The Relational Data Borg is Learning: Part Deux

fdbresearch.github.io relational.ai

Dan Olteanu

University of Zurich

VLDB 2020 Keynote
Virtual Tokyo, Sept 1, 2020

Where We Are

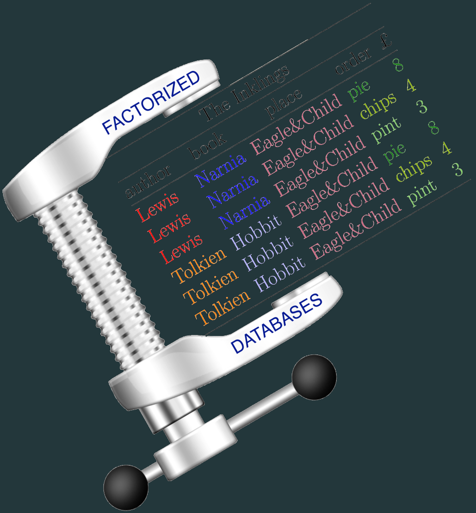
Covered so far:

- Relational data is ubiquitous
- Structure-agnostic learning is the state of the art
- Structure-aware learning can be much faster
- Idea 1: Turn learning into a DB workload challenge

To come: Exploit structure of the data and problem

- Idea 2: Lower the asymptotics
- Idea 3: Lower the constant factors

Idea 2: Exploit Problem Structure to Lower Complexity



author	book	place	order	£
Lewis	Narnia	Eagle&Child	pie	8
Lewis	Narnia	Eagle&Child	chips	4
Lewis	Narnia	Eagle&Child	pint	3
Tolkien	Hobbit	Eagle&Child	pie	8
Tolkien	Hobbit	Eagle&Child	chips	4
Tolkien	Hobbit	Eagle&Child	pint	3

Structure-aware Tools of a Database Researcher

Algebraic structure: (semi)rings $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$

- Distributivity law $\rightarrow$ Factorisation

  Factorised Databases [VLDB'12+'13,TODS'15,SIGREC'16]

  Factorised Machine Learning [SIGMOD'16+'19,DEEM'18,PODS'18+'19, TODS'20]

- Additive inverse $\rightarrow$ Uniform treatment of updates

  Factorised Incremental Maintenance [SIGMOD'18+'20]

- Sum-Product abstraction $\rightarrow$ Same processing for distinct tasks

  DB queries, Covariance matrix, PGM inference, Matrix chain multiplication

  [SIGMOD'18+'19]

Structure-aware Tools of a Database Researcher

Combinatorial structure: query width and data degree measures

- Width measure w for FEQ $\rightarrow$ Low complexity $\tilde{O}(N^w)$

  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  worst-case optimal size and time for factorised joins

  [ICDT'12+'18,TODS'15,PODS'19,TODS'20]

- Degree $\rightarrow$ Adaptive processing depending on high/low degrees

  worst-case optimal incremental maintenance [ICDT'19a, PODS'20]

  evaluation of queries with negated relations of bounded degree [ICDT'19b]

- Functional dependencies $\rightarrow$ Learn simpler, equivalent models

  reparameterisation of polynomial regression models and factorisation machines

  [PODS'18,TODS'20]

Factorised Query Evaluation

$\Downarrow$

Time/Size Improvement

A Burgers & Hotdogs Use Case

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

A Burgers & Hotdogs Use Case

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

Consider the natural join of the above relations:

O(customer, day, dish), D(dish, item), I(item, price)

customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Monday	burger	onion	2
Elise	Monday	burger	bun	2
Elise	Friday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2
...	...	...	...	...

Burgers & Hotdogs in Relational Algebra

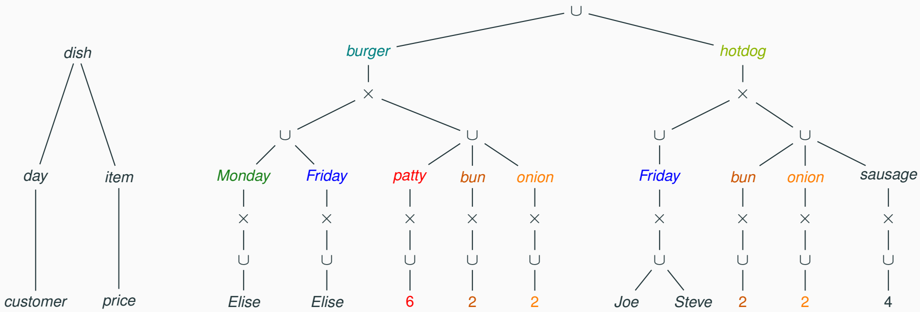
O(customer, day, dish), D(dish, item), I(item, price)

customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Monday	burger	onion	2
Elise	Monday	burger	bun	2
Elise	Friday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2
...	...	...	...	...

