Joins → Aggregates → Optimization

https://fdbresearch.github.io

Dan Olteanu

PhD Open School
University of Warsaw
November 22, 2018

Acknowledgements

Some work reported in this course has been done in the context of the FDB project, LogicBlox, and RelationalAI by

- Zavodný, Schleich, Kara, Nikolic, Zhang, Ciucanu, and Olteanu (Oxford)
- Abo Khamis and Ngo (RelationalAI), Nguyen (U. Michigan)

Some of the following slides are derived from presentations by

- Abo Khamis (optimization diagrams)
- Kara (covers, IVM$^\epsilon$, and many graphics)
- Ngo (functional aggregate queries)
- Schleich (performance and quizzes)

Lastly, Kara and Schleich proofread the slides.

I would like to thank them for their support!

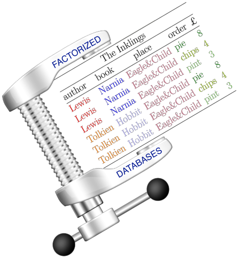
Goal of This Course

Introduction to a principled approach to in-database computation

This course starts where mainstream databases courses finish.

- **Part 1: Joins**
  - Basic building blocks in query languages. Studied extensively.
  - Systematic study of redundancy in the computation and representation of join results [OZ12,OZ15,KO18]
  - Worst-case optimal join algorithms [NPRR12,NRR13,V14,OZ15,ANS17]
- Part 2: Aggregates
- Part 3: Optimization

Outline of Part 1: Joins



Introduction by Examples

Decompositions and Variable Orders

Size Bounds for Join Results

Worst-Case Optimal Join Algorithms

Further Work and References

Quiz

Join Queries

$$\underbrace{Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n)}_{\text{head}} = \underbrace{R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)}_{\text{body}}$$

- Query variables: $\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n$. *All variables in the body occur in the head.*
- Relational atoms: $R_1, \ldots, R_n$
- Natural join: Same variable occurs in different relational atoms

Examples of bodies of queries used in the following slides:

- Path: $O(customer, day, dish), D(dish, item), I(item, price)$
- Path: $R_1(A, B), R_2(B, C), R_3(C, D)$
- Acyclic: $R(A, B, C), S(A, B, D), T(A, E), U(E, F)$.
- Triangle: $R_1(A, B), R_2(A, C), R_3(B, C)$
- Loop: $R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$

Join Example: Itemized Customer Orders

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

Join Example: Listing the Triangles in the Database

R_1

A	B
a_0	b_0
a_0	$\dots$
a_0	b_m
a_1	b_0
$\dots$	b_0
a_m	b_0

R_2

A	C
a_0	c_0
a_0	$\dots$
a_0	c_m
a_1	c_0
$\dots$	c_0
a_m	c_0

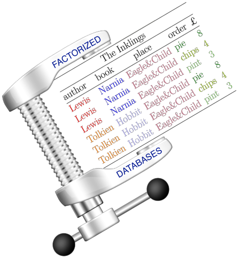
R_3

B	C
b_0	c_0
b_0	$\dots$
b_0	c_m
b_1	c_0
$\dots$	c_0
b_m	c_0

$R_1(A, B), R_2(A, C), R_3(B, C)$

A	B	C
a_0	b_0	c_0
a_0	b_0	$\dots$
a_0	b_0	c_m
a_0	b_1	c_0
a_0	$\dots$	c_0
a_0	b_m	c_0
a_1	b_0	c_0
$\dots$	b_0	c_0
a_1	b_0	c_0

Outline of Part 1: Joins



Introduction by Examples

Decompositions and Variable Orders

Size Bounds for Join Results

Worst-Case Optimal Join Algorithms

Further Work and References

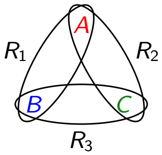
Quiz

Join Hypergraphs

We associate a (multi)hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with every join query Q

- Each variable in Q is a node in $\mathcal{V}$
- The set of variables of each relation symbol in Q is a (hyper)edge in $\mathcal{E}$

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$



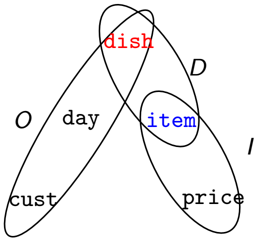
- $\mathcal{V} = \{A, B, C\}$
- $\mathcal{E} = \{\{A, B\}, \{A, C\}, \{B, C\}\}$

Join Hypergraphs

We associate a (multi)hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with every join query Q

- Each variable in Q is a node in $\mathcal{V}$
- The set of variables of each relation symbol in Q is a (hyper)edge in $\mathcal{E}$

Example: Order query $O(\texttt{cust}, \texttt{day}, \texttt{dish}), D(\texttt{dish}, \texttt{item}), I(\texttt{item}, \texttt{price})$



- $\mathcal{V} = \{\texttt{cust}, \texttt{day}, \texttt{dish}, \texttt{item}, \texttt{price}\}$
- $\mathcal{E} = \{\{\texttt{cust}, \texttt{day}, \texttt{dish}\}, \{\texttt{dish}, \texttt{item}\}, \{\texttt{item}, \texttt{price}\}\}$

Hypertree Decompositions

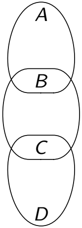
Definition[GLS99]: A (hypertree) decomposition $\mathcal{T}$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query Q is a pair (T, χ), where

- T is a tree
- χ is a function mapping each node in T to a subset of $\mathcal{V}$ called *bag*.

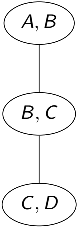
Properties of a decomposition $\mathcal{T}$:

- *Coverage*: $\forall e \in \mathcal{E}$, there must be a node $t \in T$ such that $e \subseteq \chi(t)$.
- *Connectivity*: $\forall v \in \mathcal{V}$, $\{t \mid t \in T, v \in \chi(t)\}$ forms a connected subtree.

The hypergraph of the query
$R_1(A, B), R_2(B, C), R_3(C, D)$



A hypertree decomposition



Hypertree Decompositions

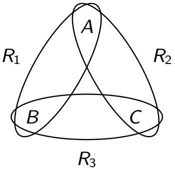
Definition[GLS99]: A (hypertree) decomposition $\mathcal{T}$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query Q is a pair (T, χ), where

- T is a tree
- χ is a function mapping each node in T to a subset of $\mathcal{V}$ called *bag*.

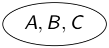
Properties of the decomposition $\mathcal{T}$:

- *Coverage*: $\forall e \in \mathcal{E}$, there must be a node $t \in T$ such that $e \subseteq \chi(t)$.
- *Connectivity*: $\forall v \in \mathcal{V}$, $\{t \mid t \in T, v \in \chi(t)\}$ forms a connected subtree.

The hypergraph of the triangle query
$R_1(A, B), R_2(A, C), R_3(B, C)$



A hypertree decomposition



Variable Orders

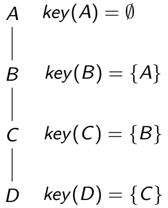
Definition[OZ15]: A *variable order* Δ for a query Q is a pair (F, key), where

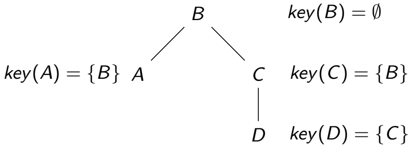
- F is a rooted forest with one node per variable in Q
- key is a function mapping each variable A to a subset of its ancestor variables in F.

Properties of a variable order Δ for Q:

- For each relation symbol, its variables lie along the same root-to-leaf path in F. For any such variables A and B, $A \in key(B)$ if A is an ancestor of B.
- For every child B of A, $key(B) \subseteq key(A) \cup \{A\}$.

Possible variable orders for the path query $R_1(A, B), R_2(B, C), R_3(C, D)$:





Variable Orders

Definition[OZ15]: A *variable order* Δ for a query Q is a pair (F, key), where

- F is a rooted forest with one node per variable in Q
- *key* is a function mapping each variable A to a subset of its ancestor variables in F.

Properties of a variable order Δ for Q:

- For each relation symbol, its variables lie along the same root-to-leaf path in F. For any such variables A and B, $A \in key(B)$ if A is an ancestor of B.
- For every child B of A, $key(B) \subseteq key(A) \cup \{A\}$.

Possible variable orders for the triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$:

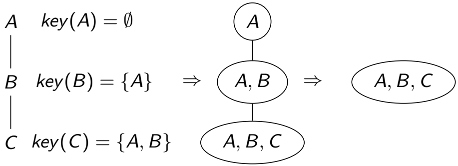
A $\quad key(A) = \emptyset$	B $\quad key(B) = \emptyset$	C $\quad key(C) = \emptyset$
B $\quad key(B) = \{A\}$	A $\quad key(A) = \{B\}$	B $\quad key(B) = \{C\}$
C $\quad key(C) = \{A, B\}$	C $\quad key(C) = \{A, B\}$	A $\quad key(A) = \{B, C\}$

Hypertree Decompositions ⇔ Variable Orders

From variable order Δ to hypertree decomposition $\mathcal{T}$: [OZ15]

- For each node A in Δ, create a bag $key(A) \cup \{A\}$.
- The bag for A is connected to the bags for its children and parent.
- Optionally, remove redundant bags

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

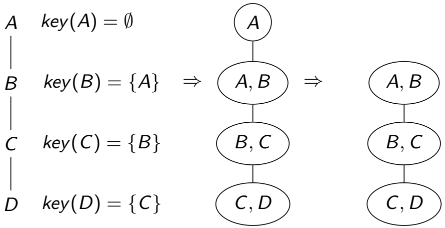


Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From variable order Δ to hypertree decomposition $\mathcal{T}$: [OZ15]

- For each node A in Δ, create a bag $key(A) \cup \{A\}$.
- The bag for A is connected to the bags for its children and parent.
- Optionally, remove redundant bags

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

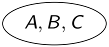


Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

$A \quad key(A) = \emptyset$

Step 1:
A is removed from $\mathcal{T}$
and inserted into Δ

$(A, B, C) \Rightarrow$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

Step 2:

B is removed from $\mathcal{T}$

and inserted into Δ

$A, B, C \quad \Rightarrow$

$A \quad key(A) = \emptyset$

$B \quad key(B) = \{A\}$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

Step 3:

C is removed from $\mathcal{T}$

and inserted into Δ

$A, B, C \quad \Rightarrow$

$A \quad key(A) = \emptyset$

$B \quad key(B) = \{A\}$

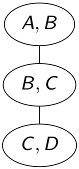
$C \quad key(C) = \{A, B\}$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$



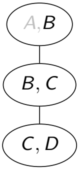
Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

Step 1:
A is removed from $\mathcal{T}$
and inserted into Δ



$\Rightarrow$

$A \quad key(A) = \emptyset$

Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

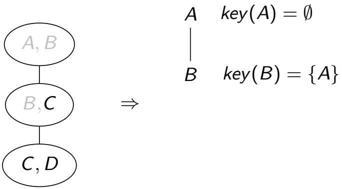
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

Step 2:

B is removed from $\mathcal{T}$

and inserted into Δ



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

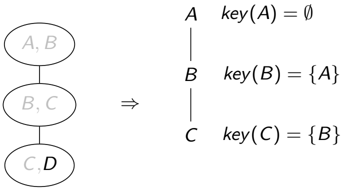
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

Step 3:

C is removed from $\mathcal{T}$

and inserted into Δ



Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order Δ: [OZ15]

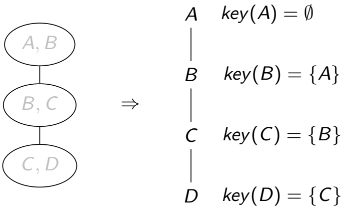
- Create a node A in Δ for a variable A in the top bag in $\mathcal{T}$
- Recurse with $\mathcal{T}$ where A is removed from all bags in $\mathcal{T}$.
- If top bag empty, then recurse independently on each of its child bags and create children of A in Δ
- Update *key* for each variable at each step.

Example: Path query $R_1(A, B), R_2(B, C), R_3(C, D)$

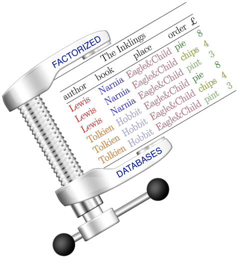
Step 4:

D is removed from $\mathcal{T}$

and inserted into Δ



Outline of Part 1: Joins



How Can We Bound the Size of the Join Result?

Example: the path query $R_1(A, B), R_2(B, C), R_3(C, D)$

- Assumption: All relations have size N.

- The query result is included in the result of $R_1(A, B), R_3(C, D)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$
  - All variables are "covered" by the relations R_1 and R_3

- There are databases for which the result size is at least N^2
  - Let $R_1 = [N] \times \{1\}, R_2 = \{1\} \times [N], R_3 = [N] \times \{1\}$.

How Can We Bound the Size of the Join Result?

Example: the path query $R_1(A, B), R_2(B, C), R_3(C, D)$

- Assumption: All relations have size N.

- The query result is included in the result of $R_1(A, B), R_3(C, D)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$
  - All variables are "covered" by the relations R_1 and R_3

- There are databases for which the result size is at least N^2
  - Let $R_1 = [N] \times \{1\}, R_2 = \{1\} \times [N], R_3 = [N] \times \{1\}$.

- Conclusion: Size of the query result is $\Theta(N^2)$ for some input classes

How Can We Bound the Size of the Join Result?

Example: the triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

- Assumption: All relations have size N.

- The query result is included in the result of $R_1(A, B), R_3(B, C)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$
  - All variables are "covered" by the relations R_1 and R_3

- There are databases for which the result size is at least N
  - Let $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supseteq \{(1, 1)\}$

How Can We Bound the Size of the Join Result?

Example: the triangle query $R_1(A, B), R_2(A, C), R_3(B, C)$

- Assumption: All relations have size N.
- The query result is included in the result of $R_1(A, B), R_3(B, C)$
  - Its size is upper bounded by $N^2 = |R_1| \times |R_3|$
  - All variables are "covered" by the relations R_1 and R_3
- There are databases for which the result size is at least N
  - Let $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supseteq \{(1, 1)\}$
- Conclusion: Size gap between the N^2 upper bound and the N lower bound!

Question: Can we close this gap and give tight size bounds?

Edge Covers and Independent Sets

We can generalize the previous examples as follows:

For the size upper bound:

- Cover all nodes (variables) by k edges (relations) $\Rightarrow$ size $\leq N^k$.
- This is an edge cover of the query hypergraph!

For the size lower bound:

- m independent nodes $\Rightarrow$ construct database such that size $\geq N^m$.
- This is an independent set of the query hypergraph!

$$\mathsf{max}_m = |\mathrm{IndependentSet}(Q)| \leq |\mathrm{EdgeCover}(Q)| = \mathsf{min}_k$$

$$\boxed{\mathsf{max}_m \text{ and } \mathsf{min}_k \text{ do not necessarily meet!}}$$

Can we further refine this analysis?

The Fractional Edge Cover Number $\rho^*(Q)$

The two bounds meet if we take their fractional versions [AGM08]

- *Fractional* edge cover of Q with weight $k \Rightarrow$ size $\leq N^k$.
- *Fractional* independent set with weight $m \Rightarrow$ size $\geq N^m$.

By duality of linear programming:

$$\max_m = |\mathrm{FractionalIndependentSet}(Q)| = |\mathrm{FractionalEdgeCover}(Q)| = \min_k$$

- This is the fractional edge cover number $\rho^*(Q)$!

For query Q and database of size N, the query result has size $O(N^{\rho^*(Q)})$.

The Fractional Edge Cover Number $\rho^*(Q)$

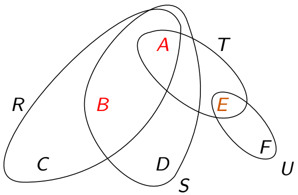
For a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$,
$\rho^*(Q)$ is the cost of an optimal solution to the linear program:

$$
\begin{array}{rll}
\text{minimize} & \sum_{i \in [n]} x_{R_i} & \\
\text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 & \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 & \forall i \in [n].
\end{array}
$$

- x_{R_i} is the weight of edge (relation) R_i in the hypergraph of Q
- Each node (variable) has to be covered by edges with sum of weights ≥ 1
- In the integer program variant for the edge cover, $x_{R_i} \in \{0, 1\}$

Example: Compute the Fractional Edge Cover (1/3)

Consider the join query Q: $R(A, B, C), S(A, B, D), T(A, E), U(E, F)$.



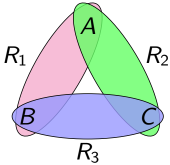
- The three edges R, S, U can cover all nodes.
  $\mathrm{FractionalEdgeCover}(Q) \leq 3$
- Each node C, D, and F must be covered by a distinct edge.
  $\mathrm{FractionalIndependentSet}(Q) \geq 3$

$\Rightarrow \rho^*(Q) = 3$

$\Rightarrow$ Size $\leq N^3$ and for some inputs is $\Theta(N^3)$.

Example: Compute the Fractional Edge Cover (2/3)

Consider the triangle query: $R_1(A, B), R_2(A, C), R_3(B, C)$.



$$
\begin{array}{lllllll}
\multicolumn{7}{l}{\textit{minimize } x_{R_1} + x_{R_2} + x_{R_3}} \\
\multicolumn{7}{l}{\textit{subject to}} \\
A: & x_{R_1} & + & x_{R_2} & & & \geq 1 \\
B: & x_{R_1} & & & + & x_{R_3} & \geq 1 \\
C: & & & x_{R_2} & + & x_{R_3} & \geq 1 \\
 & x_{R_1} \geq 0 & & x_{R_2} \geq 0 & & x_{R_3} \geq 0 &
\end{array}
$$

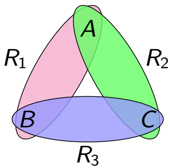
Our previous size upper bound was N^2:

- This is obtained by setting any two of $x_{R_1}, x_{R_2}, x_{R_3}$ to 1.

