From Joins to Aggregates
and Optimization Problems

Dan Olteanu (Oxford & Turing)

https://fdbresearch.github.io

Data Science Class
Alan Turing Institute
Jan 29, 2018
Acknowledgements

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

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

Some of the following slides are derived from presentations by

- Abo Khamis (in-db analytics diagrams)
- Kara (covers and various graphics)
- Ngo (FAQ)
- Schleich (performance and quizzes)

Lastly, Kara and Schleich proofread the slides.

I would like to thank them for their support and for sharing their work!
Goal of This Tutorial

Introduction to a principled, relatively new approach to in-database computation

It starts where mainstream introductory/advanced courses on databases finish.

- **Joins**
  - Worst-case optimal join algorithms
  - Listing vs. factorized representations of join results

- **Aggregates**
  - Generalization of join algorithms to aggregates over joins
  - Functional aggregate queries with applications in, e.g., DB, logic, probabilistic graphical models, matrix chain computation
  - New algorithms with low computational complexity

- **Optimizations**
  - In-database learning of regression and classification models

**Quizzes:** Test your understanding after class
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization
Join Queries

Basic building blocks in query languages. Studied extensively.

However, worst-case optimal join algorithms were only proposed recently. [NPRR12,NRR13,V14,OZ15,ANS17]

Likewise for systematic investigation of redundancy in the computation and representation of join results. [OZ12,OZ15,KO17]

This tutorial highlights recent work on worst-case optimal join algorithms under listing and factorized data representations.
Plan for Part 1 on Joins

- Introduction to join queries via examples
- Size bounds for results of join queries
  - Standard (exhaustive) listing representation
  - Factorized (succinct) representations
- Worst-case optimal join algorithms
  - LFTJ (LeapFrog TrieJoin) used by LogicBlox for listing representation
  - FDB (Factorized Databases) for factorized representations
Introduction to Join Queries
Join Example: Itemized Customer Orders

<table>
<thead>
<tr>
<th>Orders (O for short)</th>
<th>Dish (D for short)</th>
<th>Items (I for short)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Steve</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
</tbody>
</table>

Consider the natural join of the above relations:

<table>
<thead>
<tr>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>customer</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Join Example: Listing the Triangles in the Database

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_1(A, B), R_2(A, C), R_3(B, C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$C$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_0$</td>
<td>$a_0$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$...$</td>
<td>$a_0$</td>
<td>$...$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$b_m$</td>
<td>$a_0$</td>
<td>$c_m$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_0$</td>
<td>$a_1$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$...$</td>
<td>$b_0$</td>
<td>$...$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$a_m$</td>
<td>$b_0$</td>
<td>$a_m$</td>
<td>$c_0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_0$</td>
<td>$c_0$</td>
<td>$b_0$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$...$</td>
<td>$b_0$</td>
<td>$...$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$c_m$</td>
<td>$b_0$</td>
<td>$c_m$</td>
</tr>
<tr>
<td>$b_1$</td>
<td>$c_0$</td>
<td>$b_1$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$...$</td>
<td>$c_0$</td>
<td>$...$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>$b_m$</td>
<td>$c_0$</td>
<td>$b_m$</td>
<td>$c_0$</td>
</tr>
</tbody>
</table>

- $R_1$: A set of sides $(a, b)$ for triangle $R_1$.
- $R_2$: A set of sides $(a, c)$ for triangle $R_2$.
- $R_3$: A set of sides $(b, c)$ for triangle $R_3$.
- $R_1(A, B), R_2(A, C), R_3(B, C)$: The result of the join operation on the sets $R_1, R_2, R_3$. This table shows all possible combinations of sides that can form a triangle.
Join Hypergraphs

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

- Each variable in $Q$ corresponds to a node in $\mathcal{V}$
- Each relation symbol in $Q$ corresponds to a (hyper)edge in $\mathcal{E}$

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

- $\mathcal{V} = \{A, B, C\}$
- $\mathcal{E} = \{\{A, B\}, \{A, C\}, \{B, C\}\}$
We associate a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with every join query $Q$:

- Each variable in $Q$ corresponds to a node in $\mathcal{V}$.
- Each relation symbol in $Q$ corresponds to a (hyper)edge in $\mathcal{E}$.

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

$\mathcal{V} = \{ \text{cust}, \text{day}, \text{dish}, \text{item}, \text{price} \}$

$\mathcal{E} = \{ \{\text{cust, day, dish}\}, \{\text{dish, item}\}, \{\text{item, price}\} \}$
Hypertree Decompositions

Definition[GLS99]: A (hypertree) decomposition $T$ of the hypergraph $(\mathcal{V}, \mathcal{E})$ of a query $Q$ is a pair $(T, \chi)$, where
- $T$ is a tree
- $\chi$ is a function mapping each node in $T$ to a subset of $\mathcal{V}$ called bag.

Properties of a decomposition $T$:
- **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

\[ A, B \]
\[ B, C \]
\[ C, D \]
Hypertree Decompositions

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

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

Properties of the decomposition $T$:

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

![Diagram of the triangle query with hypertree decomposition](image)
Variable Orders

**Definition**[OZ15]: A *variable order* $\Delta$ for a query $Q$ is a pair $(F, \text{key})$, where
- $F$ is a rooted forest with one node per variable in $Q$
- $\text{key}$ is a function mapping each variable $A$ to a subset of its ancestor variables in $F$.

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

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

```
  A  key(A) = {}  B  key(B) = {}
   |      |      |      |
   B  key(B) = {A}  key(A) = {B}  A  key(A) = {B}
   |      |      |      |      |      |
   C  key(C) = {B}  C  key(C) = {B}  D  key(D) = {C}
   |      |      |      |      |      |
   D  key(D) = {C}  D  key(D) = {C}  D  key(D) = {C}
```
Variable Orders

**Definition** [OZ15]: A *variable order* $\Delta$ for a query $Q$ is a pair $(F, \text{key})$, where

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

Properties of a variable order $\Delta$ for $Q$:

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

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

<table>
<thead>
<tr>
<th>A</th>
<th>$\text{key}(A) = \emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>$\text{key}(B) = {A}$</td>
</tr>
<tr>
<td>C</td>
<td>$\text{key}(C) = {A, B}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>$\text{key}(B) = \emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$\text{key}(A) = {B}$</td>
</tr>
<tr>
<td>C</td>
<td>$\text{key}(C) = {A, B}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>$\text{key}(C) = \emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>$\text{key}(B) = {C}$</td>
</tr>
<tr>
<td>A</td>
<td>$\text{key}(A) = {B, C}$</td>
</tr>
</tbody>
</table>
Hypertree Decompositions ⇔ Variable Orders

From variable order $\Delta$ to hypertree decomposition $\mathcal{T}$: 

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

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

```
A  key(A) = \emptyset

B  key(B) = \{A\}  \Rightarrow  A, B

C  key(C) = \{A, B\}  \Rightarrow  A, B, C
```

$\Rightarrow  A, B, C$
Hypertree Decompositions ⇔ Variable Orders

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

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

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

```
A  key(A) = {}   A
  |                     \downarrow
B  key(B) = \{A\}    A, B  \Rightarrow  A, B
  |                     \downarrow
  C  key(C) = \{B\}   B, C
  |                     \downarrow
  D  key(D) = \{C\}   C, D
```

```
A, B
  \downarrow
B, C
  \downarrow
C, D
```
Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order $\Delta$:

- Create a node $A$ in $\Delta$ 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 $\Delta$
- 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 $\Delta$: \[\text{[OZ15]}\]

- Create a node $A$ in $\Delta$ 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 $\Delta$
- Update $key$ for each variable at each step.

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

Step 1:

$A$ is removed from $\mathcal{T}$ and inserted into $\Delta$ $\Rightarrow$

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

$A, B, C$
Hypertree Decompositions ⇔ Variable Orders

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

- Create a node $A$ in $\Delta$ 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 $\Delta$
- 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 $\Delta$

$A, B, C \Rightarrow B$

$\begin{align*}
A & \quad \text{key}(A) = \emptyset \\
B & \quad \text{key}(B) = \{A\}
\end{align*}$
Hypertree Decompositions $\Leftrightarrow$ Variable Orders

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

- Create a node $A$ in $\Delta$ 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 $\Delta$
- Update key for each variable at each step.

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

\[\begin{align*}
\text{Step 3:} \\
C \text{ is removed from } \mathcal{T} \\
\text{and inserted into } \Delta \\
A, B, C \quad \Rightarrow \\
\text{(key update)} \\
A \quad \text{key}(A) = \emptyset \\
B \quad \text{key}(B) = \{A\} \\
C \quad \text{key}(C) = \{A, B\}
\end{align*}\]
Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $T$ to variable order $\Delta$: \cite{OZ15}

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

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

```
\begin{tikzpicture}
\node[shape=circle,draw] (1) at (0,0) {$A, B$};
\node[shape=circle,draw] (2) at (0,-1) {$B, C$};
\node[shape=circle,draw] (3) at (0,-2) {$C, D$};
\draw (1) -- (2);
\end{tikzpicture}
```
Hypertree Decompositions $\iff$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order $\Delta$: \cite{OZ15}

- Create a node $A$ in $\Delta$ 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 $\Delta$
- 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 $\Delta$

$A \quad \text{key}(A) = \emptyset$

\[ A, B \quad B, C \quad C, D \]
Hypertree Decompositions $\Leftrightarrow$ Variable Orders

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

- Create a node $A$ in $\Delta$ 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 $\Delta$
- 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 $\Delta$
$A \quad \text{key}(A) = \emptyset$
$B \quad \text{key}(B) = \{A\}$
Hypertree Decompositions $\Leftrightarrow$ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order $\Delta$:  

- Create a node $A$ in $\Delta$ 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 $\Delta$
- 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 $\Delta$

- $key(A) = \emptyset$
- $key(B) = \{A\}$
- $key(C) = \{B\}$
Hypertree Decompositions ⇔ Variable Orders

From hypertree decomposition $\mathcal{T}$ to variable order $\Delta$:

- Create a node $A$ in $\Delta$ 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 $\Delta$
- Update $key$ for each variable at each step.

