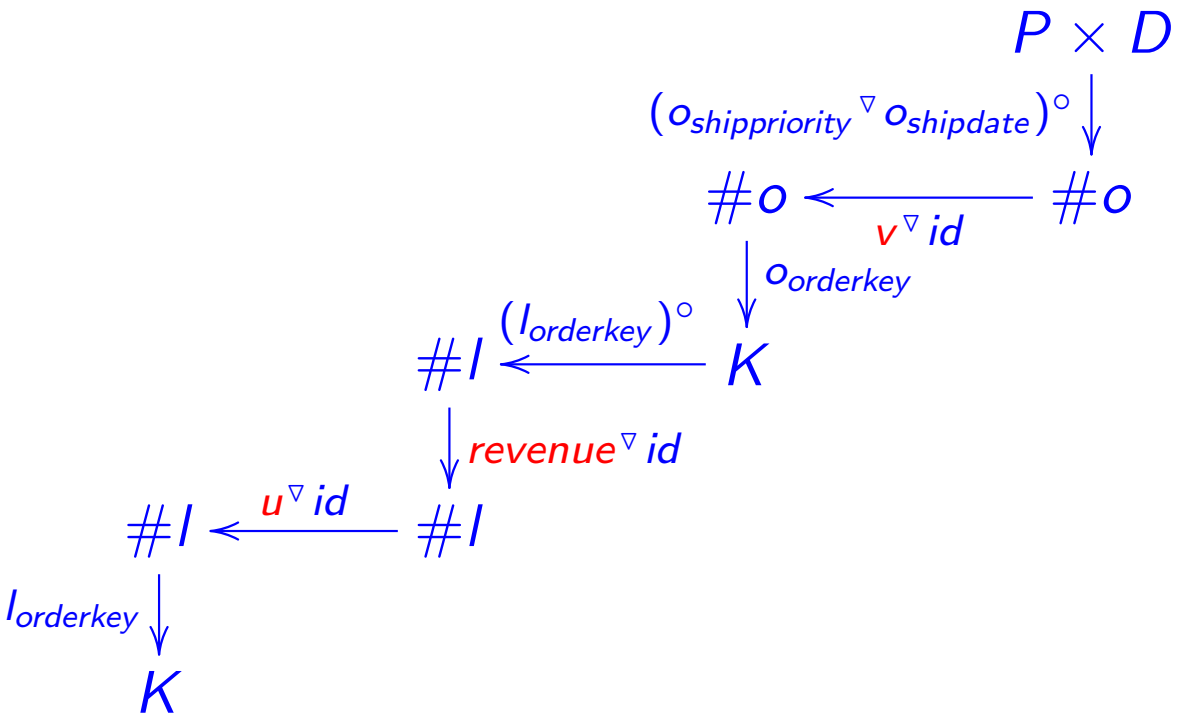
convert to vectors $v : \#o \to 1$ and $u : \#l \to 1$ defined by

$$v\,[i] = [\![o[i].orderdate < \text{'1995-03-10'}]\!]$$
$$u\,[k] = [\![l[k].shipdate > \text{'1995-03-10'}]\!]$$

recall (9).

LA semantics for TPC-H query3

Altogether, thus far:



where $v\,[i] = [\![o[i].orderdate < \text{'1995-03-10'}]\!]$
and $u\,[k] = [\![l[k].shipdate > \text{'1995-03-10'}]\!]$