An algebraic encoding uses product ($\times$), union ($\cup$), and values:

$$
\begin{array}{lllllllllll}
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{patty} & \times & 6 & \cup \\
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{onion} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{bun} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{patty} & \times & 6 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{onion} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{bun} & \times & 2 & \cup \ldots
\end{array}
$$

Factorised Join

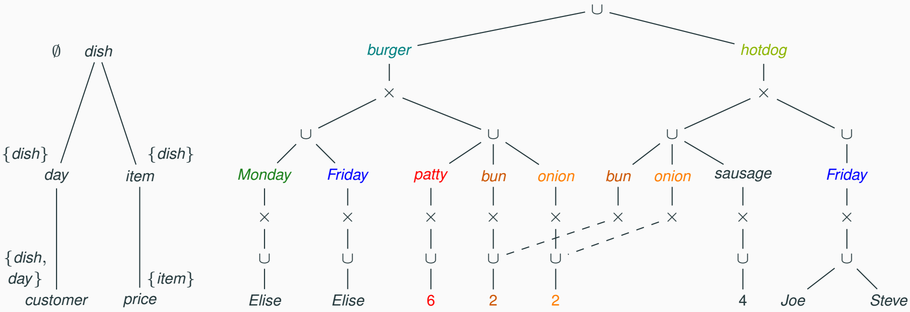


Variable order

Instantiation of the variable order over the input database

There are several algebraically equivalent factorised joins defined by distributivity of product over union and their commutativity.

... Now with Further Compression

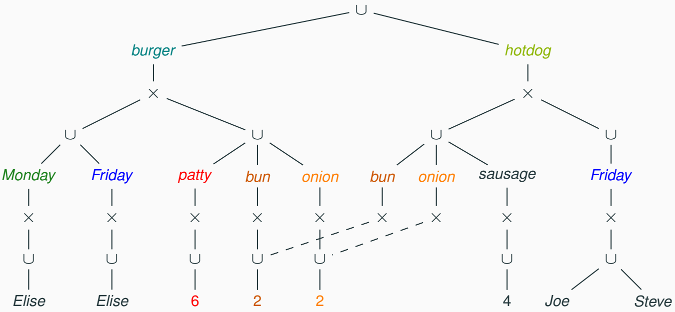


Observation:

- `price` is under `item`, which is under `dish`, but only *depends* on `item`,
- .. so the same `price` appears under an `item` *regardless* of the `dish`.

Idea: *Cache* `price` for a specific `item` and avoid repetition!

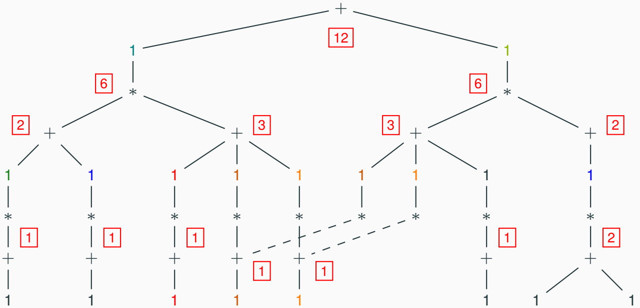
Factorised Aggregate Computation



`COUNT(*)` computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

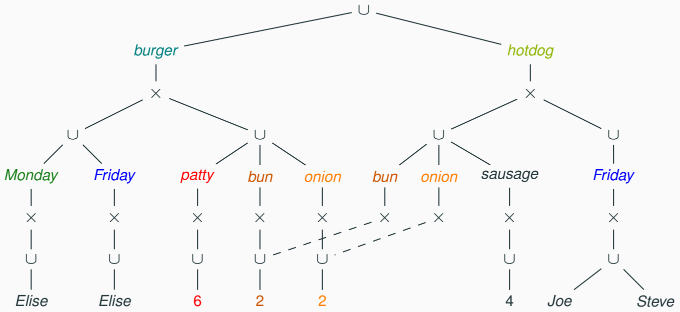
Factorised Aggregate Computation



COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

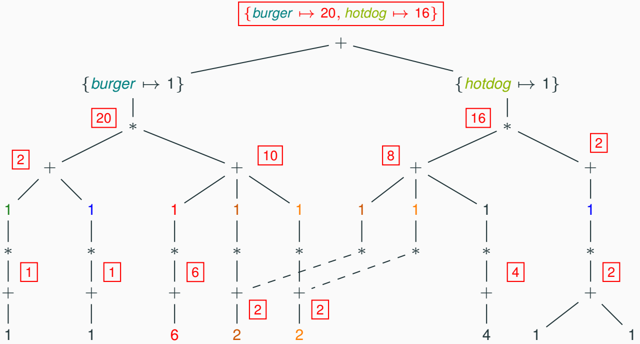
Factorising the Computation of Aggregates (2/2)