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

```
A, B

B, C

C, D

Step 4:
D is removed from $\mathcal{T}$
and inserted into $\Delta$

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

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

$C$ $key(C) = \{B\}$

$D$ $key(D) = \{C\}$
```
Size Bounds for Listing Representation of Join Results
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 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 inputs
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 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 inputs.
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 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 $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supset \{(1, 1)\}$
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 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 $R_1 = [N] \times \{1\}, R_2 = [N] \times \{1\}, R_3 \supset \{(1, 1)\}$

- 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?
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!

\[
\max_m = |\text{IndependentSet}(Q)| \leq |\text{EdgeCover}(Q)| = \min_k
\]

\[\text{max}_m \text{ and } \min_k \text{ do not necessarily meet!}\]

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 $m \Rightarrow \exists$ database with size $\geq N^m$.

By duality of linear programming:

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

- 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 $Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n)$, $\rho^*(Q)$ is the cost of an optimal solution to the linear program:

$$\text{minimize} \quad \sum_{i \in [n]} x_{R_i}$$

subject to

$$\sum_{i: \text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} A_j,$$

$$x_{R_i} \geq 0 \quad \forall i \in [n].$$

- $x_{R_i}$ 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 $\geq 1$.
- In the integer program variant for the edge cover, $x_{R_i} \in \{0, 1\}$.
Example of Fractional Edge Cover Computation (1)

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

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

\[ \Rightarrow \rho^*(Q) = 3 \]

\[ \Rightarrow \text{Size} \leq N^3 \text{ and for some inputs is } \Theta(N^3). \]
Example of Fractional Edge Cover Computation (2)

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

\[
\begin{align*}
\text{minimize } & x_{R_1} + x_{R_2} + x_{R_3} \\
\text{subject to } & x_{R_1} \geq 0 \\
& x_{R_2} \geq 0 \\
& x_{R_3} \geq 0
\end{align*}
\]

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?
Consider the triangle query \( Q: R_1(A, B), R_2(A, C), R_3(B, C) \).

\[
\begin{align*}
\text{minimize} & \quad x_{R_1} + x_{R_2} + x_{R_3} \\
\text{subject to} & \\
A & : \quad x_{R_1} + x_{R_2} \geq 1 \\
B & : \quad x_{R_1} + x_{R_3} \geq 1 \\
C & : \quad x_{R_2} + x_{R_3} \geq 1 \\
x_{R_1} & \geq 0 \quad x_{R_2} \geq 0 \quad x_{R_3} \geq 0
\end{align*}
\]

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 \( R_1 = R_2 = R_3 = [\sqrt{N}] \times [\sqrt{N}] \).
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:

\[
\begin{align*}
\text{minimize } & \quad x_R + x_S + x_T + x_W \\
\text{subject to } & \quad A_1 : \quad x_R + x_W \geq 1 \\
& \quad A_2 : \quad x_R + x_S \geq 1 \\
& \quad A_3 : \quad x_S + x_T \geq 1 \\
& \quad A_4 : \quad x_T + x_W \geq 1 \\
& \quad x_R \geq 0 \quad x_S \geq 0 \quad x_T \geq 0 \quad x_W \geq 0
\end{align*}
\]

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 \( S = W = \{1\} \times [N] \).
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 $Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n)$:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in [n]} x_{R_i} \\
\text{subject to} & \quad \sum_{i : \text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} A_j, \\
& \quad x_{R_i} \geq 0 \quad \forall i \in [n].
\end{align*}
\]
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 \( Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n) \):

\[
\begin{align*}
\text{minimize} \quad & N \sum_{i \in [n]} x_{R_i} \\
\text{subject to} \quad & \sum_{i:\text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} A_j, \\
& x_{R_i} \geq 0 \quad \forall i \in [n].
\end{align*}
\]

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 $Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n)$:

$$\text{minimize} \quad \prod_{i \in [n]} N^{x_{R_i}}$$

subject to

$$\sum_{i : \text{edge } R_i \text{ covers node } A} x_{R_i} \geq 1 \quad \forall A \in \bigcup_{j \in [n]} A_j,$$

$$x_{R_i} \geq 0 \quad \forall i \in [n].$$

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 $Q(A_1 \cup \cdots \cup A_n) = R_1(A_1), \ldots, R_n(A_n)$:

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

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

<table>
<thead>
<tr>
<th>Orders (O for short)</th>
<th>Dish (D for short)</th>
<th>Items (I for short)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Steve</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
</tbody>
</table>

Consider the natural join of the above relations:

<table>
<thead>
<tr>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Factor Out Common Data Blocks

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The listing representation of the above query result is:

\[
\langle \text{Elise} \rangle \times \langle \text{Monday} \rangle \times \langle \text{burger} \rangle \times \langle \text{patty} \rangle \times \langle 6 \rangle \cup \\
\langle \text{Elise} \rangle \times \langle \text{Monday} \rangle \times \langle \text{burger} \rangle \times \langle \text{onion} \rangle \times \langle 2 \rangle \cup \\
\langle \text{Elise} \rangle \times \langle \text{Monday} \rangle \times \langle \text{burger} \rangle \times \langle \text{bun} \rangle \times \langle 2 \rangle \cup \\
\langle \text{Elise} \rangle \times \langle \text{Friday} \rangle \times \langle \text{burger} \rangle \times \langle \text{patty} \rangle \times \langle 6 \rangle \cup \\
\langle \text{Elise} \rangle \times \langle \text{Friday} \rangle \times \langle \text{burger} \rangle \times \langle \text{onion} \rangle \times \langle 2 \rangle \cup \\
\langle \text{Elise} \rangle \times \langle \text{Friday} \rangle \times \langle \text{burger} \rangle \times \langle \text{bun} \rangle \times \langle 2 \rangle \cup \ldots
\]

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!

There are several \textit{algebraically equivalent} factorized representations defined:

\begin{itemize}
  \item by distributivity of product over union and their commutativity;
  \item as groundings of variable orders.
\end{itemize}
.. 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:

\[ F_1 = (\langle 1 \rangle \cup \cdots \cup \langle n \rangle) \times (\langle 1 \rangle \cup \cdots \cup \langle m \rangle) \]

\[ F_2 = \langle 1 \rangle \times \langle 1 \rangle \cup \cdots \cup \langle 1 \rangle \times \langle m \rangle \]

\[ \quad \cup \cdots \cup \]

\[ \langle n \rangle \times \langle 1 \rangle \cup \cdots \cup \langle n \rangle \times \langle m \rangle. \]

- \( F_1 \) is factorized, \( F_2 \) is a listing representation
- \( F_1 \equiv F_2 \)
- \textbf{BUT} \( |F_1| = m + n \ll |F_2| = m \times n. \)

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 $O(N^\rho^*(Q))$.  

[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 $O(N^{\rho^*(Q)})$. [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 $O(N^{\rho^*(Q)})$. \[ \text{[AGM08]} \]

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

- a factorization \textit{with caching} of size $O(N^{fhtw(Q)})$. \[ \text{[OZ15]} \]
Given a join query $Q$, for any database of size $N$, the join result admits

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

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

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

\[
1 \leq fhtw(Q) \leq s(Q) \leq \rho^*(Q) \leq |Q| \\
\text{up to } \log |Q| \quad \text{up to } |Q|
\]

- $|Q|$ 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 $Q$ \[\text{M10}\]
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 $O(N^{\rho^*(Q)})$. [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.
The structure of the factorization over the above variable order $\Delta$:

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

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

- 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 $\Delta$ is the largest $\rho^*$ over subqueries defined by root-to-leaf paths in $\Delta$.
- $s(Q)$ is the minimum factorization width over all variable orders of $Q$.

In our example:

- Path $A\rightarrow E\rightarrow F$ 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.

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

Observation:

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

\[
\Rightarrow \text{Define } U_e = \bigcup_{(e,f) \in U} \langle f \rangle \text{ for each value } \langle e \rangle \text{ and use } U_e \text{ instead of the union } \bigcup_{(e,f) \in U} \langle f \rangle.
\]
Example: The Fractional Hypertree Width \textit{fhtw}

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

\begin{itemize}
  \item \textit{fhtw} for $\Delta$ is the largest $\rho^*(Q_{\text{key}(X) \cup \{X\}})$ over subqueries $Q_{\text{key}(X) \cup \{X\}}$ defined by the variables $\text{key}(X) \cup \{X\}$ for each variable $X$ in $\Delta$
  \item \textit{fhtw}(Q) is the minimum \textit{fhtw} over all variable orders of $Q$
\end{itemize}

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

A factorization with caching would be:

$$
\bigcup_{a \in A} \langle a \rangle \times \cdots \times \bigcup_{e \in E} (\langle e \rangle \times U_e) ;
\quad \begin{cases}
  U_e = \bigcup_{(e,f) \in U} \langle f \rangle
\end{cases}
$$
Alternative Characterizations of $fhtw$

The fractional hypertree width $fhtw$ has been originally defined for hypertree decompositions. \([M10]\)

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

\[
fhtw(Q) = \min_{(T, \chi) \in T} \max_{n \in T} \rho^*(Q_{\chi(n)})
\]
Alternative Characterizations of \( fhtw \)

The fractional hypertree width \( fhtw \) has been originally defined for hypertree decompositions.  