LA semantics for TPC-H query3

Finally, clauses

```
c_mktsegment = 'MACHINERY' and c_custkey = o_custkey
```

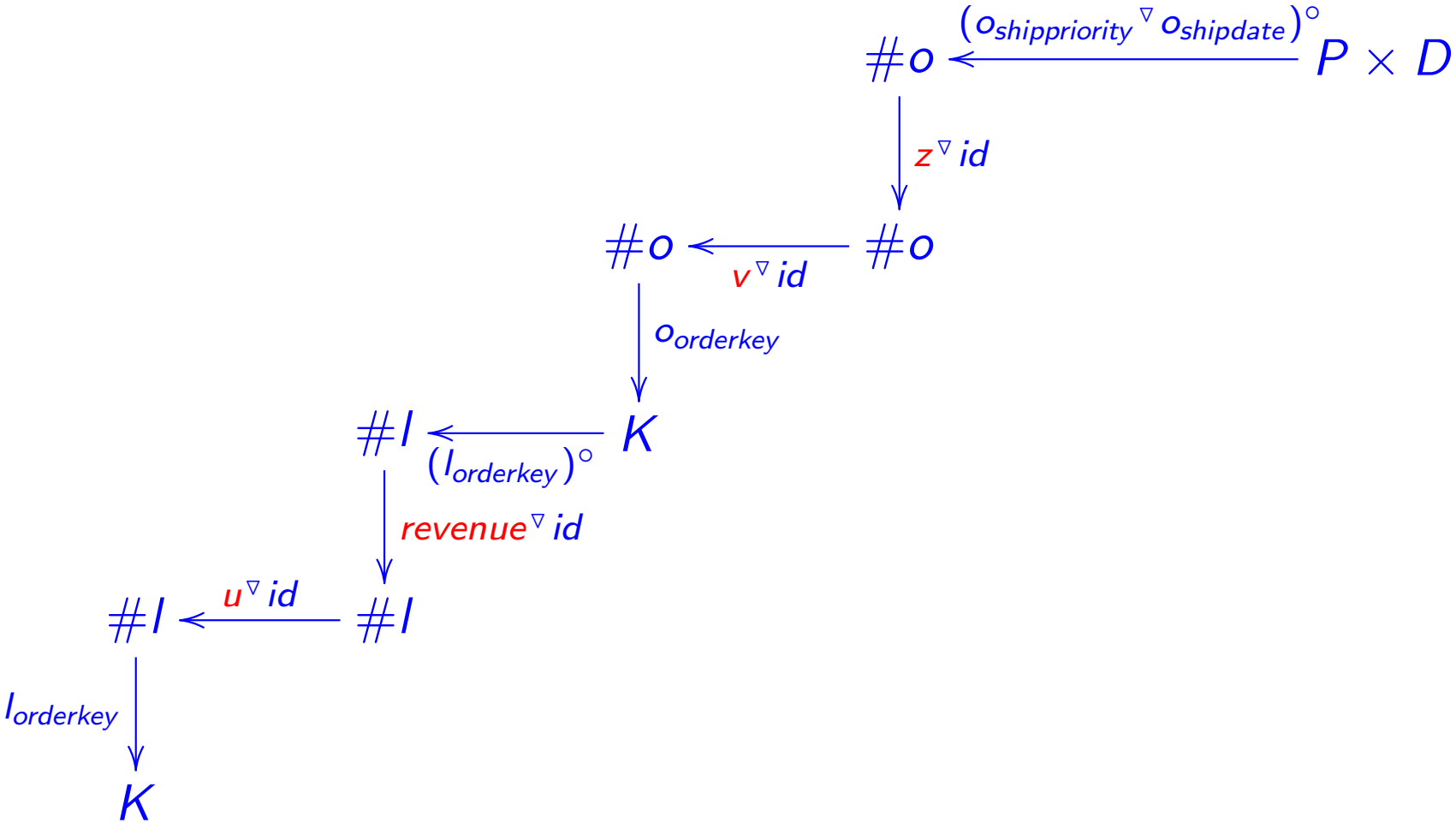
amount to Boolean path (vector)

$$z = 1 \xleftarrow{\text{MACHINERY}^{\circ}} S \xleftarrow{c_{mktsegment}} \#c \xleftarrow{c^{\circ}_{custkey}} C \xleftarrow{o_{custkey}} \#o$$

which **counts** how many customers exhibit the specified market segment:

$$z\,[k] = \langle \sum i \;:\; c[i].custkey = o[k].custkey \wedge c[i].mktsegment = MACHINERY \;:\; 1 \rangle$$

Query final path



Simplification of ("water fall") path

Thanks to LA laws:

$$Q3 = \begin{array}{ccc} & & \#o \xleftarrow{(o_{shippriority} \triangledown o_{shipdate})^\circ} P \times D \\ & & \downarrow {\scriptstyle o_{orderkey} \triangledown (v \times z)} \\ \#l & \xleftarrow[(l_{orderkey})^\circ]{} & K \\ {\scriptstyle (l_{orderkey})^\triangledown (revenue \times u)} \downarrow & & \\ K & & \end{array}$$

Notice the same overall pattern: a **join** inside a **tabulation**.

