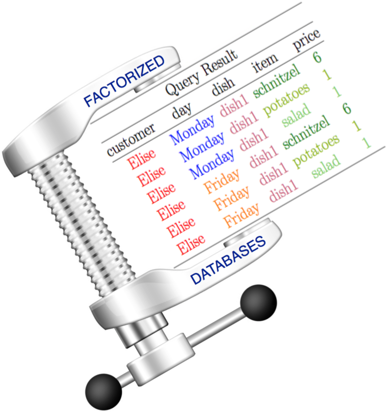
Learning Linear Regression Models over Factorized Joins



Maximilian Schleich

Dan Olteanu
Radu Ciucanu

University of Oxford

ACM SIGMOD **June 28, 2016**

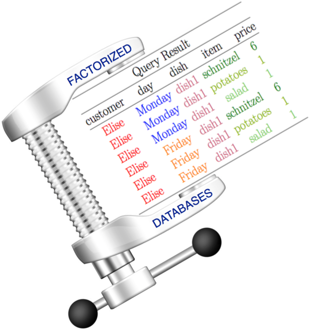
Goals of this Work

- Learn regression models over joins of large input tables.
  - Common analytics scenario in industry.
- Provide runtime guarantees for machine learning algorithms.
  - Ideally, achieve worst-case optimality.

Our Observations

- Join computation entails a high degree of redundancy, which can be avoided by factorized computation and representation.
  - We developed **worst-case optimal factorized join algorithms**. [TODS'15]
  - Factorized joins require **exponentially less time** than standard joins.
  - Aggregates (COUNT, SUM, MIN, MAX) can be computed in **one pass** over factorized data. [VLDB'13]
- Regression models can be learned in **linear time over factorized joins**.
  - This translates to **orders of magnitude performance improvements** over state of the art on real datasets.

Outline



What are Factorized Databases?

Building Regression Models at Speed

Complexity and Experiments

Factorized Databases by Example

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	sausage
hotdog	onion
hotdog	bun

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
...	...	...	...	...

Factorized Databases by Example

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
...	...	...	...	...

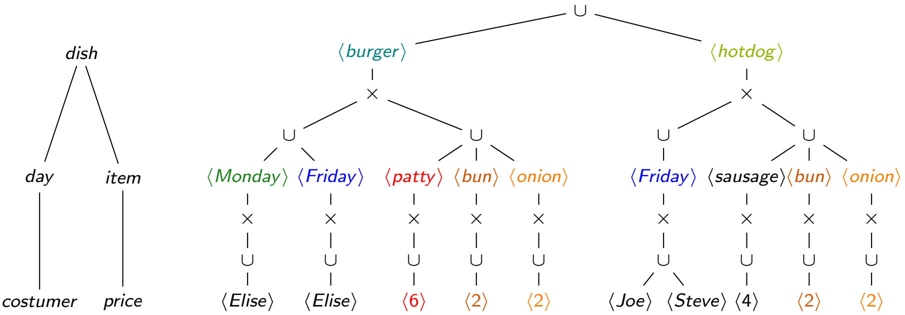
A *flat* relational algebra expression encoding the above query result is:

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

It uses relational product ($\times$), union ($\cup$), and singleton relations (e.g., $\langle 1 \rangle$).

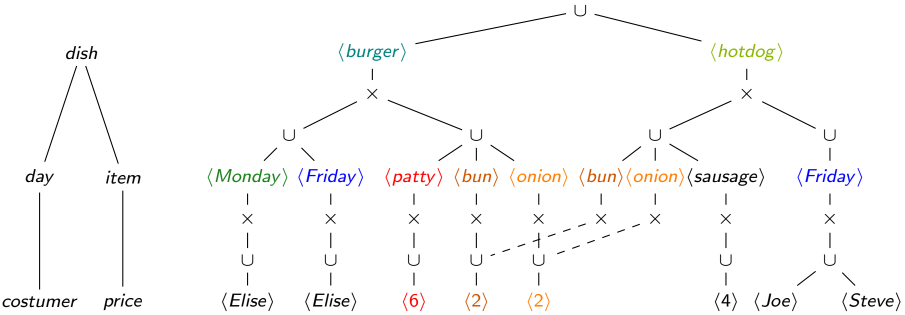
- The attribute names are not shown to avoid clutter.

Factorized Databases by Example

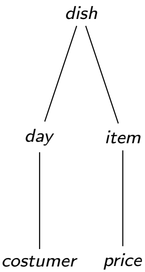


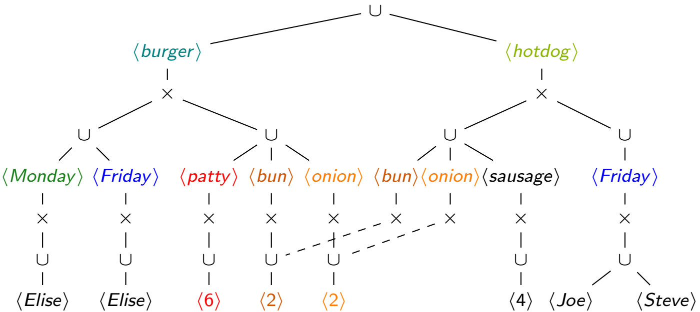
There are several *algebraically equivalent* factorized representations defined by distributivity of product over union and commutativity of product and union.