- Given a join query \( Q \).
- Let \( T \) be the set of hypertree decompositions of the hypergraph of \( Q \).

\[
fhtw(Q) = \min_{(T,\chi) \in T} \max_{n \in T} \rho^*(Q_{\chi(n)})
\]

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

- Given a join query \( Q \).
- Let \( VO \) be the set of variable orders of \( Q \).

\[
fhtw(Q) = \min_{(F,\text{key}) \in VO} \max_{v \in F} \rho^*(Q_{\text{key}(v) \cup \{v\}})
\]
Relational Counterpart of Factorized Representation
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 $\tilde{O}(N^{fhtw(Q)})$

- 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)

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Steve</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
<tr>
<td>Joe</td>
<td>Friday</td>
<td>hotdog</td>
</tr>
</tbody>
</table>

### Dish (D for short)

<table>
<thead>
<tr>
<th>dish</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>burger</td>
<td>patty</td>
</tr>
<tr>
<td>burger</td>
<td>onion</td>
</tr>
<tr>
<td>burger</td>
<td>bun</td>
</tr>
<tr>
<td>hotdog</td>
<td>bun</td>
</tr>
<tr>
<td>hotdog</td>
<td>onion</td>
</tr>
<tr>
<td>hotdog</td>
<td>sausage</td>
</tr>
</tbody>
</table>

### Items (I for short)

<table>
<thead>
<tr>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>sausage</td>
<td>4</td>
</tr>
</tbody>
</table>

---

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

```
<table>
<thead>
<tr>
<th>customer, day, dish</th>
<th>dish, item</th>
<th>item, price</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(customer, day, dish), D(dish, item), I(item, price)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
The Hypergraph of the Query Result

<table>
<thead>
<tr>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>Elise</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

<table>
<thead>
<tr>
<th>customer, day, dish</th>
<th>O(customer, day, dish), D(dish, item), I(item, price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise Monday burger</td>
<td>burger patty 6</td>
</tr>
<tr>
<td>Elise Friday burger</td>
<td>burger onion 2</td>
</tr>
<tr>
<td></td>
<td>burger bun 2</td>
</tr>
<tr>
<td></td>
<td>customer day dish item price</td>
</tr>
<tr>
<td>Elise Monday burger</td>
<td>burger patty 6</td>
</tr>
<tr>
<td>Elise Monday burger</td>
<td>burger onion 2</td>
</tr>
<tr>
<td>Elise Monday burger</td>
<td>burger bun 2</td>
</tr>
<tr>
<td>Elise Friday burger</td>
<td>burger patty 6</td>
</tr>
<tr>
<td>Elise Friday burger</td>
<td>burger onion 2</td>
</tr>
<tr>
<td>Elise Friday burger</td>
<td>burger bun 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

Elise Monday burger

Elise Friday burger

<table>
<thead>
<tr>
<th>customer, day, dish</th>
<th>dish, item</th>
<th>item, price</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(customer, day, dish), D(dish, item), I(item, price)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customer</td>
<td>day</td>
<td>dish</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Hypergraph of the Query Result

Elise  Monday  burger

Elise  Friday  burger

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
</tbody>
</table>

...
The Hypergraph of the Query Result

```
<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
A Minimal Edge Cover of the Hypergraph

- Elise Monday burger
  - burger patty
  - patty 6
- Elise Friday burger
  - burger onion
  - onion 2
- burger bun
  - bun 2

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

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

...
A Cover of (a part of) the Query Result

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>dish</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elise</td>
<td>Monday</td>
<td>burger</td>
<td>patty</td>
<td>6</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>onion</td>
<td>2</td>
</tr>
<tr>
<td>Elise</td>
<td>Friday</td>
<td>burger</td>
<td>bun</td>
<td>2</td>
</tr>
</tbody>
</table>

...
Compression by Factorization in Practice
Compression Contest: Factorized vs. Zipped Relations

![Graph showing compression ratios for different database scales.](image)

**Result of query** Orders \(\bowtie\) Dish \(\bowtie\) Items

- 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.

[2013]
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 (2/3)

LastFM public dataset


- 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/ ≥86Mx for versions of triangle/4-clique/bowtie queries with copies for UserArtists for each UserFriend copy
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
Worst-Case Optimal Join Algorithms
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

- $\tilde{O}(N^{\rho^*(Q)})$ as listing representation [NPRR12,V14]
- $\tilde{O}(N^{s(Q)})$ as factorization without caching [OZ15]
- $\tilde{O}(N^{fhtw(Q)})$ 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: $O(\text{customer, day, dish})$, $D(\text{dish, item})$, $I(\text{item, price})$

can be grounded to a factorized representation as follows:

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

| $\times$

| $\bigcup_{O(\_, \text{day, dish})} \langle \text{day} \rangle$
| $\bigcup_{D(\text{dish, item})} \langle \text{item} \rangle$

| $\times$

| $\bigcup_{O(\text{customer, day, dish})} \langle \text{customer} \rangle$
| $\bigcup_{I(\text{item, price})} \langle \text{price} \rangle$

This computation follows the variable order given below:

dish

| day
| item

| customer
| price
Example: Computing the Factorized Join Result with FDB

\[
\bigcup_{O(-,\text{-},\text{dish}),\, D(\text{dish},\text{-})} \langle \text{dish} \rangle \\
\times \\
\bigcup_{O(-,\text{day},\text{dish})} \langle \text{day} \rangle \quad \bigcup_{D(\text{dish},\text{item})} \langle \text{item} \rangle \\
\times \\
\bigcup_{O(\text{customer},\text{day},\text{dish})} \langle \text{customer} \rangle \quad \bigcup_{I(\text{item},\text{price})} \langle \text{price} \rangle
\]

- Relations are sorted following any topological order of the variable order.

- The intersection of relations \( O \) and \( D \) on \( \text{dish} \) takes time \( O(N_{\text{min}} \log(N_{\text{max}}/N_{\text{min}})) = \tilde{O}(N_{\text{min}}) \), where \( N_m = m(|\pi_{\text{dish}} O|, |\pi_{\text{dish}} D|) \).

- The remaining operations are lookups in the relations, where we first fix the \( \text{dish} \) value and then the \( \text{day} \) and \( \text{item} \) values.
LeapFrog TrieJoin Algorithm

- Much acclaimed worst-case optimal join algorithm used by LogicBlox [V14]
- Computes a listing representation of the join result
  \[ \Rightarrow \text{It does not exploit factorization} \]
- Glorified multi-way sort-merge join with an efficient list intersection
- Several generalizations, e.g., PANDA [NRR13,ANS17]

LeapFrog TrieJoin is a special case of FDB, where

- the input variable order \( \Delta \) is a path, and
- for each variable \( A \), \( \text{key}(A) \) consists of all ancestors of \( A \) in \( \Delta \).
Experiment: Factorized vs. Listing Computation

<table>
<thead>
<tr>
<th></th>
<th>Retailer (3B)</th>
<th>LastFM (5.8M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Factorization</td>
<td>169M</td>
<td>316K</td>
</tr>
<tr>
<td>Size Listing</td>
<td>3.6B</td>
<td>591M</td>
</tr>
<tr>
<td>(values) Compression</td>
<td>21.4×</td>
<td>1870.7×</td>
</tr>
<tr>
<td>Join FDB</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Time PostgreSQL</td>
<td>217</td>
<td>61</td>
</tr>
<tr>
<td>(sec) Speedup</td>
<td>7×</td>
<td>6.1×</td>
</tr>
</tbody>
</table>

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

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization
Aggregates

Important operators in database query languages and essential for applications.

Natural generalization of aggregates over joins can express a host of problems across Computer Science. [ANR16]

We highlight recent work on aggregate computation with lowest known computational complexity. This extends the work from Part 1. [BKOZ13,ANR16]

Part 3 later discusses an extension of this work to state-of-the-art machine learning inside the database. [SOC16,ANNOS17]
Plan for Part 2 on Aggregates

- Computation of aggregates over factorized joins using the FDB algorithm
  \[\text{[BKOZ13]}\]

- Factorized computation of aggregates using optimized relational queries.
  \[\text{[SOC16,OS16]}\]

- Functional Aggregate Queries (FAQs)
  \[\text{[ANR16]}\]
  - Generalize aggregate-join queries to many semirings, e.g., sum-product, max-product, Boolean
  - FAQ computation is factorized and has the computational complexity of aggregates over factorized joins

- FAQ computation using the InsideOut algorithm
  \[\text{[ANR16]}\]
Examples: Aggregates over Factorized Joins
Example 1: COUNT Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **COUNT(*):**
  - values $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto \ast$. 
Example 1: COUNT Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

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

*Example Diagram*
Example 2: SumProd Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

- **SUM(dish * price):**
  - Assume there is a function $f$ that turns dish into reals or indicator vectors.
  - All values except for dish & price $\mapsto 1$,
  - $\cup \mapsto +$,
  - $\times \mapsto \ast$. 
Example 2: SumProd Aggregate over Factorized Join

SQL aggregates can be computed in one pass over the factorization:

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

```plaintext
20 \ast f(\langle \text{burger} \rangle) + 16 \ast f(\langle \text{hotdog} \rangle)
```
Given an aggregate-join query $Q$ \cite{BKOZ13}

- Construct a variable order $\Delta$ where the group-by (free) variables are above the other (bound) variables of $Q$
  - A new width $w$ measure that is at least $\text{fhtw}$

- Compute the factorized join over $\Delta$
  - The complexity now depends on the width $w$

- Finally compute the aggregates in one pass over the factorized join.

Is it necessary to first compute the factorized join? Aggregates can be computed without materializing the factorized join. The factorized join becomes the trace of the aggregate computation. This is the factorized computation of the query $Q$. 

---

88 / 171
Given an aggregate-join query $Q$

- Construct a variable order $\Delta$ where the **group-by** (free) variables are above the other (bound) variables of $Q$
  - A new width $w$ measure that is at least $fhtw$

- Compute the factorized join over $\Delta$
  - The complexity now depends on the width $w$

- Finally compute the aggregates in one pass over the factorized join.

**Is it necessary to first compute the factorized join?**
Given an aggregate-join query $Q$ [BKOZ13]

- Construct a variable order $\Delta$ where the group-by (free) variables are above the other (bound) variables of $Q$
  
  ▶ A new width $w$ measure that is at least $fhtw$

- Compute the factorized join over $\Delta$
  
  ▶ The complexity now depends on the width $w$

- Finally compute the aggregates in one pass over the factorized join.

Is it necessary to first compute the factorized join?

Aggregates can be computed without materializing the factorized join

- The factorized join becomes the \textit{trace} of the aggregate computation

- This is the \textit{factorized computation} of the query $Q$. 

Example: Factorized Aggregate Computation

The 4-path query $Q_4$ on a graph with the edge relation $E$ ($E_i$'s are copies of $E$):

$$V_1(A), E_1(A, B), E_2(B, C), E_3(C, D), E_4(D, E), V_2(E)$$
Example: Factorized Aggregate Computation

The 4-path query $Q_4$ on a graph with the edge relation $E$ ($E_i$'s are copies of $E$):

$$V_1(A), E_1(A, B), E_2(B, C), E_3(C, D), E_4(D, E), V_2(E)$$

Recall sizes for factorized results of path queries

- $\rho^*(Q_4) = 3 \Rightarrow$ listing representation has size $O(|E|^3)$.