`SUM(price) GROUP BY dish` computed in one pass over the factorisation:

- All values except for `dish` & `price` $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Factorising the Computation of Aggregates (2/2)



SUM(price) GROUP BY dish computed in one pass over the factorisation:

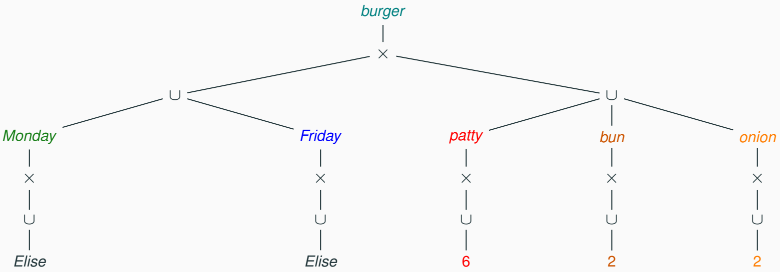
- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Sum-Product Ring Abstraction

$$\Downarrow$$

Sharing Aggregate Computation

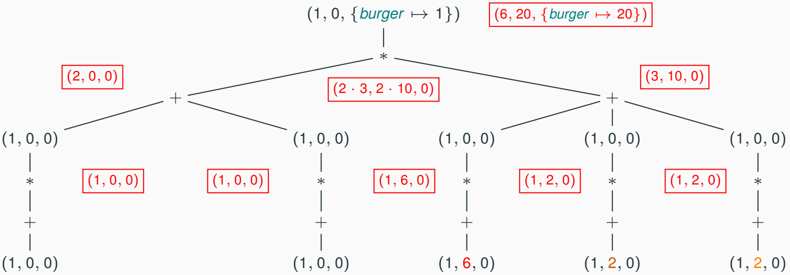
Shared Computation of Several Aggregates (1/2)



Ring for computing `SUM(1), SUM(price), SUM(price) GROUP BY dish`:

- Elements = triples, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates

Shared Computation of Several Aggregates (2/2)

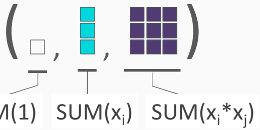


Ring for computing `SUM(1), SUM(price), SUM(price) GROUP BY dish`:

- Elements = triples, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates

Ring Generalisation for the Entire Covariance Matrix

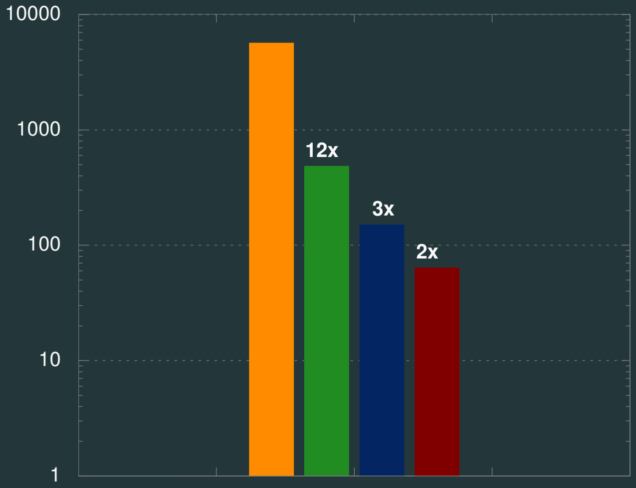
Ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$ over triples of aggregates $(c, \mathbf{s}, \mathbf{Q}) \in \mathcal{R}$:



$$
\begin{aligned}
(c_1, \mathbf{s}_1, \mathbf{Q}_1) + (c_2, \mathbf{s}_2, \mathbf{Q}_2) &= (c_1 + c_2, \mathbf{s}_1 + \mathbf{s}_2, \mathbf{Q}_1 + \mathbf{Q}_2) \\
(c_1, \mathbf{s}_1, \mathbf{Q}_1) * (c_2, \mathbf{s}_2, \mathbf{Q}_2) &= (c_1 \cdot c_2, c_2 \cdot \mathbf{s}_1 + c_1 \cdot \mathbf{s}_2, \\
&\qquad c_2 \cdot \mathbf{Q}_1 + c_1 \cdot \mathbf{Q}_2 + \mathbf{s}_1 \mathbf{s}_2^T + \mathbf{s}_2 \mathbf{s}_1^T) \\
\mathbf{0} &= (0, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n}) \\
\mathbf{1} &= (1, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n})
\end{aligned}
$$