What is the fractional edge cover number for the triangle query?

Example: Compute the Fractional Edge Cover (2/3)

Consider the triangle query: $R_1(A, B), R_2(A, C), R_3(B, C)$.



$$
\begin{array}{lccccccc}
\multicolumn{8}{l}{\textit{minimize } x_{R_1} + x_{R_2} + x_{R_3}} \\
\multicolumn{8}{l}{\textit{subject to}} \\
A: & x_{R_1} & + & x_{R_2} & & & \geq 1 \\
B: & x_{R_1} & & & + & x_{R_3} & \geq 1 \\
C: & & & x_{R_2} & + & x_{R_3} & \geq 1 \\
 & x_{R_1} \geq 0 & & x_{R_2} \geq 0 & & x_{R_3} \geq 0 &
\end{array}
$$

Our previous size upper bound was N^2:

- This is obtained by setting any two of $x_{R_1}, x_{R_2}, x_{R_3}$ to 1.

What is the fractional edge cover number for the triangle query?

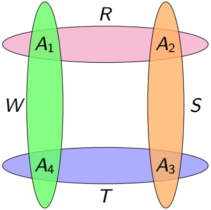
We can do better: $x_{R_1} = x_{R_2} = x_{R_3} = 1/2$. Then, $\rho^* = 3/2$.

Lower bound reaches $N^{3/2}$ for $R_1 = R_2 = R_3 = [\sqrt{N}] \times [\sqrt{N}]$.

Example: Compute the Fractional Edge Cover (3/3)

Consider the (4-cycle) join: $R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$.

The linear program for its fractional edge cover number:



$$
\begin{array}{lccccccccc}
\multicolumn{10}{l}{\textit{minimize } x_R + x_S + x_T + x_W} \\
\multicolumn{10}{l}{\textit{subject to}} \\
A_1: & x_R & & & & & + & x_W & \geq 1 \\
A_2: & x_R & + & x_S & & & & & \geq 1 \\
A_3: & & & x_S & + & x_T & & & \geq 1 \\
A_4: & & & & & x_T & + & x_W & \geq 1 \\
& x_R \geq 0 & & x_S \geq 0 & & x_T \geq 0 & & x_W \geq 0 &
\end{array}
$$

Possible solution: $x_R = x_T = 1$. Another solution: $x_S = x_W = 1$. Then, $\rho^* = 2$.

Lower bound reaches N^2 for $R = T = [N] \times \{1\}$ and $S = W = \{1\} \times [N]$.

Historical Note on the Fractional Edge Cover Number

Tight size bounds via $\rho*$ have been known from earlier works in other contexts:

- (special case) Loomis-Whitney inequality [LW49]
- (general case) number of occurrences of a subgraph in a graph [A81]
- generalization of Loomis-Whitney that subsumes the AGM bound [BT95]

Recent insightful travel through the history of this result [H18]

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Recall the linear program for computing the fractional edge cover number $\rho^*(Q)$ of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{array}{ll}
\text{minimize} & \sum_{i \in [n]} x_{R_i} \\[1em]
\text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\[1em]
& x_{R_i} \geq 0 \qquad\qquad\qquad\qquad \forall i \in [n].
\end{array}
$$

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{aligned}
\text{minimize} \quad & N^{\sum_{i \in [n]} x_{R_i}} \\
\text{subject to} \quad & \textstyle\sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 \qquad\qquad\qquad\qquad\qquad \forall i \in [n].
\end{aligned}
$$

Assumption: All relations have the same size N.

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$\begin{array}{rll} \text{minimize} & \prod_{i\in[n]} N^{x_i} & \\ \text{subject to} & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 & \forall A \in \bigcup_{j\in[n]} \boldsymbol{A}_j, \\ & x_{R_i} \geq 0 & \forall i \in [n]. \end{array}$$

Assumption: All relations have the same size N.

Refinement under Cardinality Constraints

Common case in practice:

- Relations have different sizes
- Small-size projections of relations may be added to the join query

Add relation sizes into the linear program that computes the result size of a join query $Q(\boldsymbol{A}_1 \cup \cdots \cup \boldsymbol{A}_n) = R_1(\boldsymbol{A}_1), \ldots, R_n(\boldsymbol{A}_n)$:

$$
\begin{aligned}
\text{minimize} \quad & \prod_{i\in[n]} N_i^{x_i} \\
\text{subject to} \quad & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j\in[n]} \boldsymbol{A}_j, \\
& x_{R_i} \geq 0 \qquad\qquad\qquad \forall i \in [n].
\end{aligned}
$$

Assumption: Relation R_i has size N_i, $\forall i \in [n]$.

Size Bounds for Factorized Representations of Join Results

Recall the Itemized Customer Orders 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	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
...	...	...	...	...

Factor Out Common Data Blocks

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

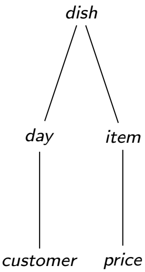
The listing representation of the above query result is:

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

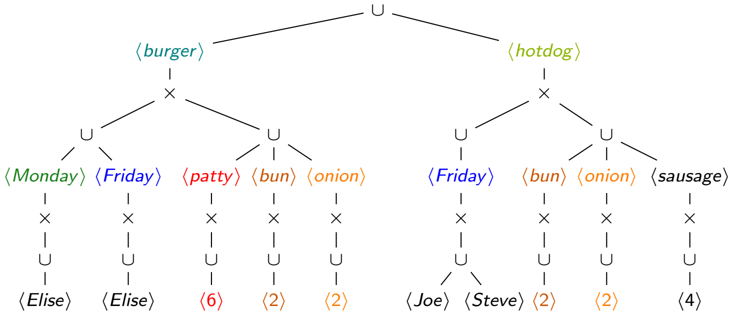
It uses relational product ($\times$), union ($\cup$), and data (singleton relations).

- The attribute names are not shown to avoid clutter.

This is How A Factorized Join Looks Like!



Var order

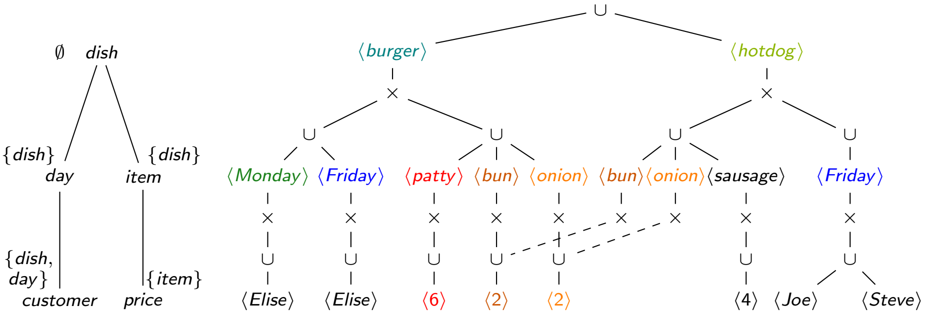


Factorized representation of the join result

There are several *algebraically equivalent* factorized representations defined:

- by distributivity of product over union and their commutativity;
- as groundings of variable orders.

.. Now with Further Compression using Caching

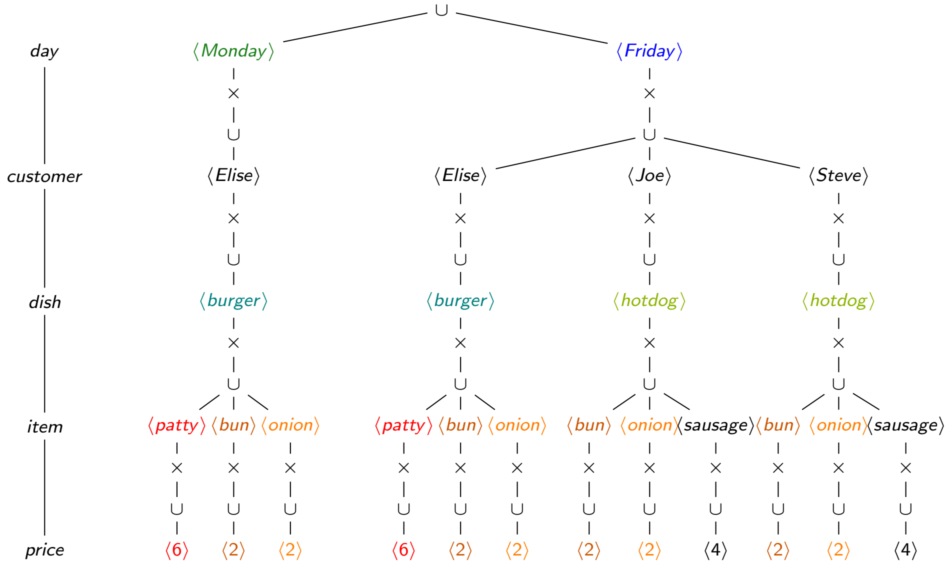


Observation:

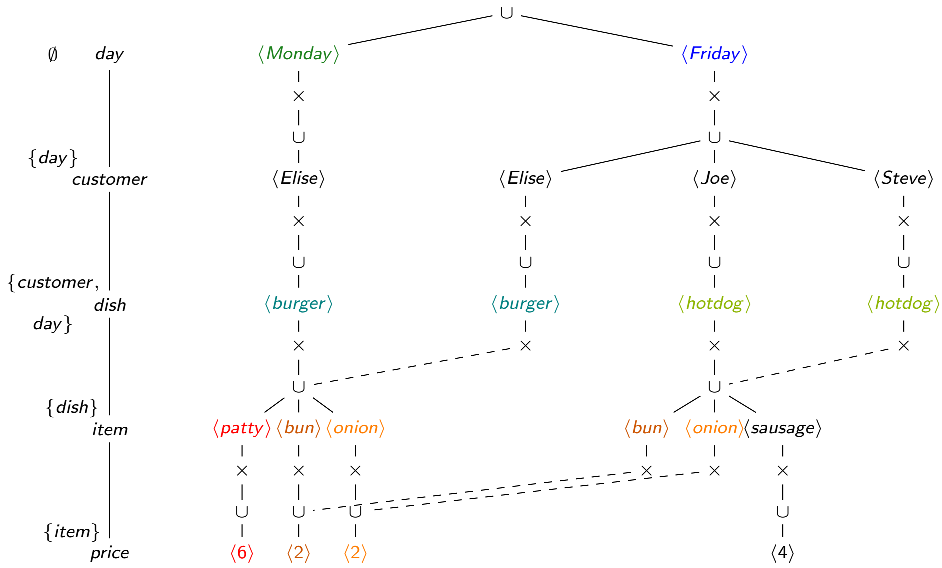
- `price` is under `item`, which is under `dish`, but only *depends* on `item`,
- .. so the same `price` appears under an `item` *regardless* of the `dish`.

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

Same Data, Different Factorization



.. and Further Compressed using Caching



Which factorization should we choose?

The *size* of a factorization is the number of its values.

Example:

$$
\begin{aligned}
F_1 =& \big(\langle 1 \rangle \cup \cdots \cup \langle n \rangle\big) \times \big(\langle 1 \rangle \cup \cdots \cup \langle m \rangle\big) \\
F_2 =& \langle 1 \rangle \times \langle 1 \rangle \cup \cdots \cup \langle 1 \rangle \times \langle m \rangle \\
& \qquad\qquad \cup \cdots \cup \\
& \quad \langle n \rangle \times \langle 1 \rangle \cup \cdots \cup \langle n \rangle \times \langle m \rangle .
\end{aligned}
$$

- F_1 is factorized, F_2 is a listing representation
- $F_1 \equiv F_2$
- **BUT** $|F_1| = m + n \ll |F_2| = m * n$.

How much space does factorization save over the listing representation?

Size Bounds for Join Results

Given a join query Q, for any database of size N, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [LW49,A81,BT95,AGM08]

Size Bounds for Join Results

Given a join query Q, for any database of size N, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [LW49,A81,BT95,AGM08]

- a factorization *without caching* of size $O(N^{s(Q)})$. [OZ12]

Size Bounds for Join Results

Given a join query Q, for any database of size N, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [LW49,A81,BT95,AGM08]
- a factorization *without caching* of size $O(N^{s(Q)})$. [OZ12]
- a factorization *with caching* of size $O(N^{fhtw(Q)})$. [OZ15]

Size Bounds for Join Results

Given a join query Q, for any database of size N, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [LW49,A81,BT95,AGM08]
- a factorization *without caching* of size $O(N^{s(Q)})$. [OZ12]
- a factorization *with caching* of size $O(N^{fhtw(Q)})$. [OZ15]

$$\boxed{1 \leq fhtw(Q) \underbrace{\leq}_{\text{up to } \log|Q|} s(Q) \underbrace{\leq}_{\text{up to } |Q|} \rho^*(Q) \leq |Q|}$$

- $|Q|$ is the number of relations in Q
- $\rho^*(Q)$ is the fractional edge cover number of Q
- $s(Q)$ is the factorization width of Q
- $fhtw(Q)$ is the fractional hypertree width of Q [M10]

Size Bounds for Join Results

Given a join query Q, for any database of size N, the join result admits

- a listing representation of size $O(N^{\rho^*(Q)})$. [LW49,A81,BT95,AGM08]

- a factorization *without caching* of size $O(N^{s(Q)})$. [OZ12]

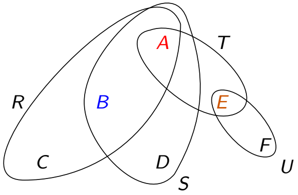
- a factorization *with caching* of size $O(N^{fhtw(Q)})$. [OZ15]

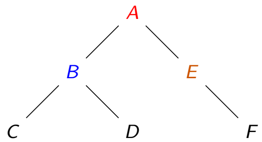
These size bounds are asymptotically tight!

- **Best possible size bounds** for factorized representations over variable orders of Q and for listing representation, *but not database optimal!*

  There exists arbitrarily large databases for which
  - the listing representation has size $\Omega(N^{\rho^*(Q)})$
  - the factorization with/without caching over *any variable order* of Q has size $\Omega(N^{s(Q)})$ and $\Omega(N^{fhtw(Q)})$ respectively.

Example: The Factorization Width s





The structure of the factorization over the above variable order Δ:

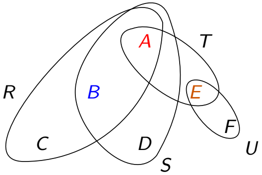
$$\bigcup_{a \in \mathbf{A}} \left(\langle a \rangle \times \bigcup_{b \in \mathbf{B}} \left(\langle b \rangle \times \Big(\bigcup_{c \in C} \langle c \rangle \Big) \times \Big(\bigcup_{d \in D} \langle d \rangle \Big) \right) \times \bigcup_{e \in \mathbf{E}} \left(\langle e \rangle \times \Big(\bigcup_{f \in F} \langle f \rangle \Big) \right) \right)$$

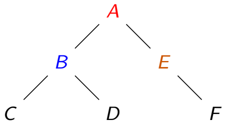
The number of values for a variable is dictated by the number of valid tuples of values for its ancestors in Δ:

- One value $\langle f \rangle$ for each tuple (a, e, f) in the join result.

Size of factorization = sum of sizes of results of **subqueries along paths**.

Example: The Factorization Width s





- The factorization width for Δ is the largest ρ^* over subqueries defined by root-to-leaf paths in Δ
- $s(Q)$ is the minimum factorization width over all variable orders of Q

In our example:

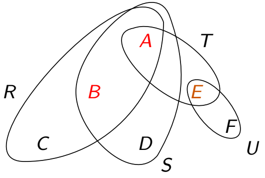
- Path A–E–F has fractional edge cover number 2.
  $\Rightarrow$ The number of F-values is $\leq N^2$, but can be $\sim N^2$.
- All other root-to-leaf paths have fractional edge cover number 1.
  $\Rightarrow$ The number of other values is $\leq N$.

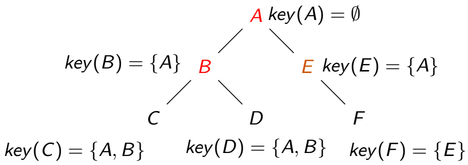
$s(Q) = 2$ $\Rightarrow$ Factorization size is $O(N^2)$

Recall that $\rho^*(Q) = 3$ $\Rightarrow$ Listing representation size is $O(N^3)$

Example: The Fractional Hypertree Width *fhtw*

Idea: Avoid repeating identical expressions, store them once and use pointers.





$$\bigcup_{a \in \mathbf{A}} \left[\langle a \rangle \times \cdots \times \bigcup_{e \in \mathbf{E}} \left(\langle e \rangle \times \left(\bigcup_{f \in F} \langle f \rangle\right)\right)\right]$$

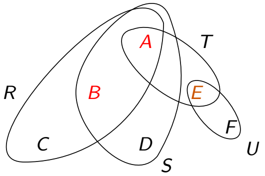
Observation:

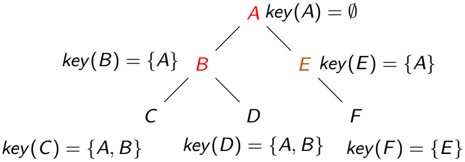
- Variable F only depends on $\mathbf{E}$ and not on $\mathbf{A}$: $key(F) = \{\mathbf{E}\}$
- A value $\langle e \rangle$ maps to the same union $\bigcup_{(e,f) \in U} \langle f \rangle$ regardless of its pairings with $\mathbf{A}$-values.