- $fhtw(Q_4) = 1 \Rightarrow$ factorization with caching has size $O(|E|)$. 
Example: Factorized Aggregate Computation

We would like to compute $\text{COUNT}(Q_4)$:

- in $O(|E|)$ time (no free variables, so use best variable order for $Q_4$)
- using optimized queries that are derived from the variable order of $Q_4$
- without materializing the factorized result of the path query

Convention:

- View the relations as functions mapping tuples to numbers.
- The functions for input relations map their tuples to 1.
Example: Factorized Computation of COUNT\( (Q_4) \)
Example: Factorized Computation of \( \text{COUNT}(Q_4) \)

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a)
\]
Example: Factorized Computation of $\text{COUNT}(Q_4)$

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \quad U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)
\]
Example: Factorized Computation of $\text{COUNT}(Q_4)$

$$U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \quad U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)$$

$$U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e)$$
Example: Factorized Computation of $\text{COUNT}(Q_4)$

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \quad U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)
\]

\[
U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e) \quad U_4(c) = \sum_{d \in \text{Dom}(D)} E_3(c, d) \cdot U_3(d)
\]
Example: Factorized Computation of \( \text{COUNT}(Q_4) \)

\[
U_1(b) = \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a)
\]
\[
U_2(c) = \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot U_1(b)
\]
\[
U_3(d) = \sum_{e \in \text{Dom}(E)} V_2(e) \cdot E_4(d, e)
\]
\[
U_4(c) = \sum_{d \in \text{Dom}(D)} E_3(c, d) \cdot U_3(d)
\]
\[
U_5 = \sum_{c \in \text{Dom}(C)} U_2(c) \cdot U_4(c)
\]
Example: Factorized Computation of \( \text{COUNT}(Q_4) \)

This computation strategy corresponds to the following query rewriting:

\[
\sum_{a \in \text{Dom}(A)} \sum_{b \in \text{Dom}(B)} \sum_{c \in \text{Dom}(C)} \sum_{d \in \text{Dom}(D)} \sum_{e \in \text{Dom}(E)} V_1(a) \cdot E_1(b, a) \cdot E_2(c, b) \cdot E_3(c, d) \cdot E_4(d, e) \cdot V_2(e)
\]

\[
= \sum_{c \in \text{Dom}(C)} \left( \sum_{b \in \text{Dom}(B)} E_2(c, b) \cdot \left( \sum_{a \in \text{Dom}(A)} V_1(a) \cdot E_1(b, a) \right) \right) \cdot \left( \sum_{d \in \text{Dom}(D)} E_3(c, d) \cdot \left( \sum_{e \in \text{Dom}(E)} E_4(d, e) \cdot V_2(e) \right) \right)
\]
Experiments published in several papers, here a quick glimpse from [ANNOS17]

<table>
<thead>
<tr>
<th>Retailer dataset (records)</th>
<th>excerpt (17M)</th>
<th>full (86M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL computing the join</td>
<td>50.63 sec</td>
<td>216.56 sec</td>
</tr>
<tr>
<td>FDB computing both the join and the aggregates</td>
<td>25.51 sec</td>
<td>380.31 sec</td>
</tr>
<tr>
<td>Number of aggregates (scalar+group-by)</td>
<td>595+2,418</td>
<td>595+145k</td>
</tr>
<tr>
<td>FDB computing both the join and the aggregates</td>
<td>132.43 sec</td>
<td>1,819.80 sec</td>
</tr>
<tr>
<td>Number of aggregates (scalar+group-by)</td>
<td>158k+742k</td>
<td>158k+37M</td>
</tr>
</tbody>
</table>

In this experiment:

- FDB only used one core of a commodity machine
- For both PostgreSQL and FDB, the dataset was entirely in memory
- The aggregates represent gradients (or parts thereof) used for learning degree 1 and 2 polynomial regression models
Functional Aggregate Queries
FAQ generalizes factorized aggregate computation to a host of problems.

We use the following notation ($i \in [n] = \{1, \ldots, n\}$):

- $X_i$ are variables,
- $x_i$ are values in discrete domain $\text{Dom}(X_i)$
- $x = (x_1, \ldots, x_n) \in \text{Dom}(X_1) \times \cdots \times \text{Dom}(X_n)$
- For any $S \subseteq [n]$,

\[
x_S = (x_i)_{i \in S} \in \prod_{i \in S} \text{Dom}(X_i)
\]

E.g. $x_{\{2,5,8\}} = (x_2, x_5, x_8) \in \text{Dom}(X_2) \times \text{Dom}(X_5) \times \text{Dom}(X_8)$
**Functional Aggregate Query: The Problem**

\[
\varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}
\]

\[
\begin{align*}
\psi_{14}(X_1, X_4) \\
\psi_{13}(X_1, X_3) \\
\psi_{23}(X_2, X_3) \\
\psi_{124}(X_1, X_2, X_4)
\end{align*}
\]
**Functional Aggregate Query: The Input**

\[
\varphi(x_3) = \sum x_1 \prod x_2 \max x_4 \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}
\]

- **n variables** \(X_1, \ldots, X_n\)
- a multi-hypergraph \(\mathcal{H} = (\mathcal{V}, \mathcal{E})\)
  - Each vertex is a variable (notation overload: \(\mathcal{V} = [n]\))
  - To each hyperedge \(S \in \mathcal{E}\) there corresponds a factor \(\psi_S\)

\[
\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D
\]
Functional Aggregate Query: The Input

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4,2,3,1,3,1,4} \]

\[ \psi_{14}(X_1, X_4) \]
\[ \psi_{13}(X_1, X_3) \]
\[ \psi_{23}(X_2, X_3) \]
\[ \psi_{124}(X_1, X_2, X_4) \]

FAQ-expression

All functions have the same range \( D \)

\[ \varphi(x_3) \]

\[ V = \{1, 2, 3, 4\} \]
\[ E = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \]

- \( n \) variables \( X_1, \ldots, X_n \)
- a multi-hypergraph \( H = (V, E) \)
  - Each vertex is a variable (notation overload: \( V = [n] \))
  - To each hyperedge \( S \in E \) there corresponds a factor \( \psi_S \)

\[ \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D \]
**Functional Aggregate Query: The Input**

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,1,4} \]

- **n variables** \(X_1, \ldots, X_n\)
- A multi-hypergraph \(\mathcal{H} = (V, E)\)
  - Each vertex is a variable (notation overload: \(V = [n]\))
  - To each hyperedge \(S \in E\) there corresponds a factor \(\psi_S\)

\[ \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D \]

\[ \mathcal{V} = \{1, 2, 3, 4\} \]
\[ \mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \]
**Functional Aggregate Query: The Input**

- **n variables** $X_1, \ldots, X_n$
- a multi-hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - Each vertex is a variable (notation overload: $\mathcal{V} = [n]$)
  - To each hyperedge $S \in \mathcal{E}$ there corresponds a factor $\psi_S$

\[
\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow \mathbb{D}
\]

All functions have the same range $\mathbb{D}$

\[
\phi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4,3} \psi_{1,3} \psi_{1,4} \\
\psi_{14}(X_1, X_4) \\
\psi_{13}(X_1, X_3) \\
\psi_{23}(X_2, X_3) \\
\psi_{124}(X_1, X_2, X_4)
\]

\[
\mathcal{V} = \{1, 2, 3, 4\} \\
\mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\}
\]
### Functional Aggregate Query: The Input

\[ \psi_{14}(X_1, X_4) \]
\[ \psi_{13}(X_1, X_3) \]
\[ \psi_{23}(X_2, X_3) \]
\[ \psi_{124}(X_1, X_2, X_4) \]

FAQ-expression

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

\[ \varphi(x_3) \]

All functions have the same range \( D \)

\[ \mathcal{V} = \{1, 2, 3, 4\} \]
\[ \mathcal{E} = \{\{1, 4\}, \{1, 3\}, \{2, 3\}, \{1, 2, 4\}\} \]

- **n variables** \( X_1, \ldots, X_n \)
- **a multi-hypergraph** \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \)
  - Each vertex is a variable (notation overload: \( \mathcal{V} = [n] \))
  - To each hyperedge \( S \in \mathcal{E} \) there corresponds a **factor** \( \psi_S \)

\[ \psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D \]

\[ \mathbb{R}_+, \{\text{true, false}\}, \{0, 1\}, 2^\mathcal{U}, \text{etc.} \]
Functional Aggregate Query: The Input

- **n variables** $X_1, \ldots, X_n$
- a multi-hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$
  - Each vertex is a variable (notation overload: $\mathcal{V} = [n]$)
  - To each hyperedge $S \in \mathcal{E}$ there corresponds a factor $\psi_S$

\[
\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D
\]

- $\mathcal{R}_+, \{\text{true, false}\}, \{0, 1\}, 2^U$, etc.