Factorized Databases by Example



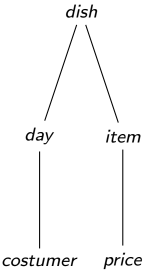
Factorized Databases by Example

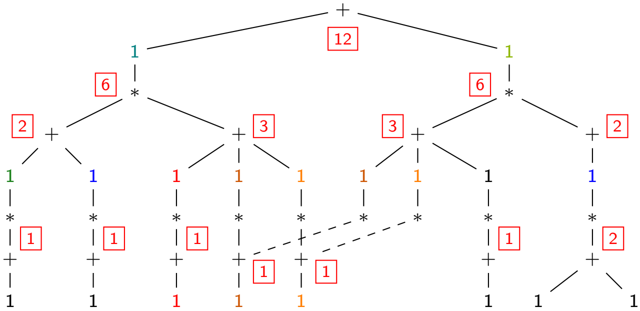




- COUNT(*):
  - values $\rightarrow$ 1,
  - $\cup \rightarrow +$,
  - $\times \rightarrow *$.

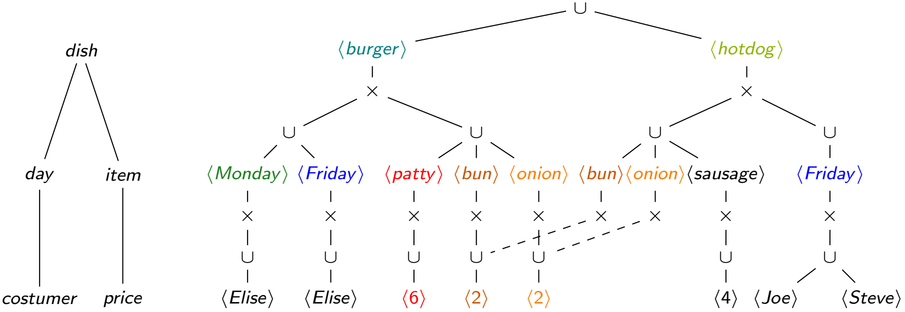
Factorized Databases by Example





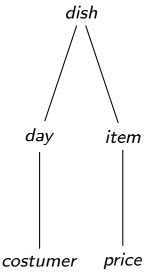
- COUNT(*):
  - values $\rightarrow 1$,
  - $\cup \rightarrow +$,
  - $\times \rightarrow *$.

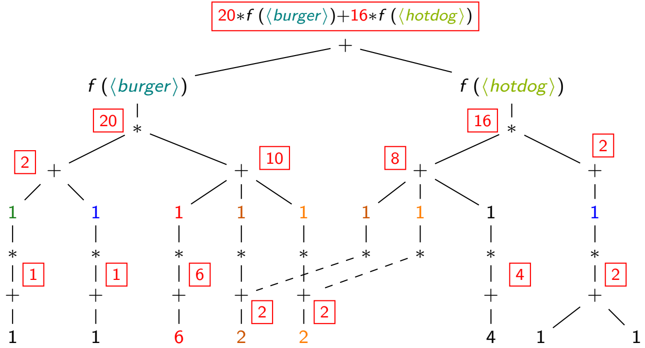
Factorized Databases by Example



- SUM(dish * price):
  - Assume there is a function f that turns dish into reals.
  - All values except for dish & price $\rightarrow 1$,
  - $\cup \rightarrow +$,
  - $\times \rightarrow *$.

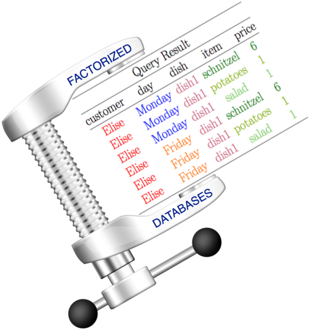
Factorized Databases by Example





- SUM(dish * price):
  - Assume there is a function f that turns dish into reals.
  - All values except for dish & price $\to 1$,
  - $\cup \to +$,
  - $\times \to *$.

Outline



What are Factorized Databases?

Building Regression Models at Speed

Complexity and Experiments

Building Regression Models at Speed

We learn regression models with an iterative optimization method.

Building Regression Models at Speed

We learn regression models with an iterative optimization method.

Building regression models in two steps.

1. **data-dependent computation**
   - Defined by set of aggregates of the form sum(X*Y) like in our example.
   - These aggregates can be done in one pass over the factorized join.
   - The redundancy in the flat data is not necessary for learning!