  $\Rightarrow$ Define $U_e = \bigcup_{(e,f) \in U} \langle f \rangle$ once for each value $\langle e \rangle$ and reuse it

Example: The Fractional Hypertree Width *fhtw*

Idea: Avoid repeating identical expressions, store them once and use pointers.





A factorization with caching would be:

$$\bigcup_{a \in \mathbf{A}} \left[\langle a \rangle \times \cdots \times \bigcup_{e \in \mathbf{E}} \left(\langle e \rangle \times U_e\right)\right]; \qquad \left\{ U_e = \bigcup_{(e,f) \in U} \langle f \rangle \right\}$$

- *fhtw* for Δ is the largest $\rho^*(Q_{key(X) \cup \{X\}})$ over subqueries $Q_{key(X) \cup \{X\}}$ defined by the variables $key(X) \cup \{X\}$ for each variable X in Δ
- $fhtw(Q)$ is the minimum *fhtw* over all variable orders of Q

In our example: $fhtw(Q) = 1 < s(Q) = 2 < \rho^*(Q) = 3$.

Alternative Characterizations of *fhtw*

The fractional hypertree width *fhtw* has been originally defined for hypertree decompositions. [M10]

- Given a join query Q.
- Let $\mathbf{T}$ be the set of hypertree decompositions of the hypergraph of Q.

$$\boxed{\mathit{fhtw}(Q) = \min_{(T,\chi)\in\mathbf{T}} \max_{n\in T} \rho^*(Q_{\chi(n)})}$$

Alternative Characterizations of *fhtw*

The fractional hypertree width *fhtw* has been originally defined for hypertree decompositions. [M10]

- Given a join query Q.
- Let $\mathbf{T}$ be the set of hypertree decompositions of the hypergraph of Q.

$$\boxed{fhtw(Q) = \min_{(T,\chi)\in\mathbf{T}} \max_{n\in T} \rho^*(Q_{\chi(n)})}$$

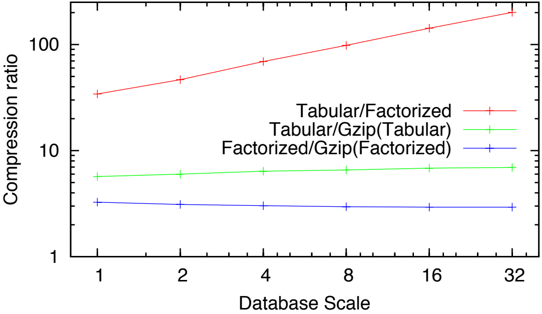
Alternative characterization of the fractional hypertree width *fhtw* using the mapping between hypertree decompositions and variable orders [OZ15]

- Given a join query Q.
- Let $\mathbf{VO}$ be the set of variable orders of Q.

$$\boxed{fhtw(Q) = \min_{(F,key)\in\mathbf{VO}} \max_{v\in F} \rho^*(Q_{key(v)\cup\{v\}})}$$

Compression by Factorization in Practice

Compression Contest: Factorized vs. Zipped Relations



Result of query $\text{Orders} \bowtie \text{Dish} \bowtie \text{Items}$ [BKOZ13]

- Tabular = listing representation in CSV text format
- Gzip (compression level 6) outputs binary format
- Factorized representation in text format (each digit takes one character)

Observations:

- Gzip does not exploit distant repetitions!
- Factorizations can be arbitrarily more succinct than gzipped relations.
- Gzipping factorizations improves the compression by 3x.

Factorization Gains in Practice (1/4)

Retailer dataset used for LogicBlox analytics

- Relations: Inventory (84M), Sales (1.5M), Clearance (368K), Promotions (183K), Census (1K), Location (1K).
- Compression factors (caching not used):
  - **26.61x** for natural join of Inventory, Census, Location.
  - **159.59x** for natural join of Inventory, Sales, Clearance, Promotions

Factorization Gains in Practice (2/4)

LastFM public dataset

- Relations: UserArtists (93K), UserFriends (25K), TaggedArtists (186K).
- Compression factors:
  - **143.54x** for joining two copies of Userartists and Userfriends

    With caching: **982.86x**
  - **253.34x** when also joining on TaggedArtists
  - **2.53x/ 3.04x/ 924.46x** for triangle/4-clique/bowtie query on UserFriends
  - **9213.51x/ 552Kx/ $\geq$86Mx** for versions of triangle/4-clique/bowtie queries with copies for UserArtists for each UserFriend copy

Factorization Gains in Practice (3/4)

Twitter public dataset

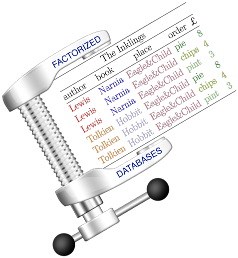
- Relation: Follower-Followee (1M)
- Compression factors:
  - **2.69x** for triangle query
  - **3.48x** for 4-clique query
  - **4918.73x** for bowtie query

Factorization Gains in Practice (4/4)

Yelp Dataset Challenge

- Relations: Business (174K), User (1.3M), Review (5.2M), Category(667K), Attribute (1.3M)
- Compression factors:
  - **39.43x** for natural join of Business, User, Review, Attribute (with caching)
  - **185.87x** for natural join of Business, User, Review, Attribute, Category (with caching)

Outline of Part 1: Joins



Introduction by Examples

Decompositions and Variable Orders

Size Bounds for Join Results

Worst-Case Optimal Join Algorithms

Further Work and References

Quiz

How Fast Can We Compute Join Results?

Given a join query Q, for any database of size N, the join result can be computed in time

- $O(N^{\rho^*(Q)})$ as listing representation [NPRR12,V14]
- $O(N^{s(Q)})$ as factorization *without caching* [OZ15]
- $O(N^{fhtw(Q)})$ as factorization *with caching* [OZ15]

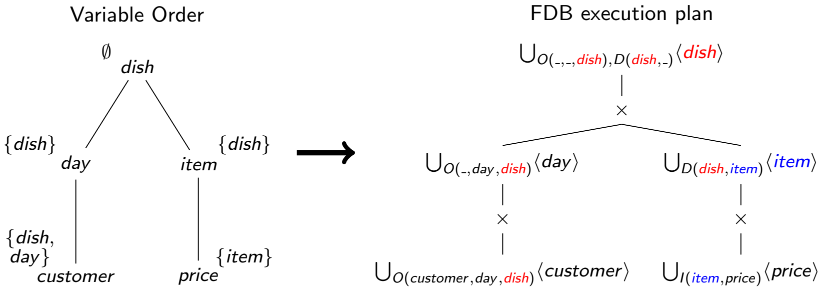
These upper bounds essentially follow the succinctness gap. They are:

- worst-case optimal (modulo $\log N$) within the given representation model
- with respect to *data* complexity
  - additional quadratic factor in the number of variables and linear factor in the number of relations in Q

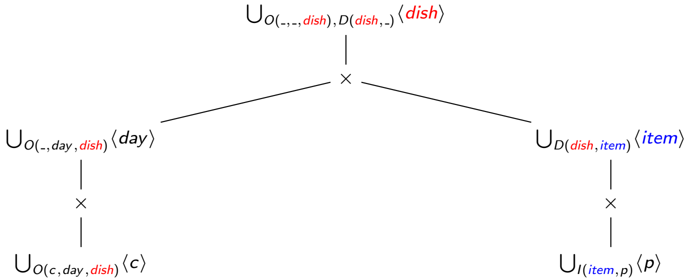
Example: Computing the Factorized Join Result with FDB

Our join: O(customer, day, dish), D(dish, item), I(item, price)

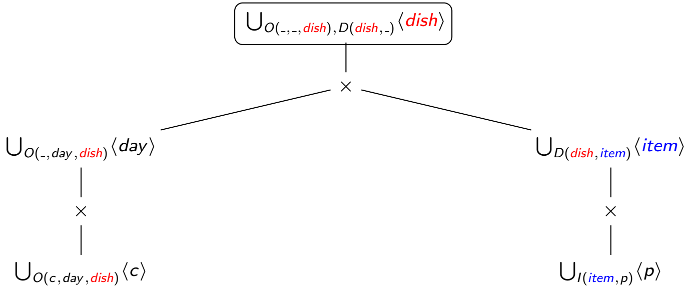
can be grounded to a factorized representation as follows:



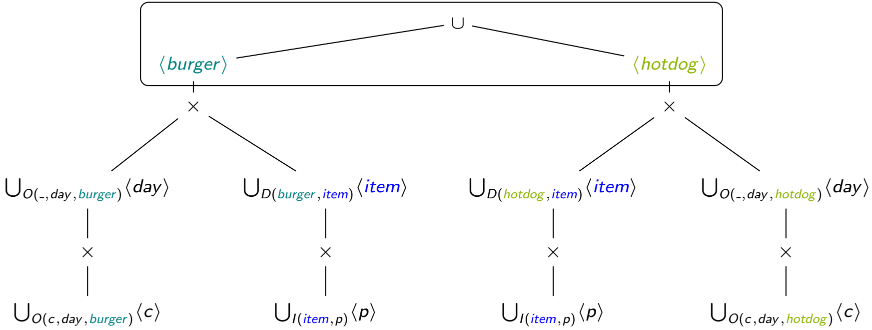
Example: Computing the Factorized Join Result with FDB



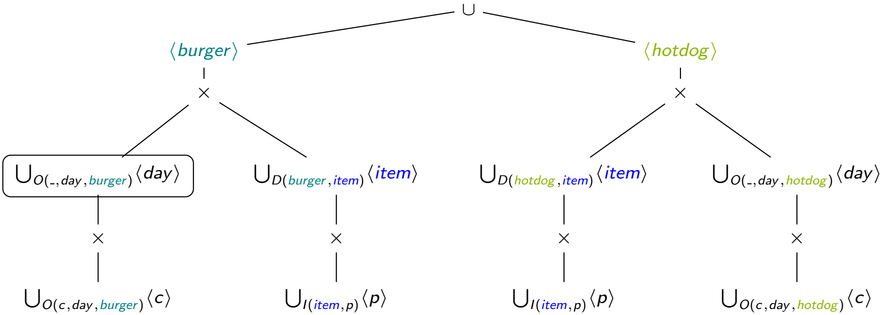
Example: Computing the Factorized Join Result with FDB



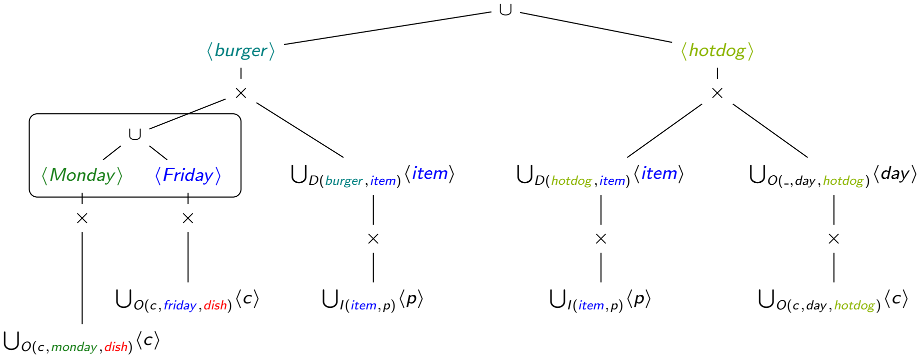
Example: Computing the Factorized Join Result with FDB



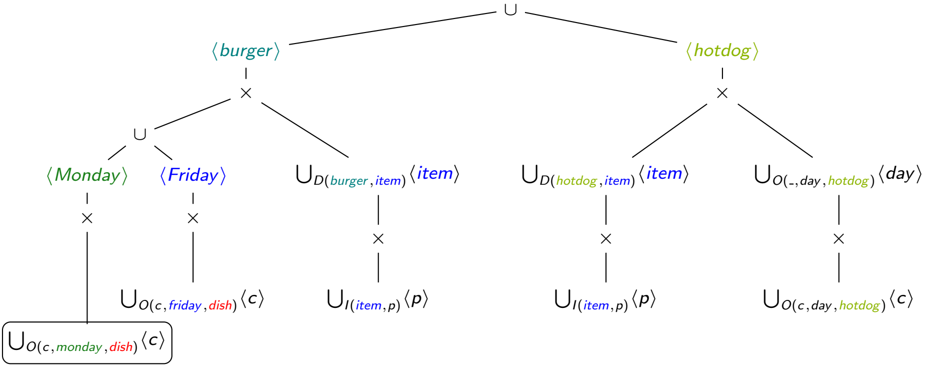
Example: Computing the Factorized Join Result with FDB



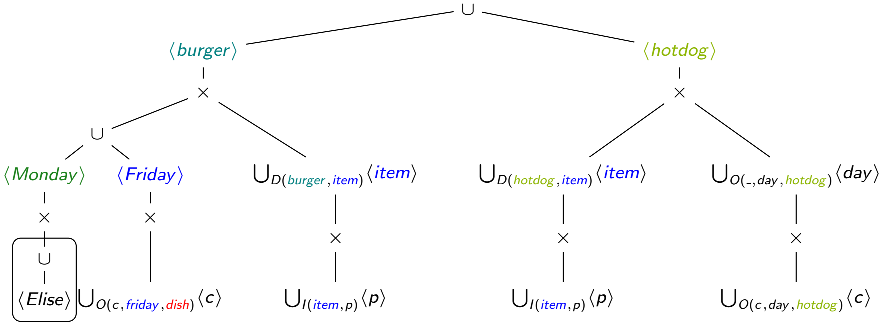
Example: Computing the Factorized Join Result with FDB



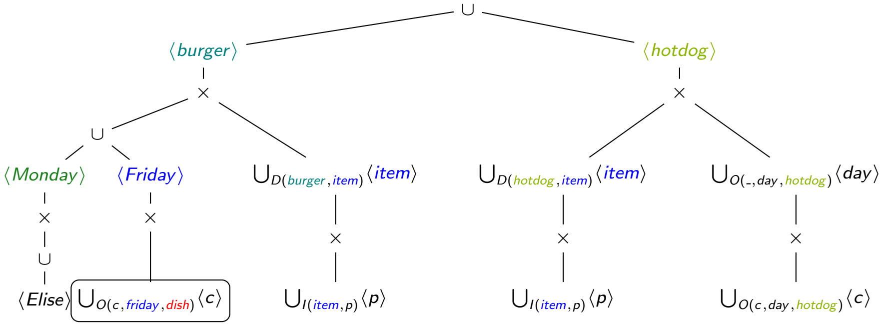
Example: Computing the Factorized Join Result with FDB



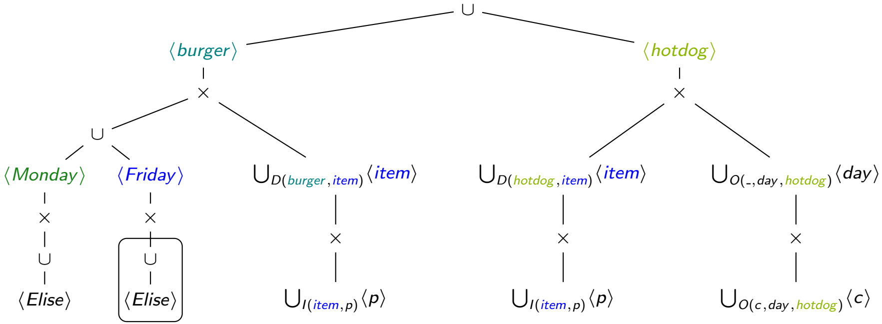
Example: Computing the Factorized Join Result with FDB



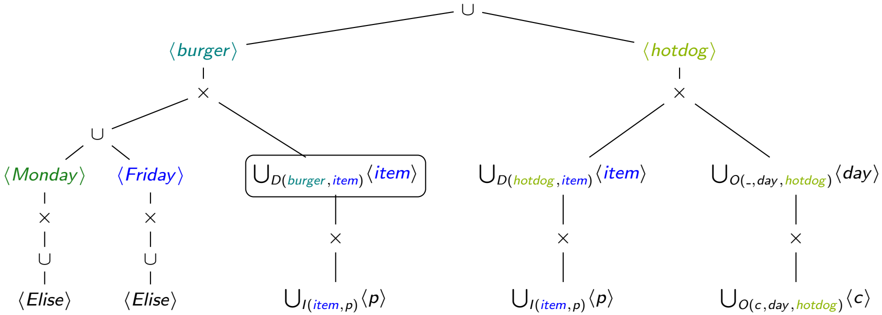
Example: Computing the Factorized Join Result with FDB



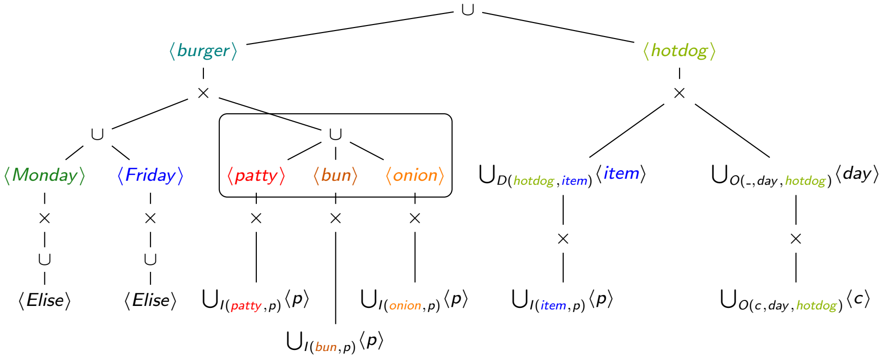
Example: Computing the Factorized Join Result with FDB