- a set $F \subseteq \mathcal{V}$ of free variables (wlog, $F = [f] = \{1, \ldots, f\}$)
Functional Aggregate Query: The Output

\[ \varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

- Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D. \)

All functions have the same range \( D \).
**Functional Aggregate Query: The Output**

\[
\varphi(x_3) = \sum_{x_1} \prod_{x_2} \max_{x_4} \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4}
\]

- Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D \).
- \( \varphi \) defined by the FAQ-expression

\[
\varphi(x_{[f]}) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \text{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
\]

All functions have the same range \( D \).
**Functional Aggregate Query: The Output**

\[ \varphi(x_3) = \sum x_1 \prod x_2 \max x_4 \psi_{1,2,4} \psi_{2,3} \psi_{1,3} \psi_{1,4} \]

- All functions have the same range \( D \)

- Compute the function \( \varphi : \prod_{i \in F} \text{Dom}(X_i) \to D \).
- \( \varphi \) defined by the FAQ-expression

\[
\varphi(x_{[f]}) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \text{Dom}(X_n)}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
\]

- For each \( \bigoplus^{(i)} \)
**Functional Aggregate Query: The Output**

- Compute the function $\varphi : \prod_{i \in F} \text{Dom}(X_i) \to D$.
- $\varphi$ defined by the **FAQ-expression**

$$\varphi(x_3) = \sum x_1 \prod x_2 \max x_4 \psi_{1,2,4}\psi_{2,3}\psi_{1,3}\psi_{1,4}$$

- All functions have the same range $D$

- For each $\bigoplus^{(i)}$
  - Either $(D, \bigoplus^{(i)}, \otimes)$ is a **commutative semiring**
**Functional Aggregate Query: The Output**

- Compute the function $\varphi : \prod_{i \in F} \text{Dom}(X_i) \rightarrow D$.
- $\varphi$ defined by the FAQ-expression

$$\varphi(x_{[f]}) = \bigoplus_{x_{f+1} \in \text{Dom}(X_{f+1})}^{(f+1)} \cdots \bigoplus_{x_{n-1} \in \text{Dom}(X_{n-1})}^{(n-1)} \bigoplus_{x_n \in \text{Dom}(X_n)}^{(n)} \otimes \psi_S(x_S)$$

- For each $\bigoplus^{(i)}$
  - Either $\left(D, \bigoplus^{(i)}, \otimes\right)$ is a **commutative semiring**
  - Or $\bigoplus^{(i)} = \otimes$

All functions have the same range $D$.
Semirings

- \((D, \oplus, \otimes)\) is a **commutative semiring** when
  
  **Additive identity**  \(0 \in D : 0 \oplus e = e \oplus 0 = e\)
  
  **Multiplicative identity**  \(1 \in D : 1 \otimes e = e \otimes 1 = e\)
  
  **Annihilation by 0**  \(0 \otimes e = e \otimes 0 = 0\)
  
  **Distributive law**  \(a \otimes b \oplus a \otimes c = a \otimes (b \oplus c)\)
Semirings

- \((D, \oplus, \otimes)\) is a **commutative semiring** when

  - **Additive identity** \(0 \in D : 0 \oplus e = e \oplus 0 = e\)
  - **Multiplicative identity** \(1 \in D : 1 \otimes e = e \otimes 1 = e\)
  - **Annihilation by 0** \(0 \otimes e = e \otimes 0 = 0\)
  - **Distributive law** \(a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c\)

- Common examples (there are many more!)
  - **Boolean** \(\{\text{true, false}\}, \lor, \land\)
  - **sum-product** \((\mathbb{R}, +, \times)\)
  - **max-product** \((\mathbb{R}_+, \max, \times)\)
  - **set** \((2^U, \cup, \cap)\)
Problem (SumProduct)

*Given a commutative semiring \((D, \oplus, \otimes)\), compute the function*

\[
\varphi(x_1, \ldots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
\]
Problem (SumProduct)

*Given a commutative semiring \((D, \oplus, \otimes)\), compute the function*

\[
\varphi(x_1, \ldots, x_f) = \bigoplus_{x_{f+1}} \bigoplus_{x_{f+2}} \cdots \bigoplus_{x_n} \bigotimes_{S \in \mathcal{E}} \psi_S(x_S)
\]

- **SumProduct**
  - Rina Dechter (Artificial Intelligence 1999 and earlier)

- **Marginalize a Product Function**
Many examples for SumProduct

- $\{\text{true, false}\}, \lor, \land$
  - Constraint satisfaction problems
  - Boolean conjunctive query evaluation
  - SAT
  - $k$-colorability
  - etc.

- $(U, \cup, \cap)$
  - Conjunctive query evaluation

- $(\mathbb{R}, +, \times)$
  - Permanent
  - DFT
  - Inference in probabilistic graphical models
  - #$\text{CSP}$
  - Matrix chain multiplication
  - Aggregates in DB

- $(\mathbb{R}_+, \text{max}, \times)$
  - MAP queries in probabilistic graphical models
Boolean Conjunctive Queries:

- Boolean query $\Phi$ with set $\text{rels}(\Phi)$ of relation symbols
- Each relation symbol $R \in \text{rels}(\Phi)$ has variables $\text{vars}(R)$

$$\Phi = \exists X_1 \ldots \exists X_n : \bigwedge_{R \in \text{rels}(\Phi)} R(\text{vars}(R))$$

FAQ encoding:

$$\phi = \bigvee_{x} \bigwedge_{S \in \mathcal{E}} \psi_S(x_S), \text{ where}$$

- $\Phi$ has the hypergraph $(\mathcal{V}, \mathcal{E})$ with
- $\mathcal{V} = \bigcup_{R \in \text{rels}(\Phi)} \text{vars}(R)$ and $\mathcal{E} = \{\text{vars}(R) \mid R \in \text{rels}(\Phi)\}$
- For each $S \in \mathcal{E}$, there is a factor $\psi_S$ such that $\psi_S(x_S) = (x_S \in R)$
SumProduct Example 2: Matrix Chain Multiplication

Compute the product $A = A_1 \cdots A_n$ of $n$ matrices

- Each matrix $A_i$ is over field $\mathbb{F}$ and has dimensions $p_i \times p_{i+1}$

FAQ encoding:

- We use $n + 1$ variables $X_1, \ldots, X_{n+1}$ with domains $\text{Dom}(X_i) = [p_i]$
- Each matrix $A_i$ can be viewed as a function of two variables:

$$
\psi_{i,i+1} : \text{Dom}(X_i) \times \text{Dom}(X_{i+1}) \to \mathbb{F}, \text{ where } \psi_{i,i+1}(x, y) = (A_i)_{xy}
$$

The problem is now to compute the FAQ expression

$$
\phi(x_1, x_{n+1}) = \sum_{x_2 \in \text{Dom}(X_2)} \cdots \sum_{x_n \in \text{Dom}(X_n)} \prod_{i \in [n]} \psi_{i,i+1}(x_i, x_{i+1}).
$$
SumProduct Example 3: Queries in Graphical Models

- Discrete undirected graphical model represented by a hypergraph \((\mathcal{V}, \mathcal{E})\)
- \(\mathcal{V} = \{X_1, \ldots, X_n\}\) consists of \(n\) discrete random variables
- There is a factor \(\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow \mathbb{R}_+\) for each edge \(S \in \mathcal{E}\)

**FAQ expression to compute the marginal Maximum A Posteriori estimates:**

\[
\phi(x_1, \ldots, x_f) = \max_{x_{f+1} \in \text{Dom}(X_{f+1})} \cdots \max_{x_n \in \text{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(x_S)
\]

**FAQ expression to compute the marginal distribution of variables \(X_1, \ldots, X_f\):**

\[
\phi(x_1, \ldots, x_f) = \sum_{x_{f+1} \in \text{Dom}(X_{f+1})} \cdots \sum_{x_n \in \text{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(x_S)
\]

For conditional distributions \(p(x_A \mid x_B)\), the variables \(X_B\) are set to values \(x_B\).
Example 1: FAQ Computation using InsideOut

\[ \varphi(x_1, x_2, x_4) = \sum_{x_3,x_5,x_6,x_7,x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]

\[ \rho^*(\varphi) = 4, \quad \text{s}(\varphi) = 2, \quad \text{fhtw}(\varphi) = 1. \]

The above variable order has the free variables \( x_1, x_2, x_4 \) on top of the others and \( \text{fhtw}(\Delta) = 1. \)

The query result has size: \( O(N) \) when factorized; \( O(N^2) \) when listed.
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[\rho^*(\varphi) = 4, \ s(\varphi) = 2, \ fhtw(\varphi) = 1. \] The above variable order \( \Delta \) has the free variables \( x_1, x_2, x_4 \) on top of the others and \( fhtw(\Delta) = 1. \]

The query result has size: \( O(N) \) when factorized; \( O(N^2) \) when listed.
Example 1: FAQ Computation using InsideOut

\[ \varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]
Example 1: FAQ Computation using InsideOut

\[ \varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \left( \sum_{x_3} \psi_1(x_1, x_2, x_3) \right) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]

\[ \psi_6(x_1, x_2) \]
Example 1: FAQ Computation using InsideOut

\[\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)\]

\[\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N)\]
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \quad \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \left( \sum_{x_8} \psi_4(x_6, x_8) \right) \cdot \psi_5(x_5, x_7)
\]

\[
\psi_7(x_6)
\]
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \left( \sum_{x_7} \psi_5(x_5, x_7) \right)
\]

\[
\psi_8(x_5)
\]
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \tilde{O}(N)
\]
Example 1: FAQ Computation using InsideOut

\[
\begin{align*}
\varphi(x_1, x_2, x_4) &= \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \\
\varphi(x_1, x_2, x_4) &= \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \\
\varphi(x_1, x_2, x_4) &= \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \\
\varphi(x_1, x_2, x_4) &= \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \tilde{O}(N) \\
\varphi(x_1, x_2, x_4) &= \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \left( \sum_{x_6} \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \right) \cdot \psi_8(x_5) \\
\varphi(x_1, x_2, x_4) &= \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \left( \sum_{x_6} \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \right) \cdot \psi_8(x_5) \\
\psi_9(x_4, x_5)
\end{align*}
\]
Example 1: FAQ Computation using InsideOut

\[ \varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \tilde{O}(N) \]
Example 1: FAQ Computation using InsideOut

\[
\varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \tilde{O}(N)
\]

\[
\varphi(x_1, x_2, x_4) = \psi_6(x_1, x_2) \cdot \left( \sum_{x_5} \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \right)
\]

\[
\psi_{10}(x_2, x_4)
\]
Example 1: FAQ Computation using InsideOut

\[ \varphi(x_1, x_2, x_4) = \sum_{x_3, x_5, x_6, x_7, x_8} \psi_1(x_1, x_2, x_3) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7, x_8} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_4(x_6, x_8) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6, x_7} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_5(x_5, x_7) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5, x_6} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_3(x_4, x_5, x_6) \cdot \psi_7(x_6) \cdot \psi_8(x_5) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \sum_{x_5} \psi_6(x_1, x_2) \cdot \psi_2(x_2, x_4, x_5) \cdot \psi_9(x_4, x_5) \cdot \psi_8(x_5) \tilde{O}(N) \]

\[ \varphi(x_1, x_2, x_4) = \psi_6(x_1, x_2) \cdot \psi_{10}(x_2, x_4) \tilde{O}(N) \]
Example 2: FAQ Computation with Indicator Projections

$$\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)$$

$\rho^*(\varphi) = 2.5$, $s(\varphi) = 1.5$, $fhtw(\varphi) = 1.5$. The above variable order $\Delta$ has the free variable $x_1$ on top of the others and $fhtw(\Delta) = 1.5$.