2. **data-independent convergence**
   - Parameter convergence step on top of the aggregate set

Building Regression Models at Speed

We learn regression models with an iterative optimization method.

Building regression models in two steps.

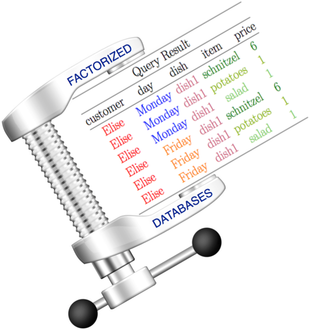
1. **data-dependent computation**
   - Defined by set of aggregates of the form sum(X*Y) like in our example.
   - These aggregates can be done in one pass over the factorized join.
   - The redundancy in the flat data is not necessary for learning!

2. **data-independent convergence**
   - Parameter convergence step on top of the aggregate set

System **F** for learning regression models over joins of large input tables.

- Three flavors to compute the aggregates:
  1. over the factorized join,
  2. on the fly, over a non-materialized factorized join,
  3. or in one optimized SQL query.

Outline



What are Factorized Databases?

Building Regression Models at Speed

Complexity and Experiments

Complexity of **F**

For a given join query Q over any database $\mathbf{D}$,
the factorized join can be computed in time $O(|\mathbf{D}|^{fhtw(Q)})$. [TODS'15]

- $fhtw(Q)$ is the fractional hypertree width of Q.

Aggregates can be computed in linear time over the factorized join. [VLDB'13]

For a training dataset defined by a join query Q over any database $\mathbf{D}$, **F** learns any linear regression model in time $O(\lvert\mathbf{D}\rvert^{fhtw(Q)})$.

For (α-)acyclic joins, $fhtw = 1$ and **F** learns in optimal time.

Complexity of **F**

For a given join query Q over any database $\mathbf{D}$,
the factorized join can be computed in time $O(|\mathbf{D}|^{\mathit{fhtw}(Q)})$. [TODS'15]

- *fhtw*(Q) is the fractional hypertree width of Q.

Aggregates can be computed in linear time over the factorized join. [VLDB'13]

For a training dataset defined by a join query Q over any database $\mathbf{D}$, **F** learns any linear regression model in time $O(|\mathbf{D}|^{\mathit{fhtw}(Q)})$.

For (α-)acyclic joins, *fhtw* $= 1$ and **F** learns in optimal time.

Worst-case optimal algorithm for flat joins needs time $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]

- ρ^* is the fractional edge cover number of Q.

$$\boxed{1 \leq \underbrace{\mathit{fhtw}(Q) \leq \rho^*(Q)}_{\text{up to } |Q|\text{, the number of relations joined in } Q} \leq |Q|}$$

This gap translates to orders of magnitude performance speedups in practice.

Experimental Setup

We benchmark **F** against

- **R** (QR-decomp.),
- **P**ython StatsModels (ols),
- and **MA**Dlib (glm, ols).

We use FDB and PostgreSQL to compute the factorized and respectively flat joins. Aggregates in **F/SQL** are computed in PostgreSQL.

US Retailer (real):

- Three tables: Inventory, Census, and Location.
- Regression model predicts the amount of inventory units.

LastFM (real and public):

- Three tables.
- Regression model predicts how often a user would listen to an artist based on similar information for its friends.

F versus **R**, **P**ython StatsModels and **M**ADlib

		US retailer L	US retailer N_1	LastFM L_1	LastFM L_2
# parameters		31	33	6	10
Size	Factorized	97,134,675	97,134,675	376,402	315,818
	Flat	2,585,046,352	2,585,046,352	369,986,292	590,793,800
	Compression	26.61×	26.61×	26.61×	982.86×
Join Time	Fact. (FDB)	36.03	36.03	4.79	9.94
	Flat (PSQL)	249.41	249.41	54.25	61.33
Import Time	R	1189.12*	1189.12*	155.91	276.77
	P	1164.40*	1164.40*	179.16	328.97
Learn Time	**F/FDB**	9.69	9.82	0.53	0.89
	M (glm)	2671.88	2937.49	572.88	746.50
	R	810.66*	873.14*	268.04	466.52
	P	1199.50*	1277.10*	35.74	148.84
Total Time	**F**	16.29	16.56	0.11	0.25
	F/FDB	45.72	45.85	5.32	10.83
	F/SQL	108.81	109.02	0.58	2.00
	M (ols)	680.60	737.02	152.37	196.60
	M (glm)	2921.29	3186.90	627.13	807.83
	R	2249.19*	2311.67*	478.20	804.62
	P	2613.31*	2690.91*	269.15	539.14
Speedup	**F** vs. M (ols)	41.78×	44.51×	1385.18×	786.40×
	F vs. M (glm)	179.33×	192.45×	5701.18×	3231.32×
	F vs. R	138.07×	139.59×	4347.27×	3218.48×
	F vs. P	160.42×	162.49×	2446.82×	2156.56×