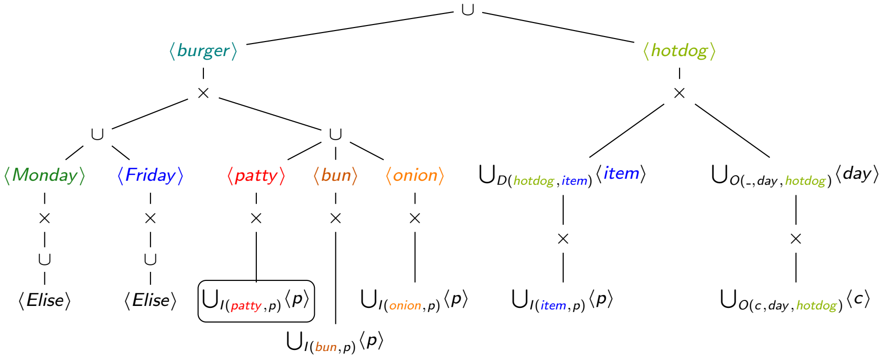
Example: Computing the Factorized Join Result with FDB



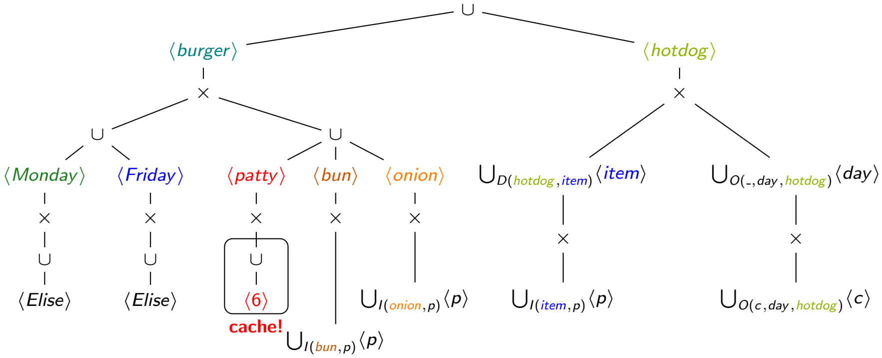
Example: Computing the Factorized Join Result with FDB



Example: Computing the Factorized Join Result with FDB

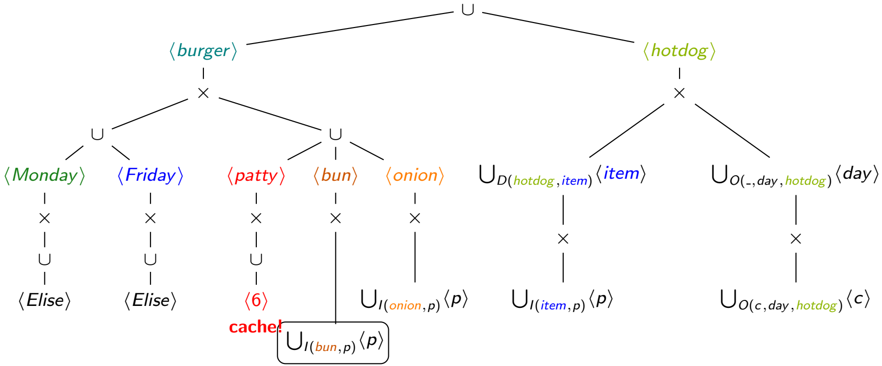


Example: Computing the Factorized Join Result with FDB



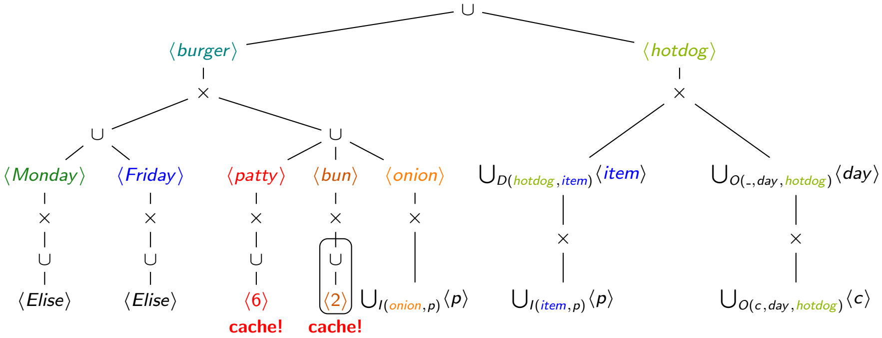
- price depends on item, but not on dish.
  Cache prices for specific items!

Example: Computing the Factorized Join Result with FDB



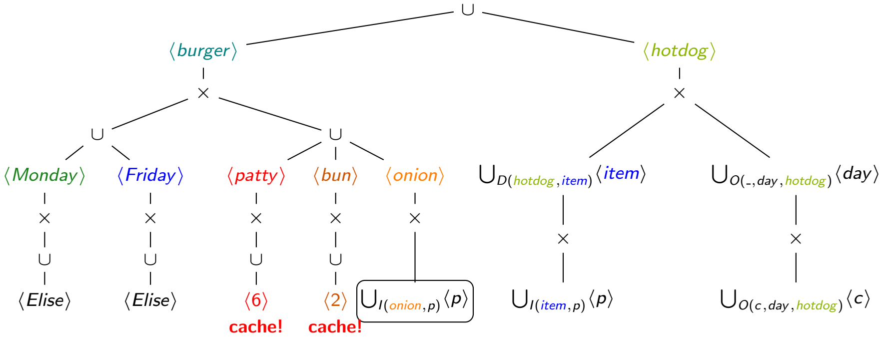
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



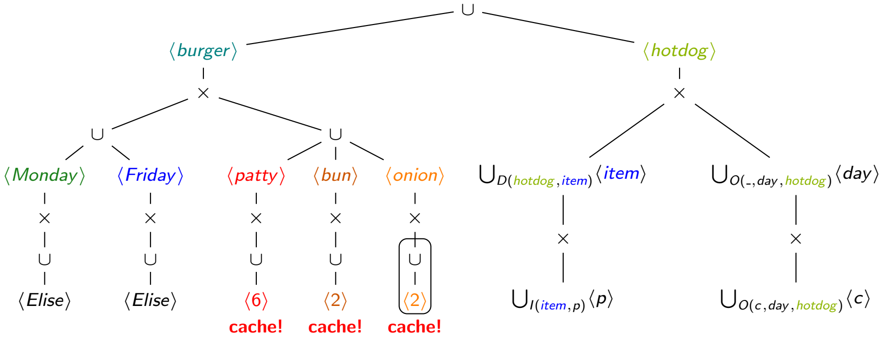
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



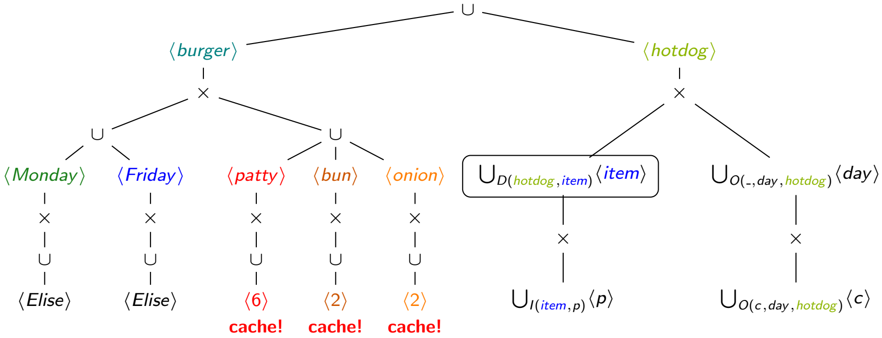
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



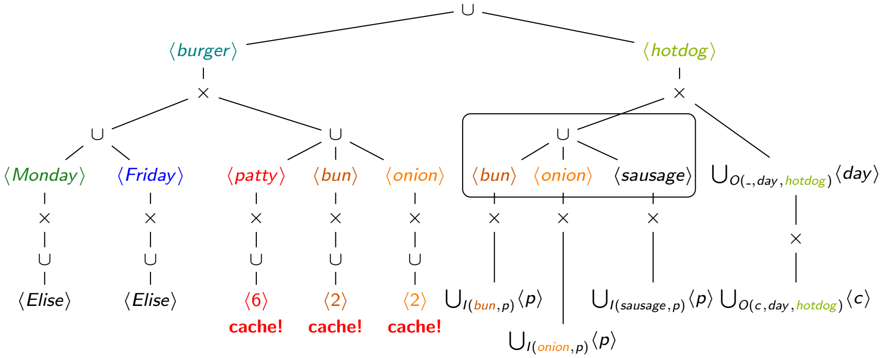
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



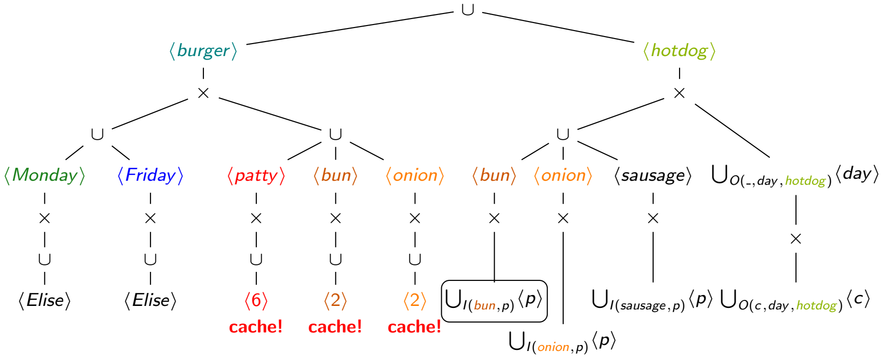
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



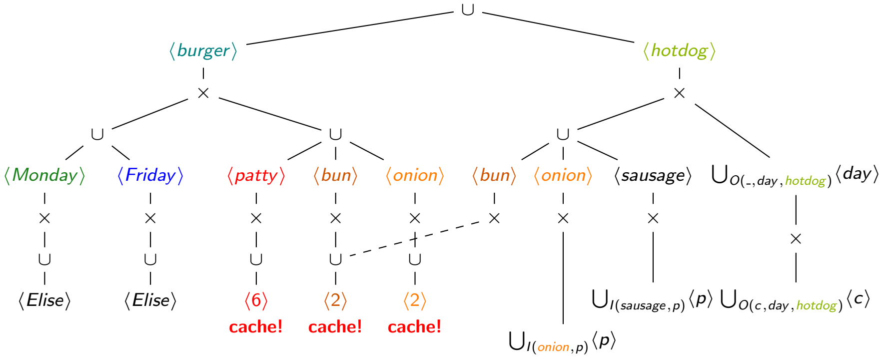
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



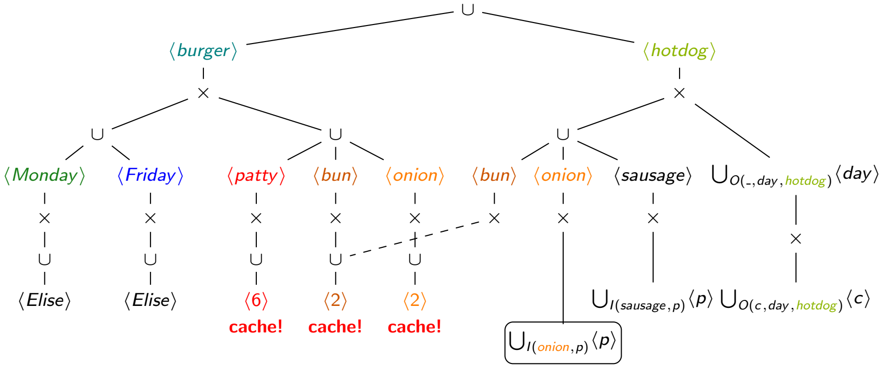
- price depends on item, but not on dish.
  Cache prices for specific items!

Example: Computing the Factorized Join Result with FDB



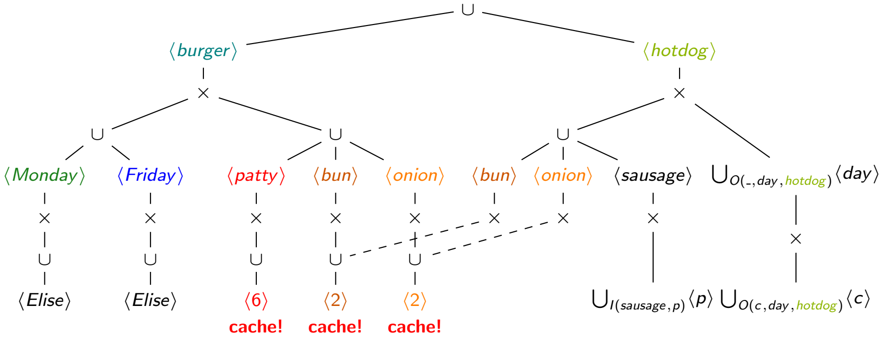
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



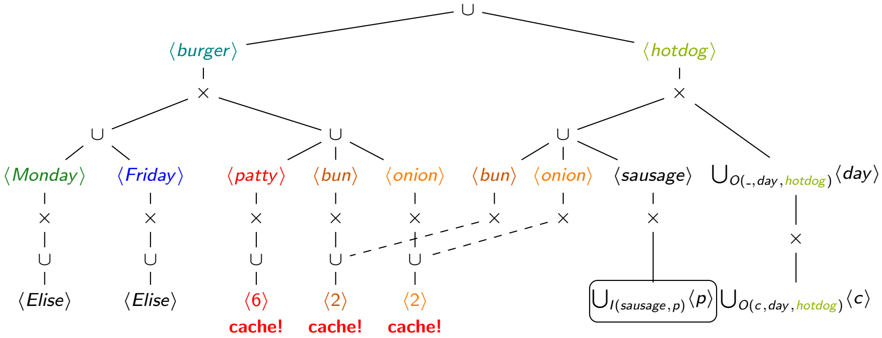
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



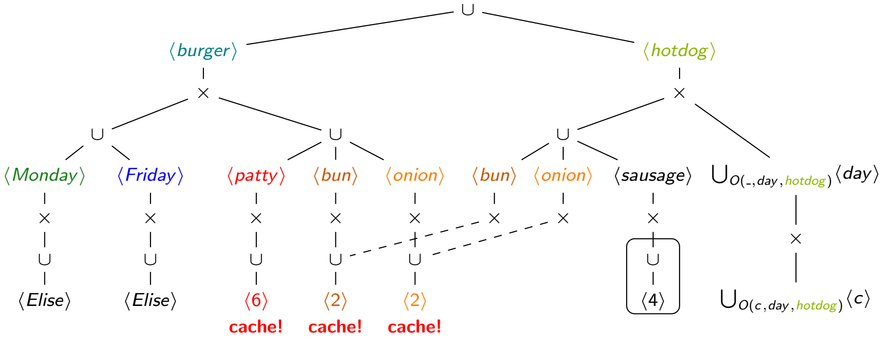
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



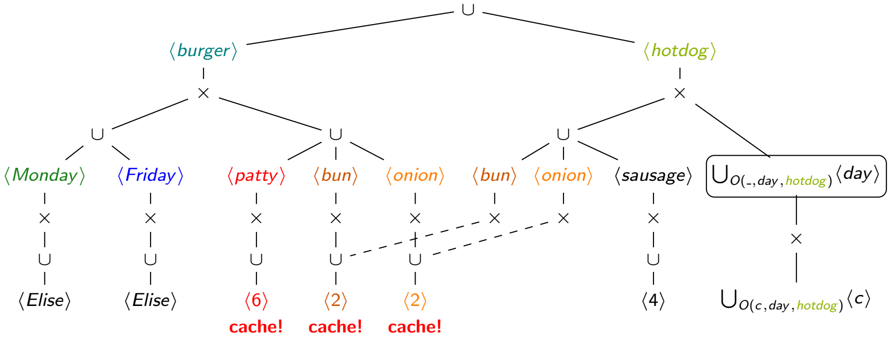
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



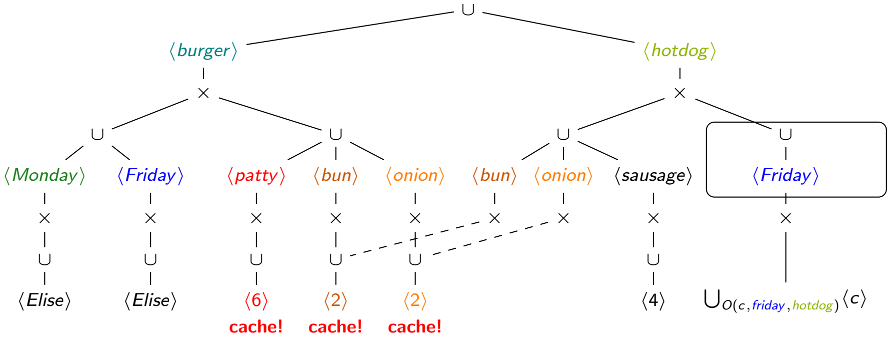
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



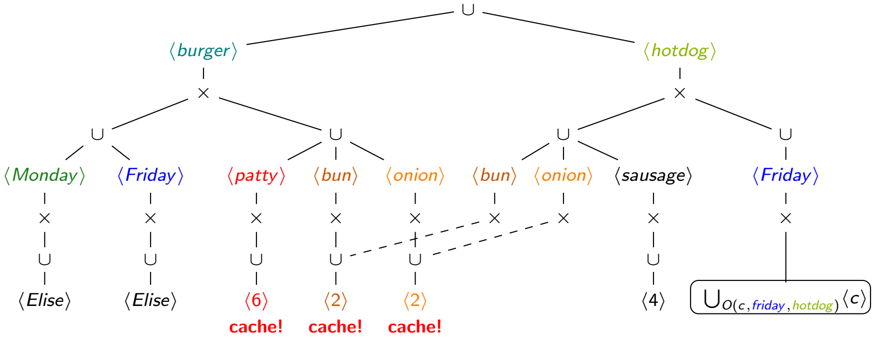
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



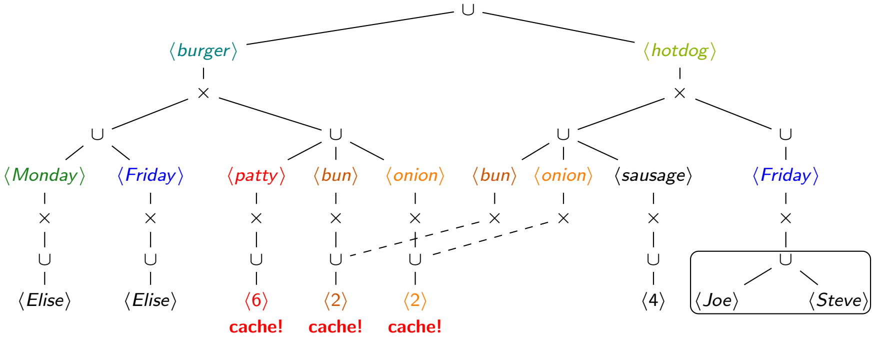
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



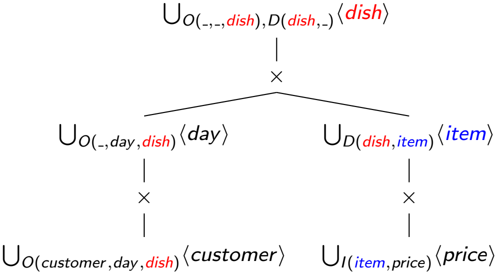
- `price` depends on `item`, but not on `dish`.
  Cache `prices` for specific `items`!
- Reuse cached `prices` for specific `items`!