The (unary) query result has size $O(N)$ when factorized or listed.
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \rho^*(\varphi) = 2.5, \ s(\varphi) = 1.5, \ fhtw(\varphi) = 1.5. \] The above variable order \( \Delta \) has the free variable \( x_1 \) on top of the others and \( fhtw(\Delta) = 1.5 \).

- The (unary) query result has size \( O(N) \) when factorized or listed.
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]
Example 2: FAQ Computation with Indicator Projections

\[
\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)
\]

\[
\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \left( \sum_{x_3} \psi_1'(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4'(x_1) \cdot \psi_6'(x_1) \right) \cdot \psi_7(x_1, x_2)
\]

\[
\psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)
\]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \tilde{O}(N^{1.5}) \]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \tilde{O}(N^{1.5}) \]

\[ \varphi(x_1) = \sum_{x_4, x_5} \left( \sum_{x_2} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi'_4(x_1) \cdot \psi'_6(x_1) \right) \cdot \psi_8(x_1) \]

\[ \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \tilde{O}(N^{1.5}) \]

\[ \varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \quad \tilde{O}(N) \]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \tilde{O}(N^{1.5}) \]

\[ \varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \tilde{O}(N) \]

\[ \varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \left( \sum_{x_5} \psi_8'(x_1) \cdot \psi_4'(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \right) \]

\[ \psi_9(x_1, x_4) \]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \sim \tilde{O}(N^{1.5}) \]

\[ \varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \sim \tilde{O}(N) \]

\[ \varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \sim \tilde{O}(N^{1.5}) \]
Example 2: FAQ Computation with Indicator Projections

\[ \varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \]

\[ \varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \sim O(N^{1.5}) \]

\[ \varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \sim O(N) \]

\[ \varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \sim O(N^{1.5}) \]

\[ \varphi(x_1) = \psi_8(x_1) \cdot \left( \sum_{x_4} \psi_8'(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \right) \]

\[ \psi_{10}(x_1) \]
Example 2: FAQ Computation with Indicator Projections

\[
\varphi(x_1) = \sum_{x_2, x_3, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_2(x_2, x_3) \cdot \psi_3(x_3, x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1)
\]

\[
\varphi(x_1) = \sum_{x_2, x_4, x_5} \psi_1(x_1, x_2) \cdot \psi_7(x_1, x_2) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \tilde{O}(N^{1.5})
\]

\[
\varphi(x_1) = \sum_{x_4, x_5} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_5(x_4, x_5) \cdot \psi_6(x_5, x_1) \tilde{O}(N)
\]

\[
\varphi(x_1) = \sum_{x_4} \psi_8(x_1) \cdot \psi_4(x_1, x_4) \cdot \psi_9(x_1, x_4) \tilde{O}(N^{1.5})
\]

\[
\varphi(x_1) = \psi_8(x_1) \cdot \psi_{10}(x_1) \tilde{O}(N)
\]
Outline

Part 1. Joins

Part 2. Aggregates

Part 3. Optimization
Why solving optimization problems aka *analytics* inside the database?

1. Bring analytics close to data
   ⇒ Save non-trivial export/import time

2. Large chunks of analytics code can be rewritten into SumProduct FAQs
   ⇒ Use scalable/factorized query processing

Hot topic in the current DB research & industry landscape:

- Very recent tutorials and research agenda \([A17,KBY17,PRWZ17]\)
- This tutorial highlights our recent work \([SOC16,ANNOs17]\)
In-database vs. Out-of-database Analytics

- $h$ and $g$ are functions over features and respectively model parameters
- $\theta^*$ are the parameters of the learned model
Plan for Part 3 on Optimization

- We will first introduce the main technical ideas via an example
  - Train a linear regression model using batch gradient descent
  - Express gradient computation as database queries
  - Re-parameterize the model under functional dependencies

- We will then discuss a generalization
  - Polynomial regression, factorization machines, classification

- We will conclude with complexity & experimental analysis
  - Model training faster than computing the input to external ML library!
In-Database Analytics Approach in This Tutorial

Unified in-database analytics solution for a host of optimization problems.

Deployed in industrial retail-planning and forecasting applications

- Typical databases have weekly sales, promotions, and products

- Training dataset = Result of a feature extraction query over the database

- Task = Train parameterized model to predict, e.g., additional demand generated for a product due to promotion

- Training algorithm = First-order optimization algorithm, e.g., batch or stochastic gradient descent
Retail Example
Simplified Retail Example

- Database \( I = (R_1, R_2, R_3, R_4, R_5) \)
- Feature selection query \( Q: \)
  \[
  Q(\text{sku}, \text{store}, \text{color}, \text{city}, \text{country}, \text{unitsSold}) =
  
  R_1(\text{sku}, \text{store}, \text{day}, \text{unitsSold}), R_2(\text{sku}, \text{color}),
  
  R_3(\text{day}, \text{quarter}), R_4(\text{store}, \text{city}), R_5(\text{city}, \text{country}).
  
  \]
- Free variables
  - Categorical (qualitative): \( F = \{\text{sku, store, color, city, country}\} \).
  - Continuous (quantitative): \( \text{unitsSold} \).
- Bounded variables
  - Categorical (qualitative): \( B = \{\text{day, quarter}\} \)

We learn the ridge linear regression model \( \langle \theta, x \rangle = \sum_{f \in F} \langle \theta_f, x_f \rangle \) over \( D = Q(I) \) with feature vector \( x \) and response \( y_{\text{unitsSold}} \).

The parameters \( \theta \) are obtained by minimizing the square loss function:

\[
J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y_{\text{unitsSold}})^2 + \frac{\lambda}{2} \| \theta \|_2^2
\]
Recap: One-hot encoding of categorical variables

- **Continuous** variables are mapped to scalars
  - \( y_{\text{unitsSold}} \in \mathbb{R} \).

- **Categorical** variables are mapped to indicator vectors
  - Say variable country has categories vietnam and england.
  - The variable country is then mapped to an indicator vector
    \[ x_{\text{country}} = [x_{\text{vietnam}}, x_{\text{england}}]^\top \in (\{0, 1\}^2)^\top. \]
  - \( x_{\text{country}} = [0, 1]^\top \) for a tuple with country = ‘‘england’’

One-hot encoding leads to very wide training datasets and many 0-values.
Recap: Role of the Least Square Loss Function

Goal: Describe a linear relationship $\text{fun}(x) = \theta_1 x + \theta_0$ between variables $x$ and $y = \text{fun}(x)$, so we can estimate new $y$ values given new $x$ values.

- We are given $n$ (black) data points $(x_i, y_i)_{i \in [n]}$
- We would like to find a (red) regression line $\text{fun}(x)$ such that the (green) error $\sum_{i \in [n]} (\text{fun}(x_i) - y_i)^2$ is minimized
- The role of the $\ell_2$-regularization $\|\theta\|_2^2 = \theta_0^2 + \theta_1^2$ is to avoid over/under-fitting. It gives preference to functions $\text{fun}$ with smaller norms.
We can solve $\theta^* := \arg \min_\theta J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$
We can solve $\theta^* := \arg\min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in |F|}$, the vector $c = (c_i)_{i \in |F|}$, and the scalar $s_Y$:

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^{\top} \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$
We can solve $\theta^* := \arg\min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

$$\theta := \theta - \alpha \cdot \nabla J(\theta).$$

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in |F|}$, the vector $c = (c_i)_{i \in |F|}$, and the scalar $s_Y$:

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_ix_j^\top \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$

Then,

$$J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \|\theta\|^2_2$$

$$= \frac{1}{2} \theta^\top \Sigma \theta - \langle \theta, c \rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\theta\|^2_2$$

$$\nabla J(\theta) = \Sigma \theta - c + \lambda \theta$$
Expressing $\Sigma$, $c$, $s_Y$ as SumProduct FAQ Queries

FAQ queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i$, $x_j$ continuous ⇒ FAQ query with no free variable

$$
\psi_{ij} = \sum_{f \in F : a_f \in \text{Dom}(x_f)} \sum_{b \in B : a_b \in \text{Dom}(x_b)} a_i \cdot a_j \cdot \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})}
$$

- $x_i$ categorical, $x_j$ continuous ⇒ FAQ query with one free variable

$$
\psi_{ij}[a_i] = \sum_{f \in F - \{i\} : a_f \in \text{Dom}(x_f)} \sum_{b \in B : a_b \in \text{Dom}(x_b)} a_j \cdot \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})}
$$

- $x_i$, $x_j$ categorical ⇒ FAQ query with two free variables

$$
\psi_{ij}[a_i, a_j] = \sum_{f \in F - \{i, j\} : a_f \in \text{Dom}(x_f)} \sum_{b \in B : a_b \in \text{Dom}(x_b)} \prod_{k \in [5]} 1_{R_k(a_{S(R_k)})}
$$

$S(R_k)$ is the set of variables of $R_k$; $a_{S(R_k)}$ is a tuple in relation $R_k$; $1_E$ is the Kronecker delta that is 1 (0) whenever the event $E$ holds (does not hold).
Expressing $\Sigma$, $c$, $s_Y$ as SQL Queries

SQL queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i, x_j$ continuous $\Rightarrow$ SQL query with no group-by attribute
  
  $$\text{SELECT SUM}(x_i * x_j) \text{ FROM } D;$$

- $x_i$ categorical, $x_j$ continuous $\Rightarrow$ SQL query with one group-by attribute
  
  $$\text{SELECT } x_i, \text{ SUM}(x_j) \text{ FROM } D \text{ GROUP BY } x_i;$$

- $x_i, x_j$ categorical $\Rightarrow$ SQL query with two free variables
  
  $$\text{SELECT } x_i, x_j, \text{ SUM}(1) \text{ FROM } D \text{ GROUP BY } x_i, x_j;$$

- $\Sigma$, $c$, $s_Y$ are all aggregates that can be computed inside the database!