Other simplifications possible, likely changing **performance** — in what sense ?

Divide and conquer

Block linear algebra enables **distributed** evaluation of query paths by "divide & conquer" laws for **all** operators involved, cf.

$$[A|B] \cdot \left[\frac{C}{D}\right] = A \cdot C + B \cdot D \tag{17}$$

$$\left[\frac{A}{B}\right]^{\circ} = [A^{\circ}|B^{\circ}] \tag{18}$$

and

$$[A|B] \triangledown [C|D] = [A \triangledown C|B \triangledown D] \tag{19}$$

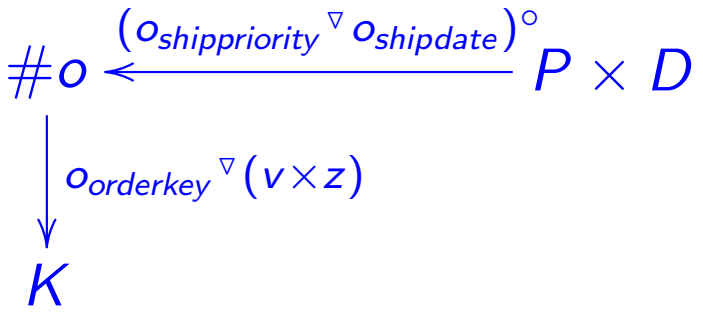
$$[A|B] \times [C|D] = [A \times C|B \times D] \tag{20}$$

which generalize to **any finite** number of blocks.

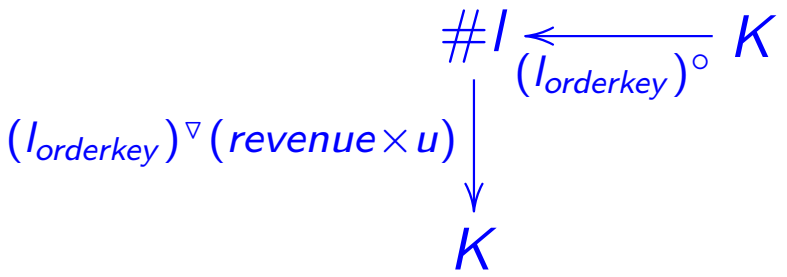
Map-reduce

Overall path splits in two parts,

- Workload over table $\#o$:



- Workload over table $\#l$:



With n **machines**, each table is divided into n **slices**, each slice residing into its machine.

Map runs the two workloads on each machine, in parallel.

Reduce joins all machine-contributions together, then performing the final composition of the 2 paths.

Summary

Recall the X/Open CAE Specification:

> *"The result of evaluating a query-specification can be explained in terms of a multi-step algorithm. The order of [the 7] steps in this algorithm follows the mandatory order of the clauses (FROM, WHERE, and so on) of the SELECT statement"*

Our **evaluation order** is clearly different !

It is "demand driven" by the **group by** clause.

In theory, everything is **embarrassingly parallel**... but read Rogério's MSc dissertation [8] before getting too excited...

[8] R. Pontes, Benchmarking a Linear Algebra Approach to OLAP (2015)

Practical side of all this

Future (practical) work:

- Define a **DSL** for the LA **path** language
- Mount a **map-reduce** interpreter for such a DSL running on a data-distributed environment
- Write a **compiler** mapping (a subset of) **SQL** to the DSL
- Enjoy experimenting with the overall toy :-)

In particular,

- Compare LA paths with TPC-H query plans
- Complete the benchmark already carried out.[9]

[9] R.Pontes, Benchmarking a Linear Algebra Approach to OLAP (2015).

Theory side of all this

A lot!