Example: Computing the Factorized Join Result with FDB



- `price` depends on `item`, but not on `dish`.
  Cache `price`s for specific `item`s!
- Reuse cached `price`s for specific `item`s!

Example: Computing the Factorized Join Result with FDB



- Relations are sorted following any topological order of the variable order
- The intersection of relations O and D on *dish* takes time $O(N_{\min} \log(N_{\max}/N_{\min}))$, where $N_m = m(|\pi_{dish} O|, |\pi_{dish} D|)$.
- The remaining operations are lookups in the relations, where we first fix the *dish* value and then the *day* and *item* values.

LeapFrog TrieJoin Algorithm

- Much acclaimed worst-case optimal join algorithm used by LogicBlox [V14]
- Computes a listing representation of the join result

  $\Rightarrow$ It does not exploit factorization
- $\approx$ Glorified multi-way sort-merge join with an efficient list intersection
- Several generalizations, e.g., Tetris, Minesweeper, and PANDA [NRR13,ANS17]

LeapFrog TrieJoin is a special case of FDB, where

- the input variable order Δ is a path

  (i.e., **no branching**)
- for each variable A, $key(A)$ consists of all ancestors of A in Δ.
  (i.e., **no caching**)

Example: Computing the Full Join Result

The listing representation of the result of our join:

$$O(\text{customer}, \text{day}, \text{dish}), D(\text{dish}, \text{item}), I(\text{item}, \text{price})$$

can be computed by FDB using any total variable order.

Variable order

day

customer

dish

item

price

$\longrightarrow$

FDB execution plan

$$\bigcup_{O(\_,day,\_)} \langle day \rangle$$

$$\times$$

$$\bigcup_{O(customer,day,\_)} \langle customer \rangle$$

$$\times$$

$$\bigcup_{O(customer,day,dish),D(dish,\_)} \langle dish \rangle$$

$$\times$$

$$\bigcup_{D(dish,item),I(item,\_)} \langle item \rangle$$

$$\times$$

$$\bigcup_{I(item,price)} \langle price \rangle$$

Example: Computing the Full Join Result

$$\bigcup_{O(\_,day,\_)} \langle day \rangle$$

$$\times$$

$$\bigcup_{O(customer,day,\_)} \langle customer \rangle$$

$$\times$$

$$\bigcup_{O(customer,day,dish),D(dish,\_)} \langle dish \rangle$$

$$\times$$

$$\bigcup_{D(dish,i),I(i,\_)} \langle i \rangle$$

$$\times$$

$$\bigcup_{I(i,p)} \langle p \rangle$$

Example: Computing the Full Join Result

$$\boxed{\bigcup_{O(\_,\,day,\,\_)} \langle day \rangle}$$

$$\times$$

$$\bigcup_{O(customer,\,day,\,\_)} \langle customer \rangle$$

$$\times$$

$$\bigcup_{O(customer,\,day,\,dish),\,D(dish,\,\_)} \langle dish \rangle$$

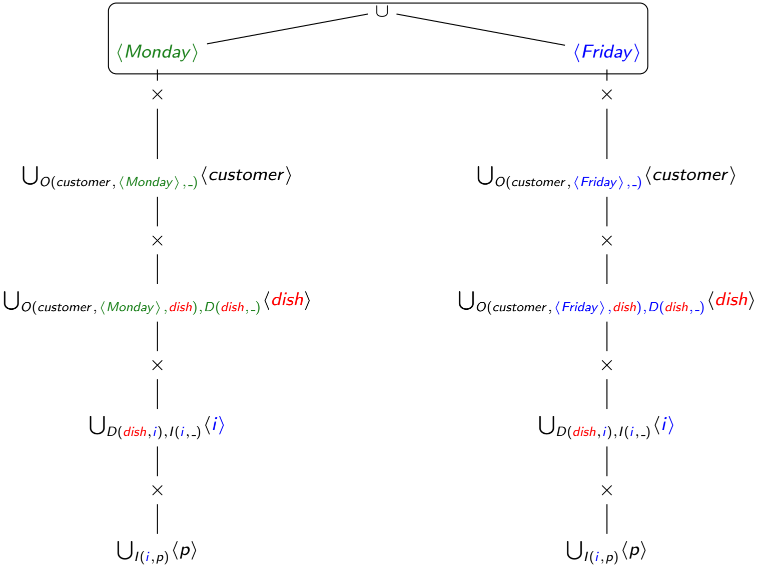
$$\times$$

$$\bigcup_{D(dish,\,i),\,I(i,\,\_)} \langle i \rangle$$

$$\times$$

$$\bigcup_{I(i,\,p)} \langle p \rangle$$

Example: Computing the Full Join Result



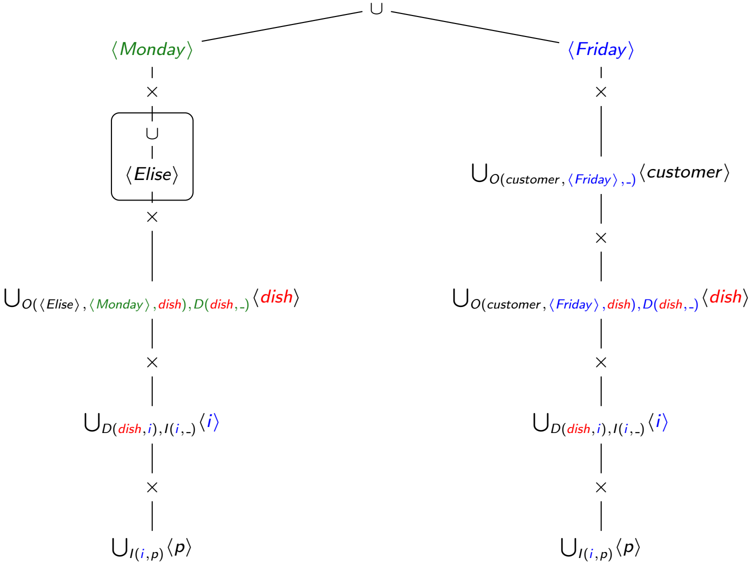
Example: Computing the Full Join Result

$$\cup$$

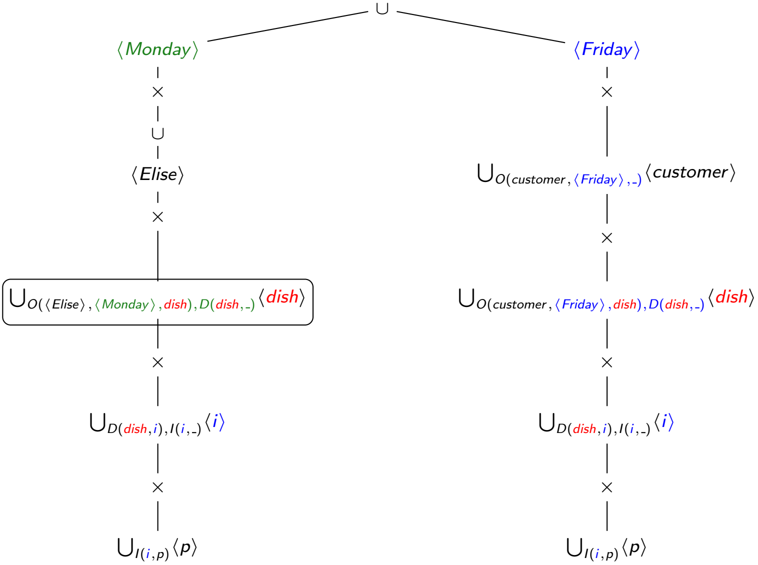
$$\langle Monday \rangle \times \boxed{\bigcup_{O(customer,\langle Monday\rangle,\_)} \langle customer \rangle} \times \bigcup_{O(customer,\langle Monday\rangle,dish),D(dish,\_)} \langle dish \rangle \times \bigcup_{D(dish,i),I(i,\_)} \langle i \rangle \times \bigcup_{I(i,p)} \langle p \rangle$$

$$\langle Friday \rangle \times \bigcup_{O(customer,\langle Friday\rangle,\_)} \langle customer \rangle \times \bigcup_{O(customer,\langle Friday\rangle,dish),D(dish,\_)} \langle dish \rangle \times \bigcup_{D(dish,i),I(i,\_)} \langle i \rangle \times \bigcup_{I(i,p)} \langle p \rangle$$