- We avoid one-hot/sparse encoding of the input data.
Consider the functional dependency city → country

- There is one country for each city.

Assume we have:

- vietnam, england as categories for country
- saigon, hanoi, oxford, leeds, bristol as categories for city

The one-hot encoding enforces the following identities:

- \( x_{\text{vietnam}} = x_{\text{saigon}} + x_{\text{hanoi}} \)
  
  That is: If country is vietnam, then city is either saigon or hanoi
  if \( x_{\text{vietnam}} = 1 \) then either \( x_{\text{saigon}} = 1 \) or \( x_{\text{hanoi}} = 1 \)

- \( x_{\text{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}} \)
  
  That is: If country is england, then city is either oxford, leeds, or bristol
  if \( x_{\text{england}} = 1 \) then either \( x_{\text{oxford}} = 1 \) or \( x_{\text{leeds}} = 1 \) or \( x_{\text{bristol}} = 1 \)
Dimensionality Reduction with Functional Dependencies

- **Identities due to one-hot encoding**
  \[
  x_{\text{vietnam}} = x_{\text{saigon}} + x_{\text{hanoi}}
  \]
  \[
  x_{\text{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}
  \]

- **Encode** \( x_{\text{country}} \) **as** \( x_{\text{country}} = Rx_{\text{city}} \), **where**
  \[
  R = \begin{bmatrix}
  1 & 1 & 0 & 0 & 0 & \text{vietnam} \\
  0 & 0 & 1 & 1 & 1 & \text{england}
  \end{bmatrix}
  \]

  For instance, if city is saigon, i.e., \( x_{\text{city}} = [1, 0, 0, 0, 0]^T \),
  then country is vietnam, i.e., \( x_{\text{country}} = Rx_{\text{city}} = [1, 0]^T \).

\[
\begin{bmatrix}
  1 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  1 \\
  0 \\
  0 \\
  0
\end{bmatrix} = \begin{bmatrix}
  1 \\
  0
\end{bmatrix}
\]
Dimensionality Reduction with Functional Dependencies

- Functional dependency: \( \text{city} \rightarrow \text{country} \)
- \( x_{\text{country}} = Rx_{\text{city}} \)
- Replace all occurrences of \( x_{\text{country}} \) by \( Rx_{\text{city}} \):

\[
\sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle = \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle = \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \left( R^\top \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right)_{\gamma_{\text{city}}}
\]

We avoid computing aggregates over \( x_{\text{country}} \).

We reparameterize the problem and ignore parameters \( \theta_{\text{country}} \).

What about the penalty term in the loss function?
Dimensionality Reduction with Functional Dependencies

- Functional dependency: city → country
- $x_{\text{country}} = Rx_{\text{city}}$
- Replace all occurrences of $x_{\text{country}}$ by $Rx_{\text{city}}$:

$$\sum_{f \in F - \{\text{city}, \text{country} \}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle$$

$$= \sum_{f \in F - \{\text{city}, \text{country} \}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle$$

$$= \sum_{f \in F - \{\text{city}, \text{country} \}} \langle \theta_f, x_f \rangle + \left( R^\top \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right)$$

- We avoid computing aggregates over $x_{\text{country}}$.
- We reparameterize the problem and ignore parameters $\theta_{\text{country}}$.
- What about the penalty term in the loss function?
Dimensionality Reduction with Functional Dependencies

- Functional dependency: $\text{city} \rightarrow \text{country}$
- $x_{\text{country}} = Rx_{\text{city}}$
- $\gamma_{\text{city}} = R^T \theta_{\text{country}} + \theta_{\text{city}}$
- Rewrite the penalty term

$$\left\| \theta \right\|_2^2 = \sum_{j \neq \text{city}} \left\| \theta_j \right\|_2^2 + \left\| \gamma_{\text{city}} - R^T \theta_{\text{country}} \right\|_2^2 + \left\| \theta_{\text{country}} \right\|_2^2$$

- "Optimize out" $\theta_{\text{country}}$ by expressing it in terms of $\gamma_{\text{city}}$:

$$\theta_{\text{country}} = (I_{\text{country}} + RR^T)^{-1} R \gamma_{\text{city}} = R(I_{\text{city}} + R^T R)^{-1} \gamma_{\text{city}}$$

$I_{\text{country}}$ is the order-$N_{\text{country}}$ identity matrix and similarly for $I_{\text{city}}$.

- The penalty term becomes

$$\left\| \theta \right\|_2^2 = \sum_{j \neq \text{city}} \left\| \theta_j \right\|_2^2 + \left\langle (I_{\text{city}} + R^T R)^{-1} \gamma_{\text{city}}, \gamma_{\text{city}} \right\rangle$$
The General Picture
General Problem Formulation

A typical machine learning task is to solve $\theta^* := \arg\min_{\theta} J(\theta)$, where

$$J(\theta) := \sum_{(x,y) \in D} \mathcal{L}(\langle g(\theta), h(x) \rangle, y) + \Omega(\theta).$$

- $\theta = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$ are parameters
- functions $g : \mathbb{R}^p \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ for $n$ numeric features, $m > 0$
  - $g = (g_j)_{j \in [m]}$ is a vector of multivariate polynomials
  - $h = (h_j)_{j \in [m]}$ is a vector of multivariate monomials
- $\mathcal{L}$ is a loss function, $\Omega$ is the regularizer
- $D$ is the training dataset with features $x$ and response $y$.

Example problems: ridge linear regression, degree-$d$ polynomial regression, degree-$d$ factorization machines; logistic regression, SVM; PCA.
Special Case: Ridge Linear Regression

General problem formulation:

\[
J(\theta) := \sum_{(x,y) \in D} \mathcal{L}(\langle g(\theta), h(x) \rangle, y) + \Omega(\theta).
\]

Under

- square loss \( \mathcal{L} \), \( \ell_2 \)-regularization,
- data points \( x = (x_0, x_1, \ldots, x_n, y) \),
- \( p = n + 1 \) parameters \( \theta = (\theta_0, \ldots, \theta_n) \),
- \( x_0 = 1 \) corresponds to the bias parameter \( \theta_0 \)
- \( g \) and \( h \) identity functions \( g(\theta) = \theta \) and \( h(x) = x \),

we obtain the following formulation for ridge linear regression:

\[
J(\theta) := \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \| \theta \|_2^2.
\]
Special Case: Degree-\(d\) Polynomial Regression

General problem formulation:

\[
J(\theta) := \sum_{(x,y) \in D} \mathcal{L}(\langle g(\theta), h(x) \rangle, y) + \Omega(\theta).
\]

Under

- square loss \(\mathcal{L}\), \(\ell_2\)-regularization,
- data points \(x = (x_0, x_1, \ldots, x_n, y)\),
- \(p = m = 1 + n + n^2 + \cdots + n^d\) parameters \(\theta = (\theta_a)\), where \(a = (a_1, \ldots, a_n)\) is a tuple of non-negative integers such that \(\|a\|_1 \leq d\).
- \(g(\theta) = \theta\),
- the components of \(h\) are given by \(h_a(x) = \prod_{i=1}^n x_i^{a_i}\).

we obtain the following formulation for polynomial regression:

\[
J(\theta) := \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle g(\theta), h(x) \rangle - y)^2 + \frac{\lambda}{2} \|\theta\|_2^2.
\]
Special Case: Factorization Machines

Under

- square loss $\mathcal{L}$, $\ell_2$-regularization,
- data points $x = (x_0, x_1, \ldots, x_n, y)$,
- $p = m = 1 + n + r \cdot n$ parameters and $m = 1 + n + \binom{n}{2}$ features

we obtain the following formulation for degree-2 rank-$r$ factorization machines:

$$J(\theta) := \frac{1}{2|D|} \sum_{(x, y) \in D} \left( \sum_{i=0}^{n} \theta_i x_i + \sum_{\{i,j\} \in \left[\frac{n}{2}\right]} \theta_i^{(\ell)} \theta_j^{(\ell)} x_i x_j - y \right)^2 + \frac{\lambda}{2} \|\theta\|^2_2.$$
Special Case: Classification methods

Examples: support vector machines, logistic regression, Adaboost

- Typically, the regularizer is $\frac{\lambda}{2} \| \theta \|^2_2$

- The response is now binary: $y \in \{\pm 1\}$

- The loss function $L(\gamma, y)$, where $\gamma := \langle g(\theta), h(x) \rangle$, takes the form:
  - $L(\gamma, y) = \max\{1 - y\gamma, 0\}$ for support vector machines (SVM),
  - $L(\gamma, y) = \log(1 + e^{-y\gamma})$ for logistic regression, and
  - $L(\gamma, y) = e^{-y\gamma}$ for Adaboost.
Batch Gradient Descent (BGD)

Repeatedly update $\theta$ in the direction of the gradient until convergence

\[
\begin{align*}
\theta & \leftarrow \text{a random point}; \\
\textbf{while} \not\text{ converged yet } & \textbf{do} \\
& \alpha \leftarrow \text{next step size}; \\
& d \leftarrow \nabla J(\theta); \\
& \textbf{while} \ (J(\theta - \alpha \cdot d) \geq J(\theta) - \frac{\alpha}{2} \cdot \|d\|^2_2) \textbf{ do} \ \alpha \leftarrow \alpha/2; / / \text{ line search} \\
& \theta \leftarrow \theta - \alpha \cdot d; \\
\textbf{end}
\end{align*}
\]

BGD needs:

- Computation of the gradient vector $\nabla J(\theta)$
  - Its data-dependent component is computed once for all iterations

- Point evaluation $J(\theta)$
  - A few times per iteration to adjust $\alpha$ using line search
Compute Parameters $\theta$ using BGD

Immediate extension of the linear regression case discussed before.