- Compare with related work on **columnar** DB systems
- Parametrize DSL on appropriate **semirings** for non arithmetic aggregations (*min*, *max* etc)
- Extend semantic coverage as much as possible, keeping the LA encoding such as e.g. in

  $$t_B^\circ \cdot t_B = id$$

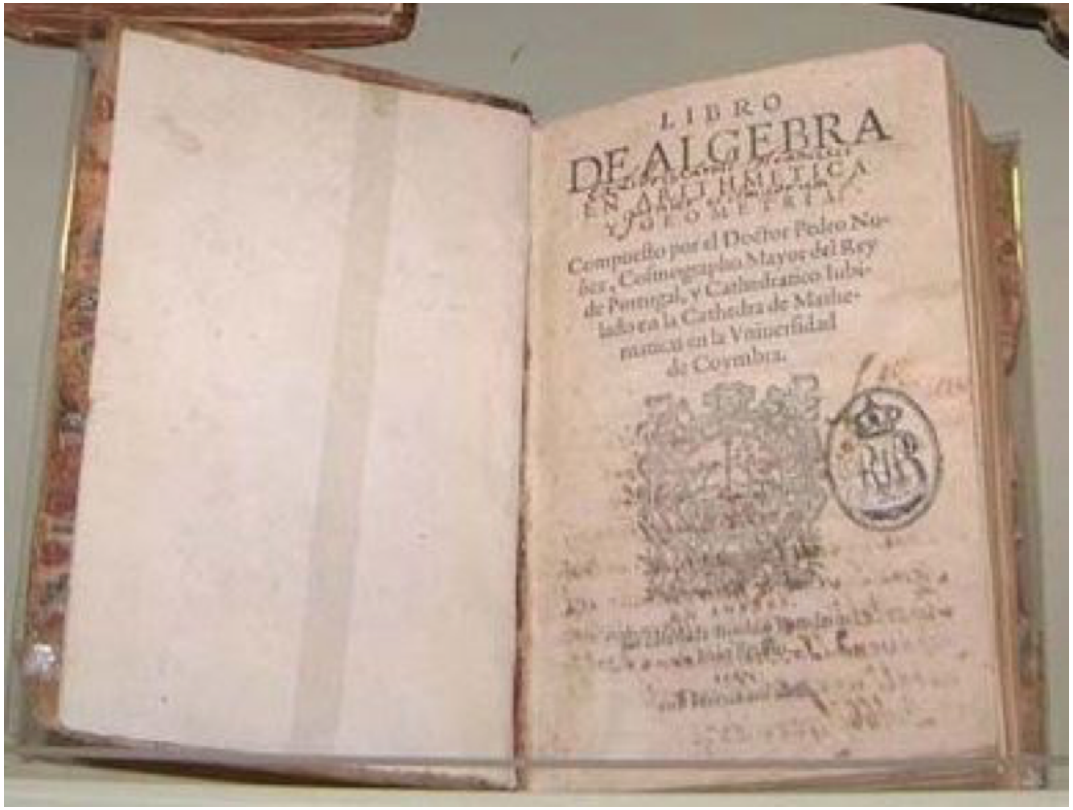
  expressing **UNIQUE** constraints, or **integrity constraints** such as in e.g.

  $$p_F \leqslant t_K \cdot t_K^\circ \cdot p_F$$

  (K primary key, F foreign key.)
- **Null values** ? ...

Today, as in 1567...

... quien sabe por Algebra, sabe scientificamente [10]



[10] (...) *who knows by Algebra knows scientifically* — Pedro Nunes, Libro de Algebra (1567).

Appendix

What about queries without **group by**?

Query:[11]

select
sum (r_a)
from r, s
where $r\_c = s\_b$ *and*
$5 < r\_a < 20$ *and*
$40 < r\_b < 50$ *and*
$30 < s\_a < 40$;

Star diagram:

$$1 \xleftarrow[r^a]{} \#r \xrightarrow{r_b} B$$
$$\downarrow r_c$$
$$C \xleftarrow{s_b} \#s \xrightarrow{s_a} A$$

Define

$$u\ i = 5 < r[i].a < 20$$
$$v\ i = 40 < r[i].b\ 50$$
$$x\ j = 30 < s[j].a < 40$$