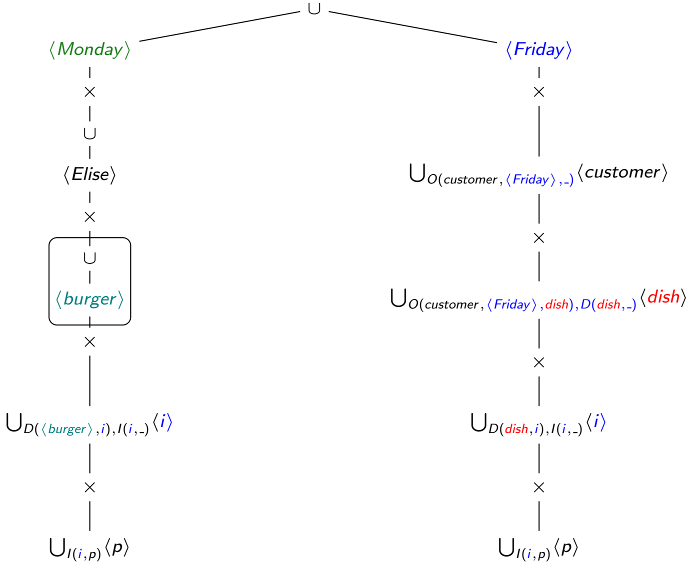
Example: Computing the Full Join Result



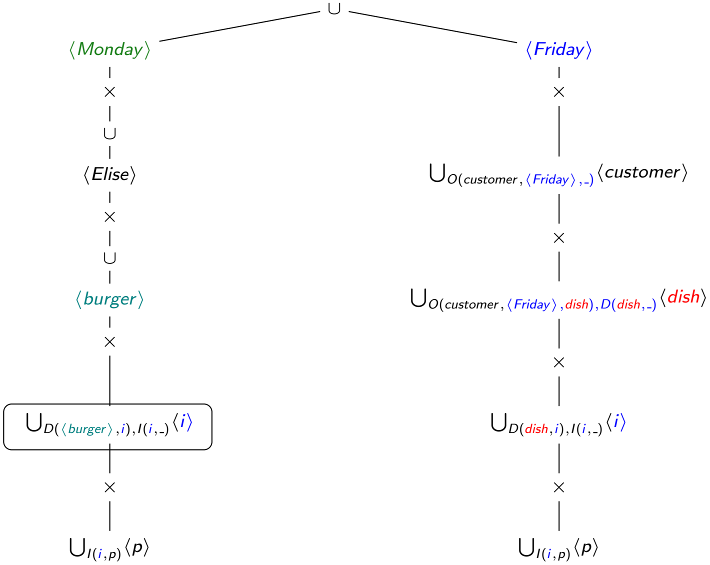
Example: Computing the Full Join Result



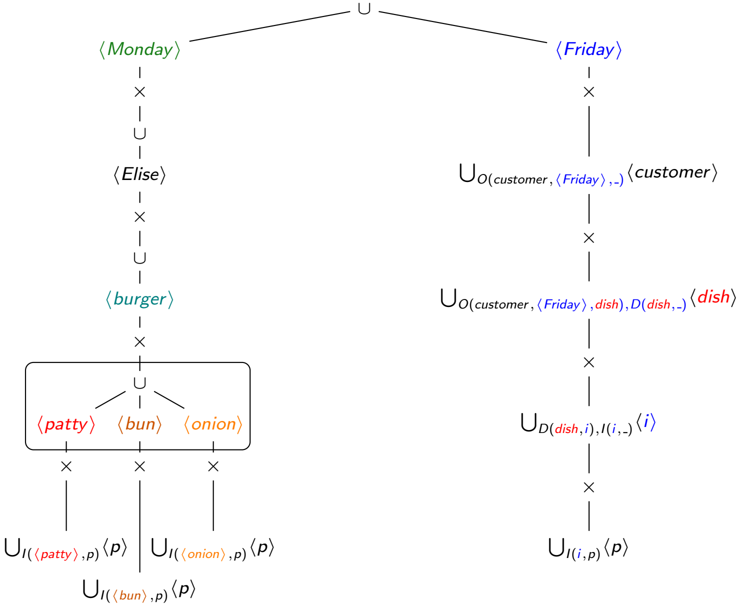
Example: Computing the Full Join Result



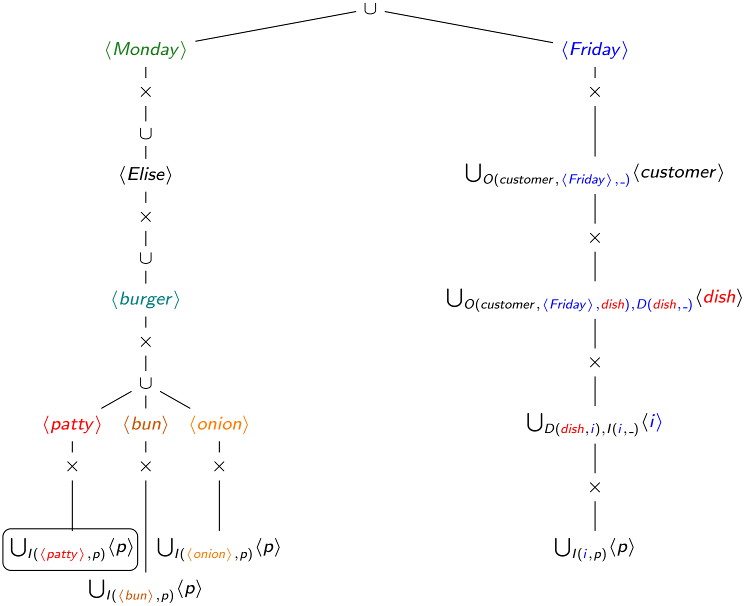
Example: Computing the Full Join Result



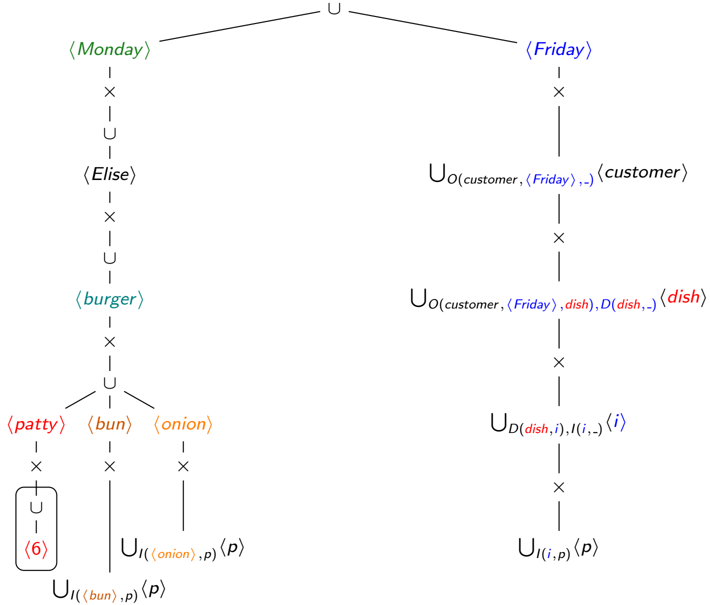
Example: Computing the Full Join Result



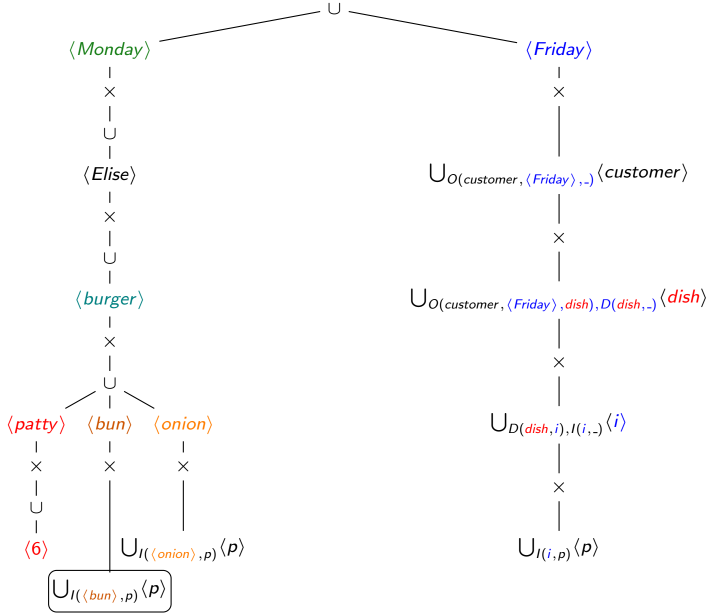
Example: Computing the Full Join Result



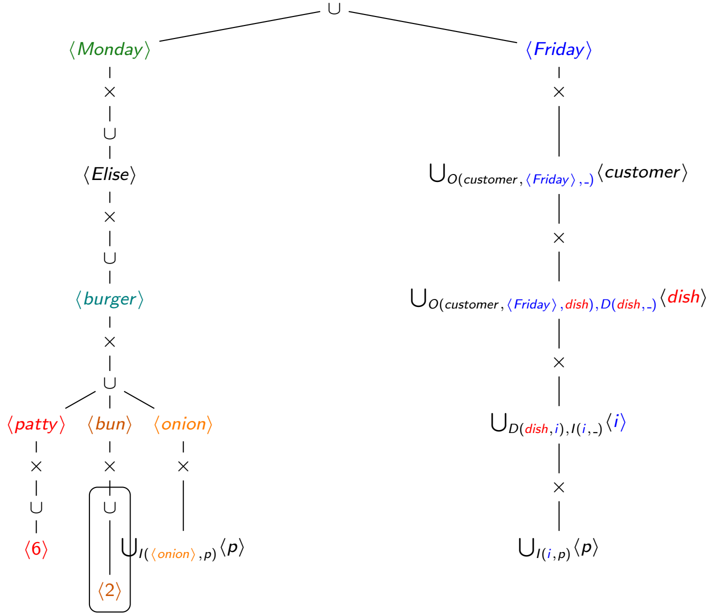
Example: Computing the Full Join Result



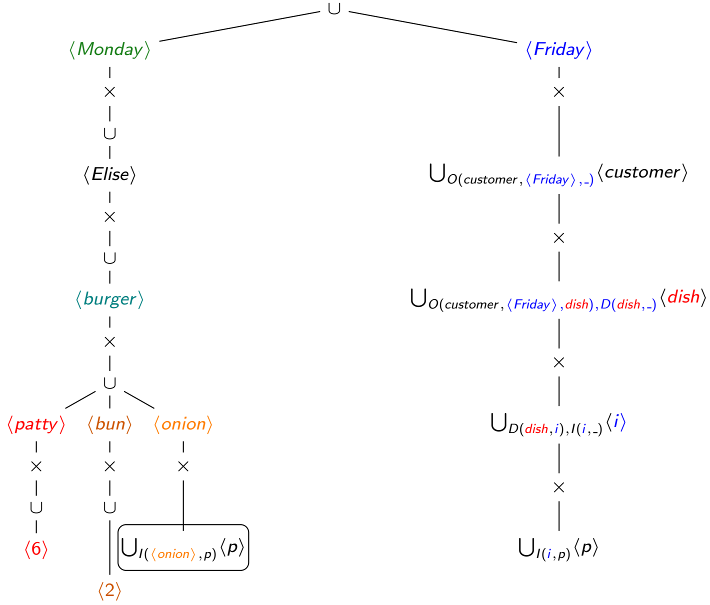
Example: Computing the Full Join Result



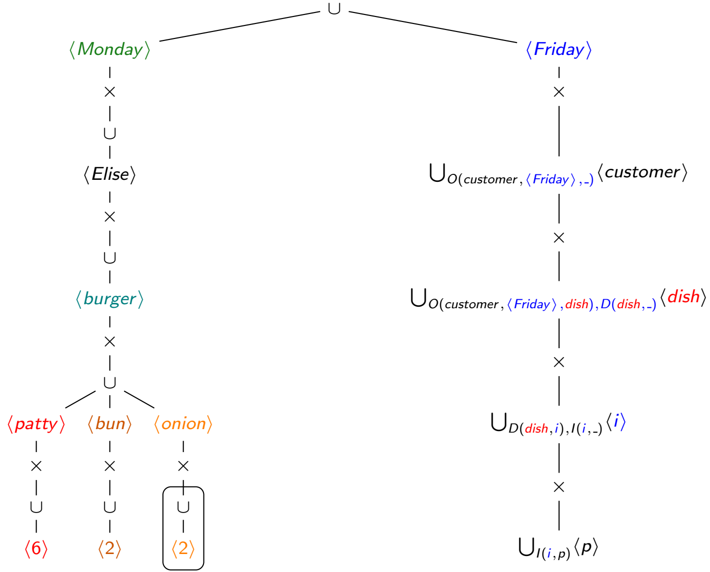
Example: Computing the Full Join Result



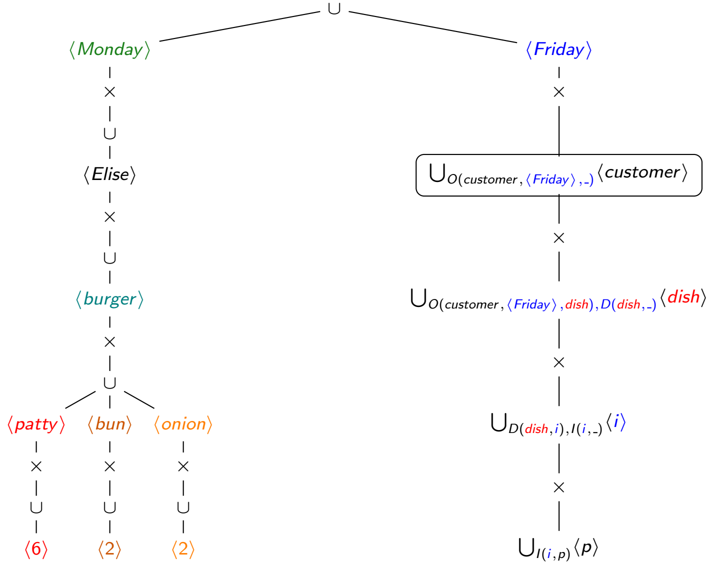
Example: Computing the Full Join Result



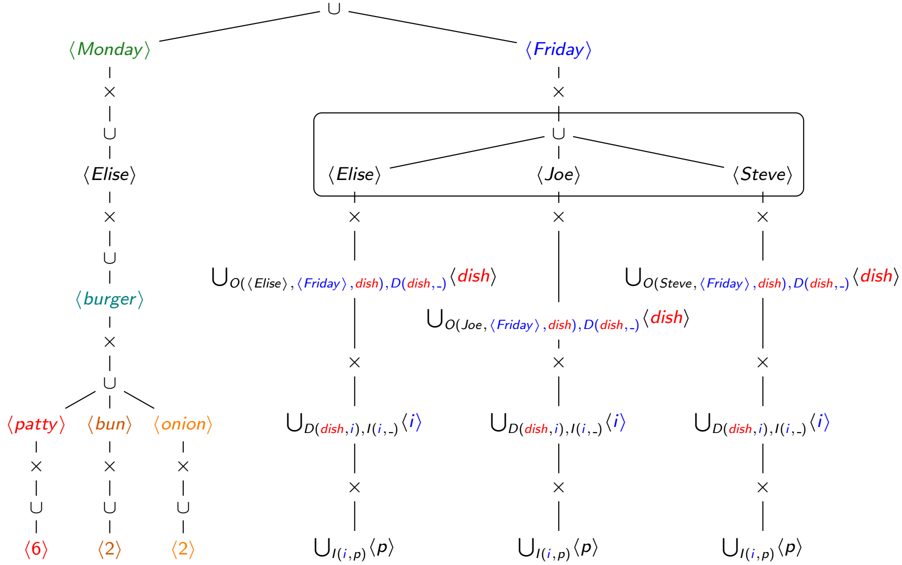
Example: Computing the Full Join Result



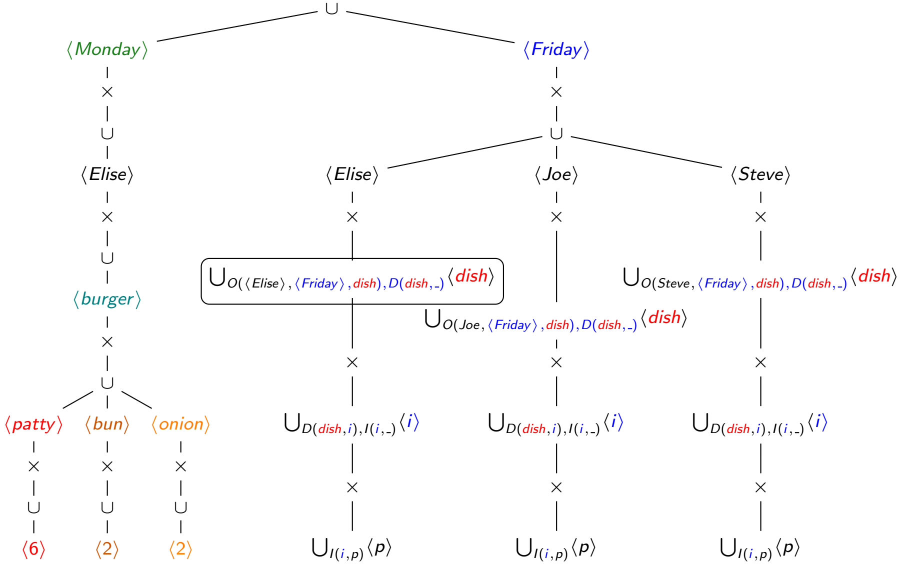
Example: Computing the Full Join Result



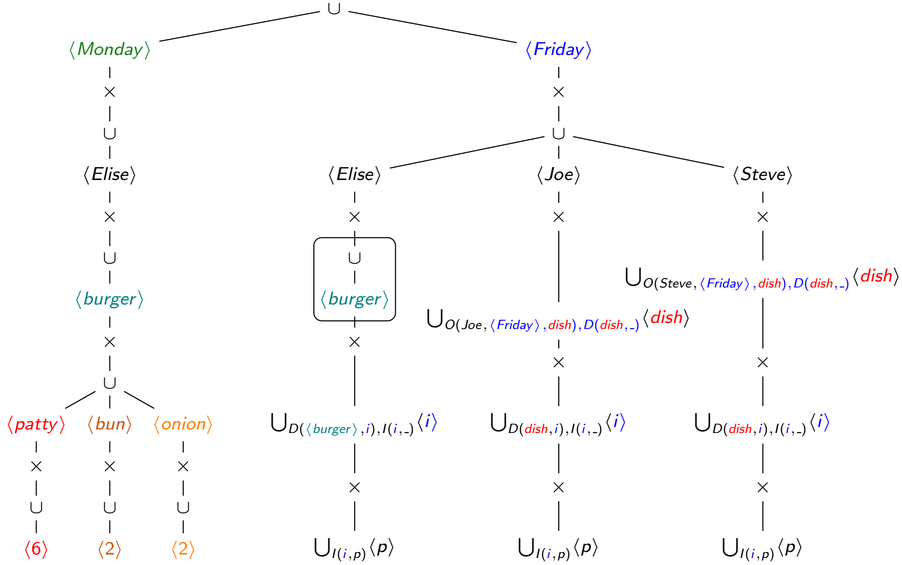
Example: Computing the Full Join Result



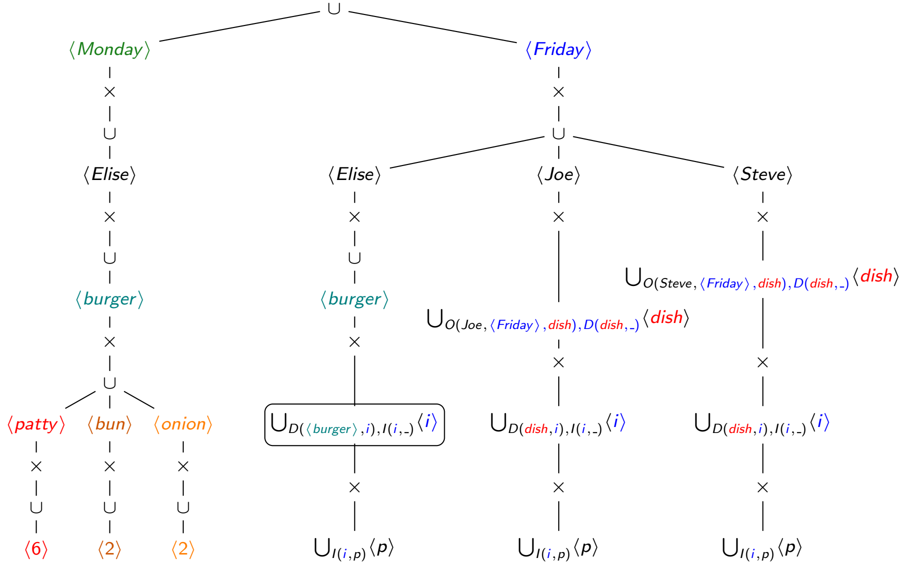
Example: Computing the Full Join Result



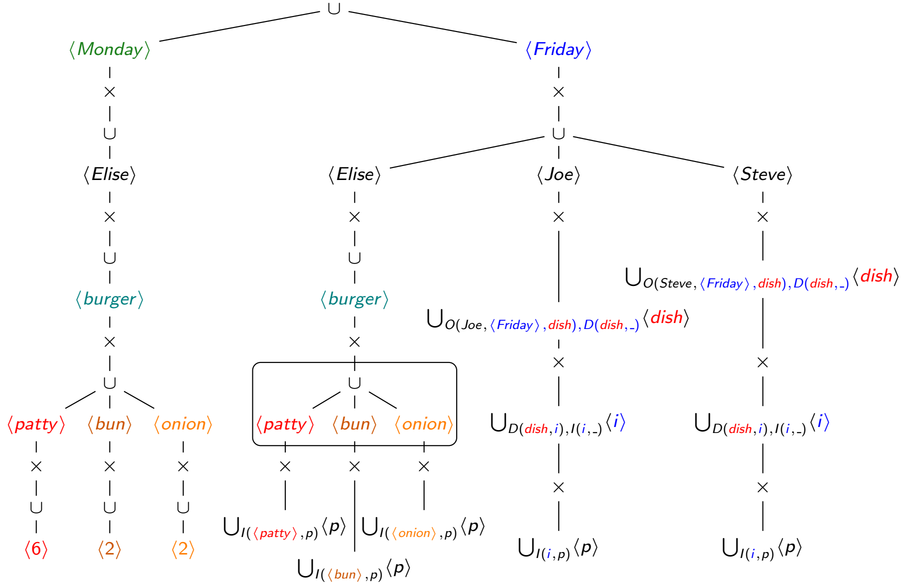
Example: Computing the Full Join Result



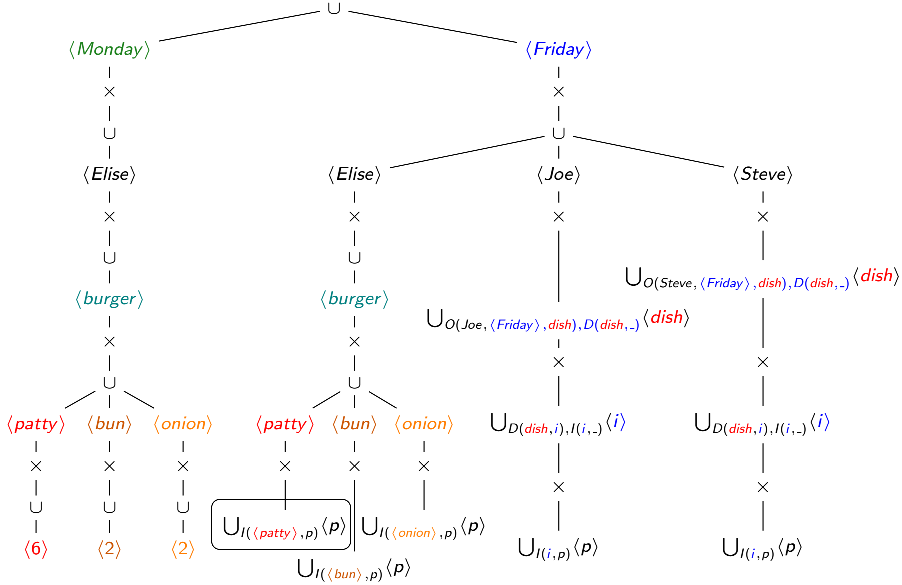
Example: Computing the Full Join Result



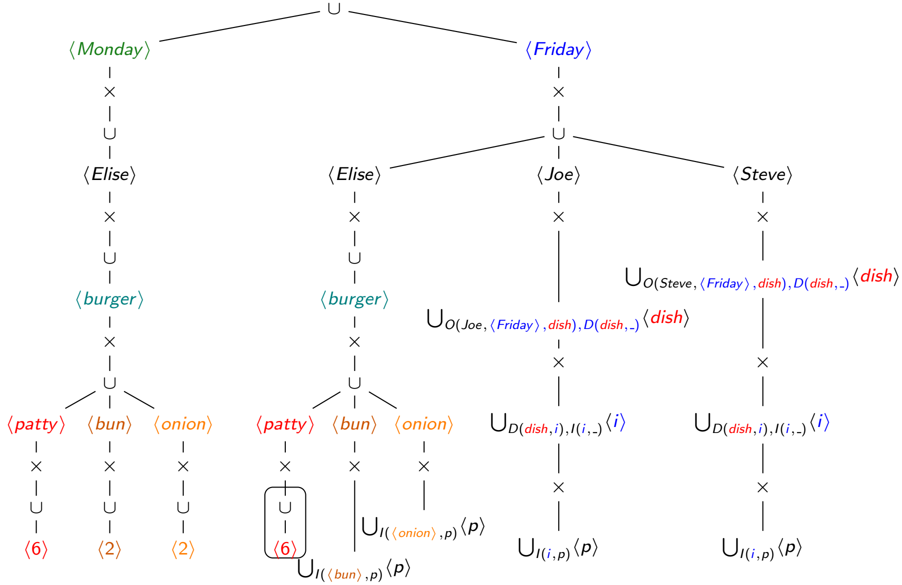
Example: Computing the Full Join Result



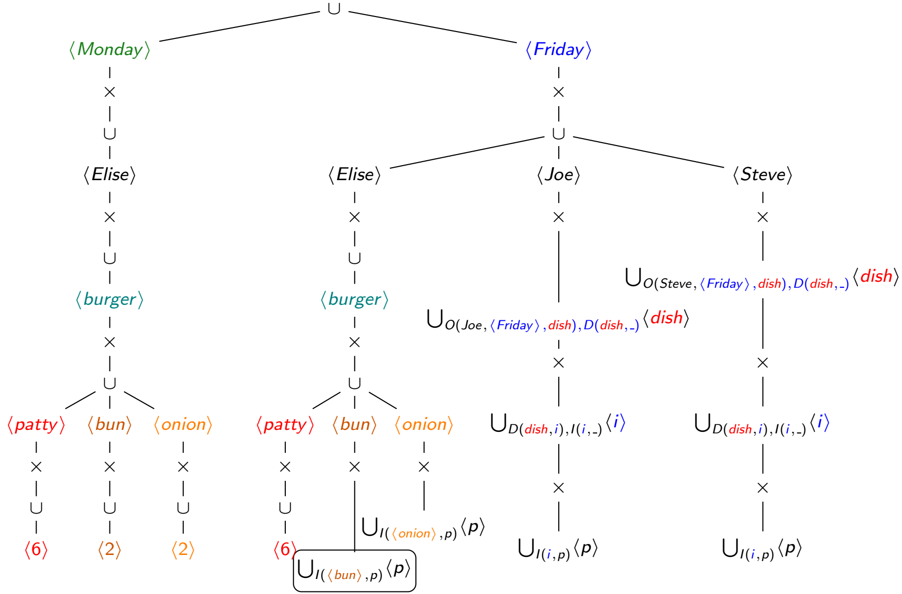
Example: Computing the Full Join Result



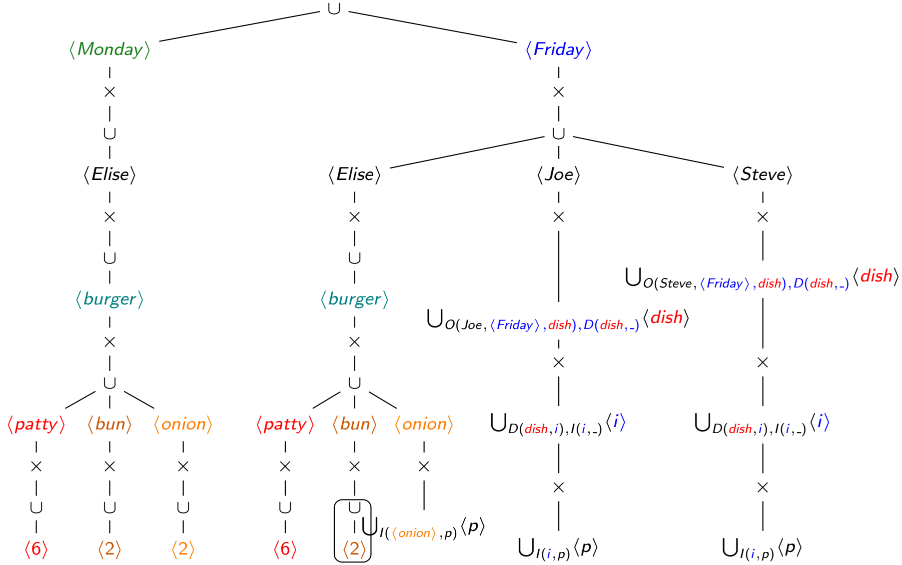
Example: Computing the Full Join Result



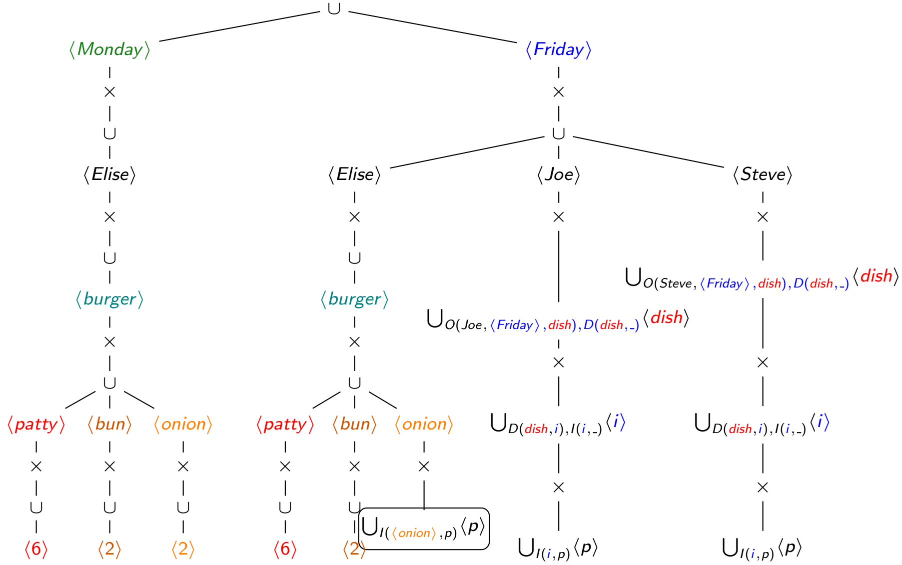
Example: Computing the Full Join Result



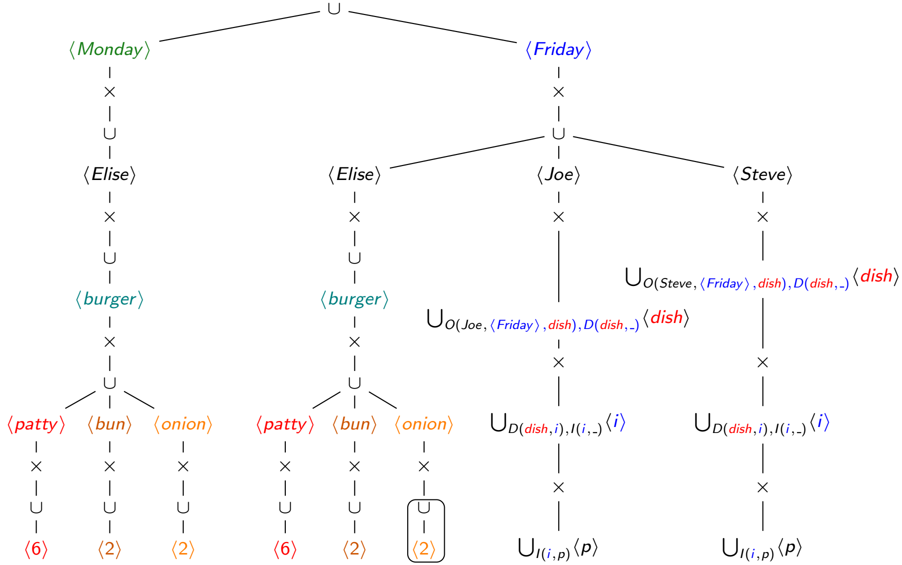
Example: Computing the Full Join Result



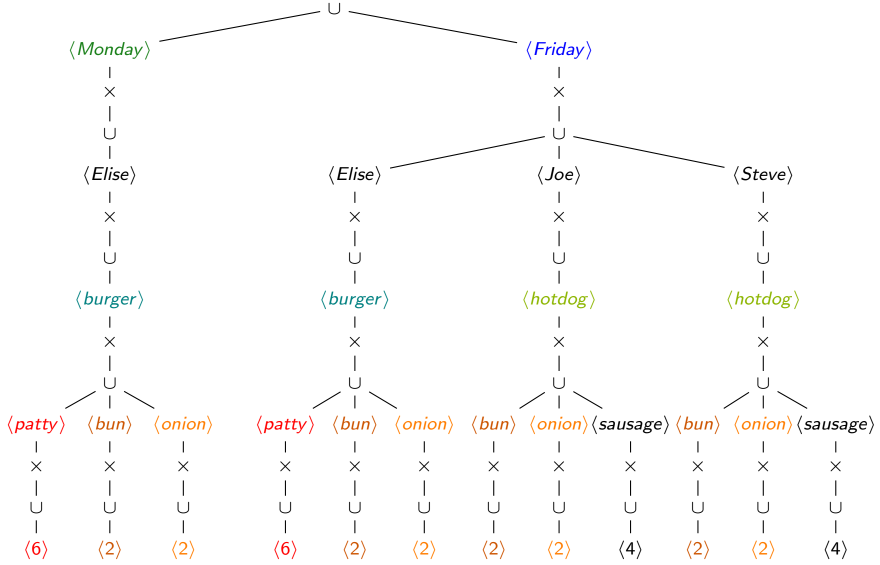
Example: Computing the Full Join Result



Example: Computing the Full Join Result

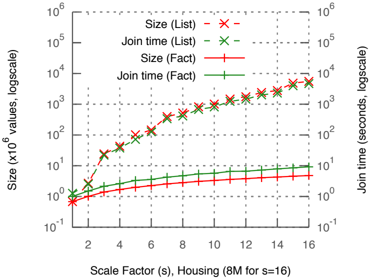


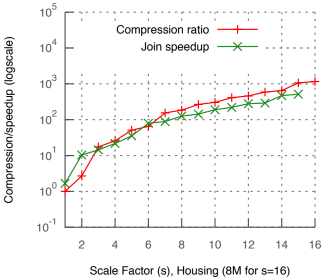
Example: Computing the Full Join Result



Experiment: Factorized vs. Listing Computation

		Retailer (3B)	LastFM (5.8M)
Join Size (values)	Factorization	169M	316K
	Listing	3.6B	591M
	Compression	21.4×	1870.7×
Join Time (sec)	FDB	30	10
	PostgreSQL	217	61
	Speedup	7×	6.1×





Both FDB and PostgreSQL list the records in the results of the join queries.

Outline of Part 1: Joins

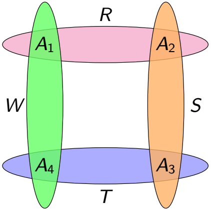
Relevant Work not Covered in the Course

- Widths, results sizes, and join computation *under functional dependencies* [GLVV12,ANS16,GT17,ANS17]
- *Adaptive* join processing with lower complexity [AYZ97,ANS17]
  - We exemplify this next with the 4-cycle join [AYZ97]
- Covers: Relational counterpart of factorized representation [KO18]

Recall the (4-cycle) Join

$$Q(A_1, A_2, A_3, A_4) = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1).$$

The linear program for its fractional edge cover number:



$$\begin{array}{llllllll}
\textit{minimize} & x_R + x_S + x_T + x_W \\
\textit{subject to} \\
A_1: & x_R & & & & + & x_W & \geq 1 \\
A_2: & x_R & + & x_S & & & & \geq 1 \\
A_3: & & & x_S & + & x_T & & \geq 1 \\
A_4: & & & & & x_T & + \; x_W & \geq 1 \\
& x_R \geq 0 & & x_S \geq 0 & & x_T \geq 0 & x_W \geq 0
\end{array}$$

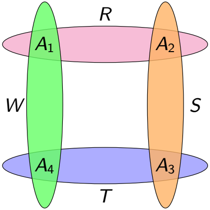
Solutions: $x_R = x_T = 1$ or $x_S = x_W = 1$. Then, $\rho^* = 2$. Also, $\text{fhtw} = 2$.

Lower bound $\Omega(N^2)$ obtained for
$R(A_1, A_2) = T(A_3, A_4) = [N] \times \{1\}$ and $S(A_2, A_3) = W(A_4, A_1) = \{1\} \times [N]$

- The variables A_1 and A_3 get values $[N]$
- The variable A_2 and A_4 get value $\{1\}$

Can We Do The *Boolean* 4-Cycle Join Faster?

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1).$$



We can use one of the two decompositions:

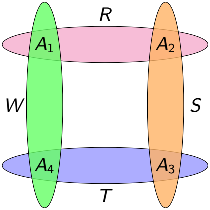