- $\texttt{SUM(1)}$ reused for all $\texttt{SUM}(x_i)$ and $\texttt{SUM}(x_i * x_j)$
- $\texttt{SUM}(x_i)$ reused for all $\texttt{SUM}(x_i * x_j)$

Idea 3: Lower the Constant Factors



Engineering Tools of a Database Researcher

1. Specialisation for workload and data

   Generate code specific to the query batch and dataset

   Improve cache locality for hot data path

2. Sharing low-level data access

   Aggregates decomposed into views over join tree
   Share data access across views with different output schemas

3. Parallelisation: multi-core (SIMD & distribution to come)

   Task and domain parallelism

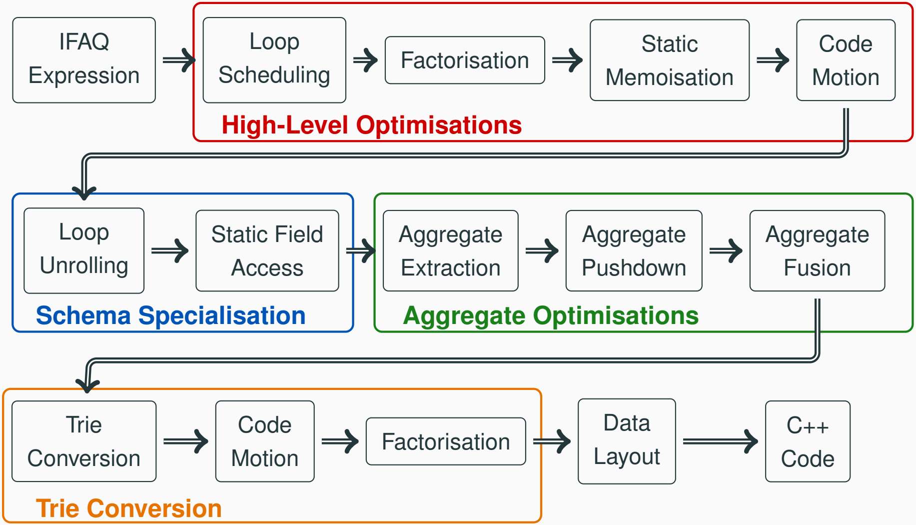
[DEEM'18,SIGMOD'19, CGO'20]

IFAQ: Iterative Functional Aggregate Queries

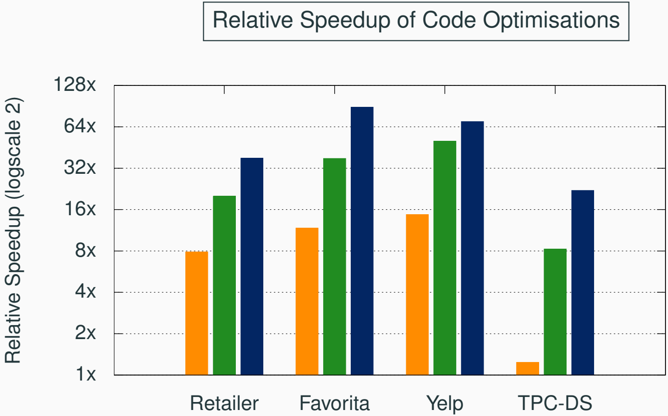
One DSL to Express both DB and ML Workloads! [CGO'20]

- Building blocks: Functional Aggregate Queries [PODS'16]
  - Formalism that expresses computation in databases, linear algebra, AI, logic
  - Relations are dictionaries
  - Sum-product computation over dictionaries
  - Conditionals using Kronecker delta
- Iteration constructs for
  - Stateful computation over collection elements
  - Constructing nested dictionaries

Transformation Steps for IFAQ Expressions



Engineering Tools of a Database Researcher



Added optimisations for covariance matrix computation:

specialisation → + sharing → + parallelisation

AWS d2.xlarge (4 vCPUs, 32GB)

Conclusion

Three-step Recipe for Learning over Relational Data

1. Turn the learning problem into a database problem
2. Exploit the problem structure to lower the complexity
3. Specialise and optimise the code to lower the constant factors

Q.E.D.

Relational Data Borg's Call to Arms

We need more sustained work on theory and systems for

Structure-aware Approaches to Data Analytics