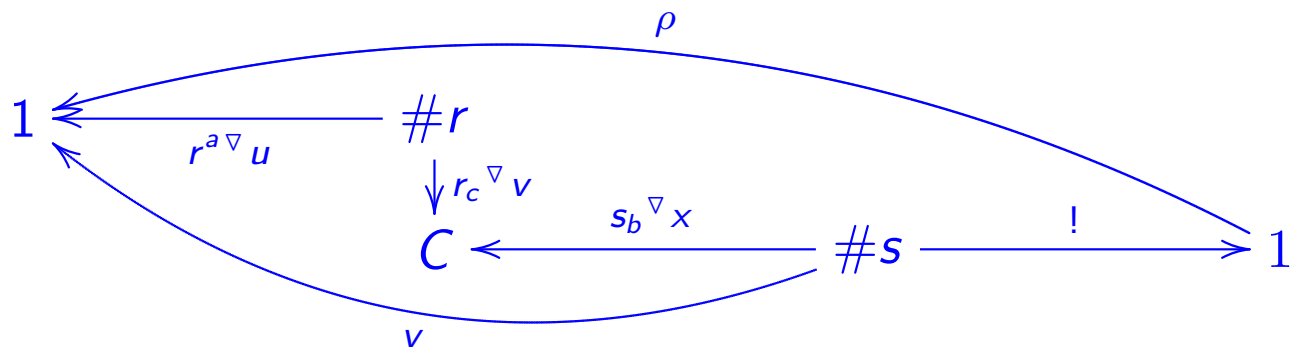
in the reduction:

$$1$$
$$\uparrow [\![u]\!]$$
$$1 \xleftarrow[r^a]{} \#r \xrightarrow{[\![v]\!]} 1$$
$$\downarrow r_c$$
$$C \xleftarrow{s_b} \#s \xrightarrow{[\![x]\!]} 1$$

[11] Example taken from D. Abadi et al, *The Design (...) Systems* (2012).

Faster, this time

Vector $\#s \xrightarrow{!} 1$ models the implicit '**group by** *all*' clause:



(21)

Thanks to (LA)

$$(M \triangledown N)^\circ \cdot (P \triangledown Q) = (M^\circ \cdot P) \times (N^\circ \cdot Q) \tag{22}$$

$$b\ (v^\circ \cdot u)\ a = v[b] \times u[a] \tag{23}$$

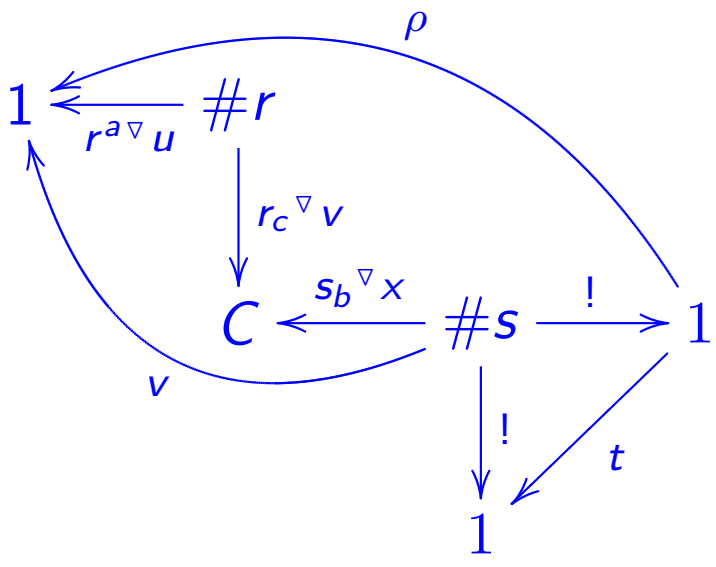
$$1\ (! \cdot M)\ a = \langle \sum\ b\ ::\ b\ M\ a \rangle \tag{24}$$

we get the expected output scalar:

$$\rho = \langle \sum\ j, i\ :\ u\ i \wedge v\ i \wedge r[i].c = s[j].b \wedge x\ j :\ r[i].a \rangle$$

Details

Details about the "hidden" tabulation in (21):



$$
\begin{aligned}
& t = ! \cdot (v \triangledown id) \cdot !^{\circ} \\
\Leftrightarrow \quad & \{ \ (14) \ \} \\
& t = (v \triangledown !) \cdot !^{\circ} \\
\Leftrightarrow \quad & \{ \ ! \text{ is the unit of Khatri-Rao} \ \} \\
& t = v \cdot !^{\circ} \\
\Leftrightarrow \quad & \{ \ \text{definition of } \rho \ \} \\
& t = \rho \\
& \square
\end{aligned}
$$