$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2}$$
$$T_2 : \underbrace{\{A_4, A_1, A_2\}}_{B_3} - \underbrace{\{A_2, A_3, A_4\}}_{B_4}$$

Lower-bound: A_1 and A_3 get values $[N]$ and A_2 and A_4 get value $\{1\}$.

- Use T_1: $\underbrace{R(A_1, A_2), S(A_2, A_3)}_{N \cdot N = N^2}$ cover B_1, $\underbrace{T(A_3, A_4), W(A_4, A_1)}_{N \cdot N = N^2}$ cover B_2

Can We Do The *Boolean* 4-Cycle Join Faster?

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1).$$



We can use one of the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2}$$

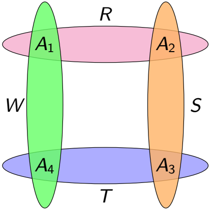
$$T_2 : \underbrace{\{A_4, A_1, A_2\}}_{B_3} - \underbrace{\{A_2, A_3, A_4\}}_{B_4}$$

Lower-bound: A_1 and A_3 get values $[N]$ and A_2 and A_4 get value $\{1\}$.

- Use T_1: $\underbrace{R(A_1, A_2), S(A_2, A_3)}_{N \cdot N = N^2}$ cover B_1, $\underbrace{T(A_3, A_4), W(A_4, A_1)}_{N \cdot N = N^2}$ cover B_2
- Use T_2: $\underbrace{R(A_1, A_2), W(A_4, A_1)}_{\mathbf{N}}$ cover B_3, $\underbrace{S(A_2, A_3), T(A_3, A_4)}_{\mathbf{N}}$ cover B_4

Can We Do The *Boolean* 4-Cycle Join Faster?

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1).$$



We can use one of the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2}$$
$$T_2 : \underbrace{\{A_4, A_1, A_2\}}_{B_3} - \underbrace{\{A_2, A_3, A_4\}}_{B_4}$$

Lower-bound: A_1 and A_3 get values $[N]$ and A_2 and A_4 get value $\{1\}$.

- Use T_1: $\underbrace{R(A_1, A_2), S(A_2, A_3)}_{N \cdot N = N^2}$ cover B_1, $\underbrace{T(A_3, A_4), W(A_4, A_1)}_{N \cdot N = N^2}$ cover B_2
- Use T_2: $\underbrace{R(A_1, A_2), W(A_4, A_1)}_{\mathbf{N}}$ cover B_3, $\underbrace{S(A_2, A_3), T(A_3, A_4)}_{\mathbf{N}}$ cover B_4

Idea: Why not use **different decompositions** for **different classes** of input databases or even for **different partitions** of a relation?

Light and Heavy Values

Fix $\epsilon \in [0, 1]$. A value a of variable A in relation R is:

HEAVY if $|\sigma_{A=a}(R)| \geq N^{\epsilon}$ **LIGHT** if $|\sigma_{A=a}(R)| < N^{\epsilon}$

Light and Heavy Values

Fix $\epsilon \in [0, 1]$. A value a of variable A in relation R is:

HEAVY if $|\sigma_{A=a}(R)| \geq N^\epsilon$ **LIGHT** if $|\sigma_{A=a}(R)| < N^\epsilon$

Partition $R(A_1, A_2)$ and $T(A_3, A_4)$ into heavy and light parts:

$$R = \underbrace{\{(a_1, a_2) \in R \mid a_1 \text{ is heavy}\}}_{R_h} \cup \underbrace{\{(a_1, a_2) \in R \mid a_1 \text{ is light}\}}_{R_l}$$

$$T = \underbrace{\{(a_3, a_4) \in T \mid a_3 \text{ is heavy}\}}_{T_h} \cup \underbrace{\{(a_3, a_4) \in T \mid a_3 \text{ is light}\}}_{T_l}$$

Evaluation of the 4-Cycle Boolean Query in $O(N^{3/2})$

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

Recall the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2} \quad T_2 : \overbrace{\{A_4, A_1, A_2\}}^{B_3} - \overbrace{\{A_2, A_3, A_4\}}^{B_4}$$

We rewrite Q as $Q() = Q_1() \cup Q_2() \cup Q_3()$, where

$$Q_1() = \mathbf{R_h}(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

$$Q_2() = \mathbf{R_l}(A_1, A_2), S(A_2, A_3), \mathbf{T_h}(A_3, A_4), W(A_4, A_1)$$

$$Q_3() = \mathbf{R_l}(A_1, A_2), S(A_2, A_3), \mathbf{T_l}(A_3, A_4), W(A_4, A_1)$$

Evaluation of the 4-Cycle Boolean Query in $O(N^{3/2})$

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

Recall the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2} \quad T_2 : \overbrace{\{A_4, A_1, A_2\}}^{B_3} - \overbrace{\{A_2, A_3, A_4\}}^{B_4}$$

We evaluate

$$Q_1() = \mathbf{R_h}(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

using T_1: $\underbrace{\pi_{A_1} R_h(A_1), S(A_2, A_3)}_{N^{1-\epsilon} \cdot N = N^{2-\epsilon}}$ covers B_1, $\underbrace{\pi_{A_1} R_h(A_1), T(A_3, A_4)}_{N^{1-\epsilon} \cdot N = N^{2-\epsilon}}$ covers B_2

For $\epsilon = 1/2$, the time to compute Q_1 is $N^{3/2}$.