Define the matrix $\Sigma = (\sigma_{ij})_{i,j \in [m]}$, the vector $c = (c_i)_{i \in [m]}$, and the scalar $s_Y$ by

$$
\Sigma = \frac{1}{|D|} \sum_{(x,y) \in D} h(x)h(x)^\top
$$

$$
c = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot h(x)
$$

$$
s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.
$$

Under square loss $\mathcal{L}$ and $\ell_2$-regularization:

$$
J(\theta) = \frac{1}{2} g(\theta)^\top \Sigma g(\theta) - \langle g(\theta), c \rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\theta\|_2^2
$$

$$
\nabla J(\theta) = \frac{\partial g(\theta)^\top}{\partial \theta} \Sigma g(\theta) - \frac{\partial g(\theta)^\top}{\partial \theta} c + \lambda \theta
$$
Summing Up

Insight #1:

- \( \Sigma, c, s_Y \) are queries that can be computed inside the database!
  - They can take much less time than computing the feature extraction query

Insight #2:

- The training dataset has repeating data blocks as it satisfies the join dependencies given by the feature extraction query.
  - A factorized training dataset avoids this redundancy.

Insight #3:

- The training dataset has many functional dependencies in practice.
  - First learn a smaller, reparameterized model whose features functionally determine the left-out features, then map it back to the original model with both functionally determining and determined parameters
Zoom-in: **In-database** vs. **Out-of-database** Learning

**Queries:**
- $\sigma_{11}$
- $\sigma_{ij}$
- $c_1$
- ...

**DB**
- Feature extraction query
  - $R_1 \times \ldots \times R_k$

**ML tool**
- $|D|$

**Model**
- $\theta^*$
- $\theta$
- $J(\theta)$
- $\nabla J(\theta)$

**Query optimizer**
- Factorized query evaluation
  - Cost $\leq N^{faqw} \ll |D|$

**Gradient-descent**
- $\nabla J(\theta)$
- Converged?
  - Yes
  - No

**Model reformulation**
- $h$

**Cost**
- $|D|$
Complexity & Experimental Analysis
Complexity Analysis: The General Case

Complexity of learning models falls back to factorized computation of aggregates over joins

Let:

- \((\mathcal{V}, \mathcal{E})\) = hypergraph of \(Q\)
- \(N = \max_{R \in I} |R|\)
- \(|\sigma_{ij}| = \text{size of the sparse representation of the } \sigma_{ij} \text{ tensor}\)
- \(faqw(i, j) = \text{FAQ-width of the query that expresses } \sigma_{ij} \text{ over } Q\)

The tensors \(\sigma_{ij}\) and \(c_j\) can be sparsely represented by queries with group-by variables and can be computed in time

\[
\tilde{O} \left( |\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [m]} (N^{faqw(i,j)} + |\sigma_{ij}|) \right).
\]
Complexity Analysis: Continuous Features Only

Complexity in the general case: \[\tilde{O}\left(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [m]} (N^{faqw(i,j)} + |\sigma_{ij}|)\right).\]

Complexity in case all features are continuous: \[\tilde{O}(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot m^2 \cdot N^{fhtw}).\]

In this case, \textit{faqw}(i,j) becomes the fractional hypertree width \textit{fhtw} of \textit{Q}. 

\[\text{[ANNOS17]}\]

\[\text{[SOC16]}\]
Complexity Analysis: Comparison with State of the Art

Let:
- \( d \) = degree of polynomial regression model
- \( c \) = max number of variables in any monomial in \( h \); \( c \leq d \)
- \( \rho^* \) = fractional edge cover number of query \( Q \)

Comparison against state of the art: [ANNOS17]
- \( \text{faqw}(i, j) \leq \text{fhtw} + c - 1 \) and \( |\sigma_{ij}| \leq \min\{|D|, N^c\} \).

For any query \( Q \) with \( \rho^* > \text{fhtw} + c - 1 \), there are infinitely many database instances of size \( N \) for which

\[
\lim_{N \to \infty} \frac{|D|}{\sum_{i,j \in [m]} (N^{\text{faqw}(i,j)} + |\sigma_{ij}|) \log N} = \infty.
\]

Computing \( \sigma_{ij} \) for degree-\( d \) polynomial regression takes

\[
\tilde{O}(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot m^2 \cdot N^{\text{fhtw}+2d}).
\]

under \textbf{one-hot encoding} of categorical variables.
Experiments published in several papers, here a glimpse from [ANNOS17]

<table>
<thead>
<tr>
<th>Retailer dataset (records)</th>
<th>excerpt (17M)</th>
<th>full (86M)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear Regression</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of features (Cont. + Categ.)</td>
<td>33+55</td>
<td>33+3702</td>
</tr>
<tr>
<td>MADlib (ols)</td>
<td>1,898.35 sec</td>
<td>–</td>
</tr>
<tr>
<td>PostgreSQL + R (qr decomposition)</td>
<td>798.96 sec</td>
<td>–</td>
</tr>
<tr>
<td>FDB</td>
<td>25.53 sec</td>
<td>380.31 sec</td>
</tr>
<tr>
<td><strong>Polynomial Regression degree 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of features (Cont. + Categ.)</td>
<td>562+2363</td>
<td>562+154K</td>
</tr>
<tr>
<td>MADlib</td>
<td>&gt; 22h</td>
<td>–</td>
</tr>
<tr>
<td>PostgreSQL + R</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>FDB</td>
<td>135.7 sec</td>
<td>2039.31 sec</td>
</tr>
</tbody>
</table>

- We measure end-to-end performance: joins + aggregates + convergence
- R: "-" means R’s data frame limit is exceeded and cannot run.
- MADlib: "-" means it cannot one-hot encode the data in a relation with more than 1600 columns.
One idea to rule them all

and at their core FACTORIZE them!
Thank you!
Quizzes
QUIZ 1: Joins (1/3)

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

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

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 1: Joins (2/3)

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

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

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 1: Joins (3/3)

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

1. Hypertree decomposition and variable order for query.
2. The fractional edge cover number and the fractional hypertree width (assume all relations have the same size).

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 $\{X_1, \ldots, X_n\} - \{X_i\}$:

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

$LW_3$ is the triangle query.
For each of the following functional aggregate queries:

1. Give a hypertree decomposition and variable order.

2. If you were to compute it as stated below (with all sums done after the products), what would be its time complexity? (Assume all functions have the same size.)

3. Is there an equivalent rewriting of $\varphi$ that would allow for quadratic time complexity? What about linear time?

The $n$-hop query:

$$\varphi(x_1, x_{n+1}) = \sum_{x_2, \ldots, x_n} \psi_1(X_1, X_2) \cdot \psi_2(X_2, X_3) \cdot \psi_3(X_3, X_4) \cdot \ldots \cdot \psi_n(X_n, X_{n+1}).$$
QUIZ 2: Aggregates (2/2)

For each of the following functional aggregate queries:

1. Give a hypertree decomposition and variable order.

2. If you were to compute it as stated below (with all sums done after the products), what would be its time complexity? Assume all functions have the same size.

3. Is there an equivalent rewriting of $\varphi$ that would allow for quadratic time complexity? What about linear time?

Query:

$$\varphi = \sum_{a} \sum_{b} \sum_{c} \sum_{f} \sum_{d} \sum_{e} \psi_1(a, b) \cdot \psi_2(a, c) \cdot \psi_3(c, d) \cdot \psi_4(b, c, e) \cdot \psi_5(e, f).$$
Assume that the natural join of the following relations provides the features we use to predict revenue:

Sales(store_id, product_id, quantity, revenue),
Product(product_id, color),
Store(store_id, distance_city_center).

Variables revenue, quantity, and distance_city_center stand for continuous features, while product_id and color for categorical features.

1. Give the FAQs required to compute the gradient of the squares loss function for learning a ridge linear regression models with the above features.

2. We know that product_id functionally determines color. Give a rewriting of the objective function that exploits the functional dependency.

3. The FAQs require the computation of a lot of common sub-problems. Can you think of ways to share as much computation as possible?
References
**References on Join Computation**

**GLS99** Hypertree decompositions and tractable queries.  
https://arxiv.org/abs/cs/9812022

**AGM08** Size bounds and query plans for relational joins.  
http://epubs.siam.org/doi/10.1137/110859440

**M10** Approximating fractional hypertree width.  
urlhttp://dl.acm.org/citation.cfm?id=1721845

**NPRR12** Worst-case optimal join algorithms: [extended abstract]  
urhttp://dl.acm.org/citation.cfm?id=2213565

**OZ12** Factorised representations of query results: size bounds and readability.  
http://dl.acm.org/citation.cfm?doid=2274576.2274607  

References on Join Computation

V14  **Triejoin: A Simple, Worst-Case Optimal Join Algorithm.**
     Veldhuizen. In ICDT 2014.

OZ15 **Size Bounds for Factorised Representations of Query Results.**
     http://dl.acm.org/citation.cfm?doid=2656335

CO15 **Worst-Case Optimal Join At A Time.**

ANS17 **What do Shannon-type inequalities, submodular width, and disjunctive Datalog have to do with one another?**
     https://arxiv.org/abs/1612.02503

KO17 **Covers of Query Results.**
     https://arxiv.org/abs/1709.01600
References on Aggregate Computation

**BKOZ13**  
Aggregation and Ordering in Factorised Databases.  
https://arxiv.org/abs/1307.0441

**ANR16**  
FAQ: Questions Asked Frequently.  
https://arxiv.org/abs/1504.04044
References on In-Database Analytics

**SOC16** Learning Linear Regression Models over Factorized Joins.
http://dl.acm.org/citation.cfm?doid=2882903.2882939

**A17** Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151).

**ANNOS17** In-Database Learning with Sparse Tensors.
https://arxiv.org/abs/1703.04780

**KBY17** Data Management in Machine Learning: Challenges, Techniques, and Systems.
Kumar, Boehm, Yang. In SIGMOD 2017, Tutorial.
https://www.youtube.com/watch?v=U8J0Dd_Z5wo

**NO17** Incremental View Maintenance with Triple Lock Factorisation Benefits.
https://arxiv.org/abs/1703.07484

**PRWZ17** Data Management Challenges in Production Machine Learning.
http://dl.acm.org/citation.cfm?doid=3035918.3054782