Evaluation of the 4-Cycle Boolean Query in $O(N^{3/2})$

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

Recall the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2} \quad T_2 : \overbrace{\{A_4, A_1, A_2\}}^{B_3} - \overbrace{\{A_2, A_3, A_4\}}^{B_4}$$

We evaluate

$$Q_2() = \mathbf{R_l}(A_1, A_2), S(A_2, A_3), \mathbf{T_h}(A_3, A_4), W(A_4, A_1)$$

using T_1: $\underbrace{\pi_{A_3} T_h(A_3), R_l(A_1, A_2)}_{N^{1-\epsilon} \cdot N = N^{2-\epsilon}}$ covers B_1, $\underbrace{\pi_{A_3} T_h(A_3), W(A_1, A_4)}_{N^{1-\epsilon} \cdot N = N^{2-\epsilon}}$ covers B_2

For $\epsilon = 1/2$, the time to compute Q_2 is $N^{3/2}$.

Evaluation of the 4-Cycle Boolean Query in $O(N^{3/2})$

$$Q() = R(A_1, A_2), S(A_2, A_3), T(A_3, A_4), W(A_4, A_1)$$

Recall the two decompositions:

$$T_1 : \overbrace{\{A_1, A_2, A_3\}}^{B_1} - \overbrace{\{A_1, A_3, A_4\}}^{B_2} \quad T_2 : \overbrace{\{A_4, A_1, A_2\}}^{B_3} - \overbrace{\{A_2, A_3, A_4\}}^{B_4}$$

We evaluate

$$Q_3() = \mathbf{R_l}(A_1, A_2), S(A_2, A_3), \mathbf{T_l}(A_3, A_4), W(A_4, A_1)$$

using T_2: $\underbrace{W(A_4, A_1), R_l(A_1, A_2)}_{N \cdot N^{\epsilon} = N^{1+\epsilon}}$ covers B_1, $\underbrace{S(A_2, A_3), T_l(A_3, A_4)}_{N \cdot N^{\epsilon} = N^{1+\epsilon}}$ covers B_2

For $\epsilon = 1/2$, the time to compute Q_3 is $N^{3/2}$.

Covers: Relational Counterparts of Factorizations

- Factorized representations are not relational :(
  - This makes it difficult to integrate them into relational data systems
- Covers of Query Results [KO17]
  - Relations that are lossless representations of query results, yet are as succinct as factorized representations
  - For a join query Q and any database of size N, a cover has size $O(N^{fhtw(Q)})$ and can be computed in time $\widetilde{O}(N^{fhtw(Q)})$
- How to get a cover?
  - Construct a hypertree decomposition of the query
  - Project query result onto the bags of the hypertree decomposition
  - Construct on these projections the hypergraph of the query result
  - Take a minimal edge cover of this hypergraph

Recall the Itemized Customer Orders Example

Orders (O for short)

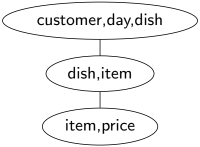
customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

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

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4



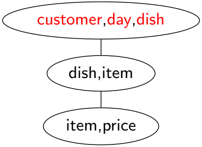
O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

Elise Monday burger

Elise Friday burger



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

The Hypergraph of the Query Result

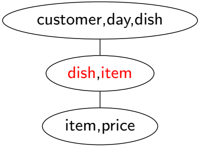
burger patty

Elise Monday burger

burger onion

Elise Friday burger

burger bun



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

The Hypergraph of the Query Result

burger patty

patty 6

Elise Monday burger

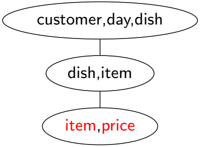
burger onion

onion 2

Elise Friday burger

burger bun

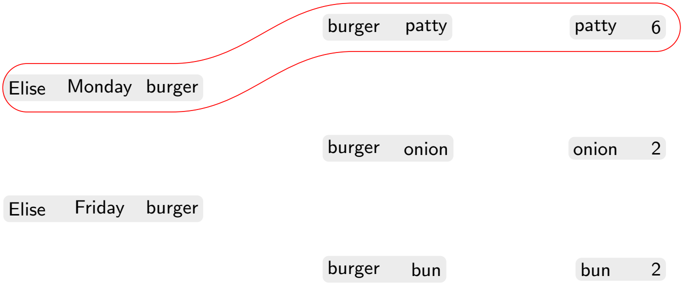
bun 2

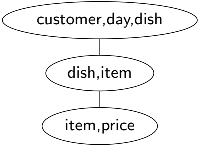


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

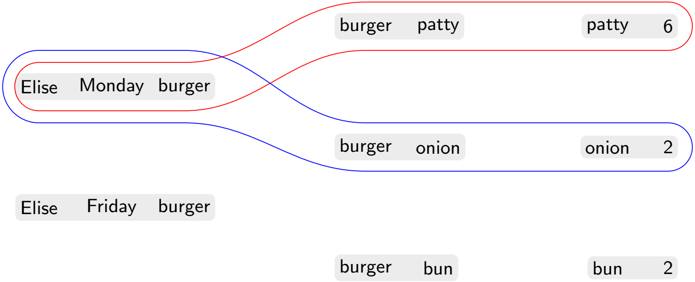


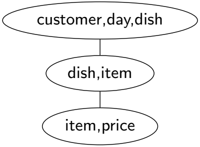


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

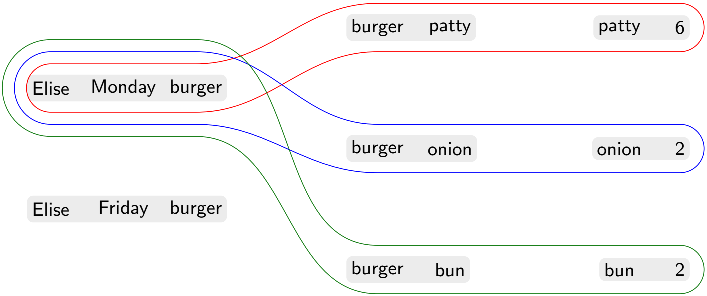


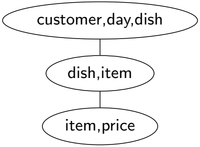


O(customer, day, dish), D(dish, item), I(item, price)

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

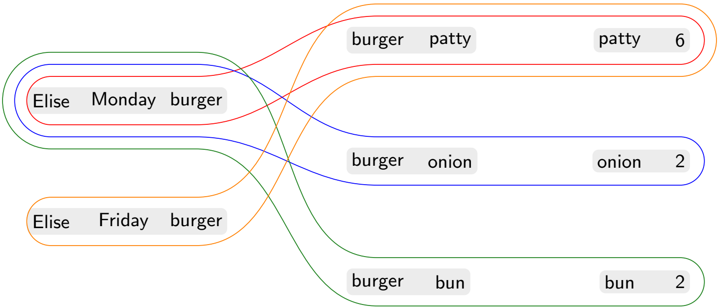
The Hypergraph of the Query Result

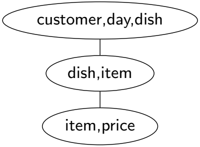




O(customer, day, dish), D(dish, item), I(item, price)				
customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Monday	burger	onion	2
Elise	Monday	burger	bun	2
Elise	Friday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2
...	...	...	...	...

The Hypergraph of the Query Result

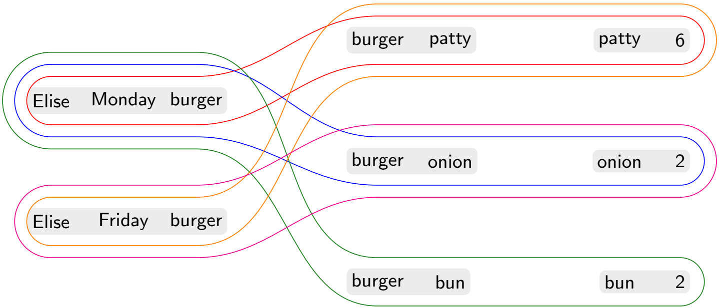


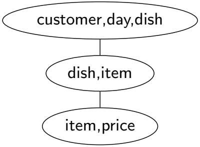


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

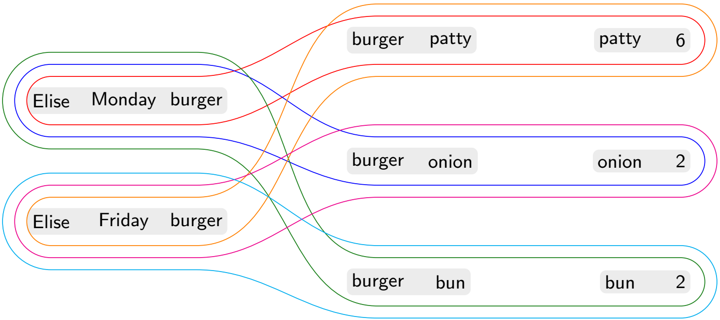


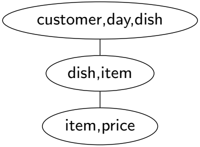


O(customer, day, dish), D(dish, item), I(item, price)

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

The Hypergraph of the Query Result

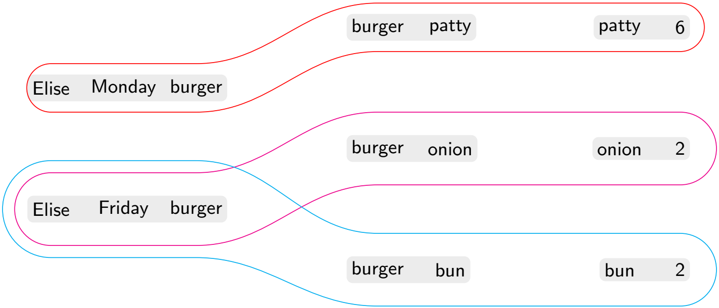


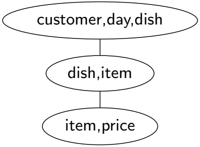


O(customer, day, dish), D(dish, item), I(item, price)

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 Minimal Edge Cover of the Hypergraph





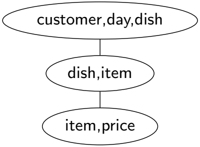
O(customer, day, dish), D(dish, item), I(item, price)

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 Cover of (a part of) the Query Result

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

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



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

References

LW49 **An inequality related to the isoperimetric inequality.**
Loomis, Whitney. In Bull. Amer. Math. Soc., 55 (1949).
https://www.ams.org/journals/bull/1949-55-10/

A81 **On the number of subgraphs of prescribed type of graphs with a given number of edges.**
Alon. In Israel J. Math., 38 (1981).
https://link.springer.com/content/pdf/10.1007/BF02761855.pdf

BT95 **Projections of bodies and hereditary properties of hypergraphs.**
Bollobaás, Thomason. In Bull. London Math. Soc., 27 (1995).
https://pdfs.semanticscholar.org/02c2/9f48e698ccbe7854be8012439c535453634f.pdf

AYZ97 **Finding and counting given length cycles.**
Alon, Yuster, Zwick. In Algorithmica 17, 3 (1997).
https://m.tau.ac.il/~nogaa/PDFS/ayz97.pdf

GLS99 **Hypertree decompositions and tractable queries.**
Gottlob, Leone, Scarcello. In PODS 1999.
https://arxiv.org/abs/cs/9812022

AGM08 **Size bounds and query plans for relational joins.**
Atserias, Grohe, Marx. In FOCS 2008 and SIAM J. Comput., 42(4) 2013.
http://epubs.siam.org/doi/10.1137/110859440

References

M10 **Approximating fractional hypertree width.**
Marx. In ACM TALG 2010.
http://dl.acm.org/citation.cfm?id=1721845

NPRR12 **Worst-case optimal join algorithms: [extended abstract]**
Ngo, Porat, Ré, Rudra. In PODS 2012.
http://dl.acm.org/citation.cfm?id=2213565

OZ12 **Factorised representations of query results: size bounds and readability.**
Olteanu, Zavodny. In ICDT 2012.
http://dl.acm.org/citation.cfm?doid=2274576.2274607
Also https://arxiv.org/abs/1104.0867, April 2011.

GLVV12 **Size and treewidth bounds for conjunctive queries.**
Gottlob, Lee, Valiant, Valiant. In J. ACM, 59 (2012).
https://www.cs.ox.ac.uk/files/5024/GLVV_7_11_conjqueries_jacm.pdf

NRR13 **Skew Strikes Back: New Developments in the Theory of Join Algorithms.**
Ngo, Ré, Rudra. In SIGMOD Rec. 2013.
https://arxiv.org/abs/1310.3314

V14 **Triejoin: A Simple, Worst-Case Optimal Join Algorithm.**
Veldhuizen. In ICDT 2014.
http://openproceedings.org/ICDT/2014/paper_13.pdf

References

OZ15 **Size Bounds for Factorised Representations of Query Results.**
Olteanu, Zavodny. In ACM TODS 2015 (submitted July 2013).
`http://dl.acm.org/citation.cfm?doid=2656335`

ANS16 **Computing join queries with functional dependencies.**
Abo Khamis, Ngo, Suciu. In PODS 2017.
`https://arxiv.org/abs/1604.00111`

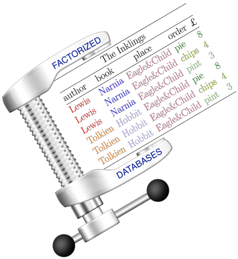
GT17 **Entropy Bounds for Conjunctive Queries with Functional Dependencies.**
Gogacz, Torunczyk. In ICDT 2017.
`http://drops.dagstuhl.de/opus/volltexte/2017/7047/`

ANS17 **What do Shannon-type inequalities, submodular width, and disjunctive Datalog have to do with one another?**
Abo Khamis, Ngo, Suciu. In PODS 2017.
`https://arxiv.org/abs/1612.02503`

KO18 **Covers of Query Results.**
Kara, Olteanu. In ICDT 2018.
`https://arxiv.org/abs/1709.01600`

N18 **Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems.**
Ngo. In PODS 2018.
`https://arxiv.org/abs/1803.09930`

Outline of Part 1: Joins



Introduction by Examples

Decompositions and Variable Orders

Size Bounds for Join Results

Worst-Case Optimal Join Algorithms

Further Work and References

Quiz

QUIZ on Joins (1/4)

For each of the following queries, please show the following:

1. A hypertree decomposition and an equivalent variable order
2. The fractional edge cover number and the fractional hypertree width

Path Query of length n:

$$P_n(X_1, \ldots, X_{n+1}) = R_1(X_1, X_2), R_2(X_2, X_3), R_3(X_3, X_4), \ldots, R_n(X_n, X_{n+1}).$$

QUIZ on Joins (2/4)

For each of the following queries, please show the following:

1. A hypertree decomposition and an equivalent variable order
2. The fractional edge cover number and the fractional hypertree width

Loop Query of length n:

$$L_n(X_1, \ldots, X_{n+1}) = R_1(X_1, X_2), R_2(X_2, X_3), R_3(X_3, X_4), \ldots, R_n(X_n, X_1).$$

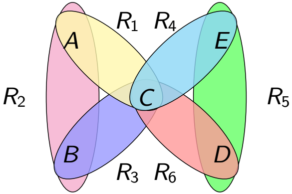
QUIZ on Joins (3/4)

For each of the following queries, please show the following:

1. A hypertree decomposition and an equivalent variable order
2. The fractional edge cover number and the fractional hypertree width

Bowtie Query:

$$Q_{\bowtie}(A, B, C, D, E) = R_1(A, C), R_2(A, B), R_3(B, C), R_4(C, E), R_5(E, D), R_6(C, D).$$



QUIZ on Joins (4/4)

For each of the following queries, please show the following:

1. A hypertree decomposition and an equivalent variable order
2. The fractional edge cover number and the fractional hypertree width

Loomis-Whitney Queries of length n: A LW_n query has n variables $X_1, \ldots, X_n$ and n relation symbols such that for every $i \in [n]$ the relation symbol R_i has variables $\{X_1, \ldots, X_n\} - \{X_i\}$:

$$\begin{aligned} LW_n(X_1, \ldots, X_n) = {} & R_1(X_2, \ldots, X_n), \ldots, R_i(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n), \ldots, \\ & R_n(X_1, \ldots, X_{n-1}) \end{aligned}$$

LW_n captures the Loomis–Whitney inequality: Estimate the "size" of a d-dimensional set by the sizes of its $(d-1)$-dimensional projections.

LW_3 is the triangle query.