# A Layered Aggregate Engine for Analytics Workloads

fdbresearch.github.io relational.ai

**Maximilian Schleich**
University of Oxford

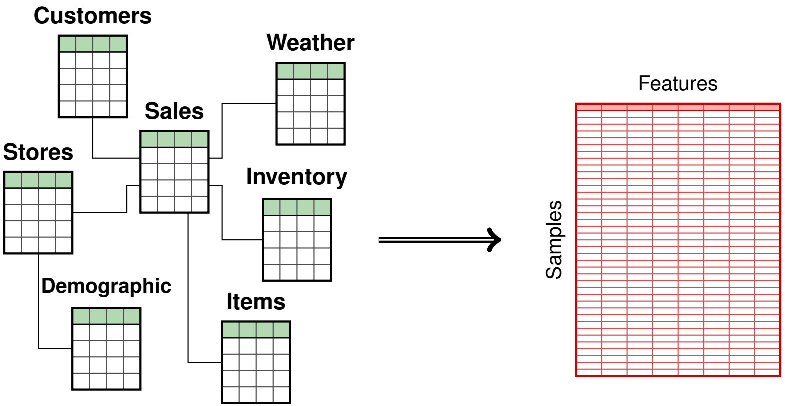
**Dan Olteanu**, University of Oxford
**Mahmoud Abo Khamis**, relationalAI
**Hung Q. Ngo**, relationalAI
**XuanLong Nguyen**, University of Michigan

AI for the enterprise

University of Washington July, 2019

# Recall relationalAI Keynote: Analytics over Databases

## Current State of Affairs in Analytics Workloads




- Carefully crafted by domain experts
- Comes with relational structure
- Throws away relational structure
- Can be order-of-magnitude larger

# Turn Analytics Workload into Database Workload!

Database Workload: **Batches of Aggregate Queries**

**Advantages**:

1. Use DB Tools for Optimization
2. Decompose Aggregates into Views over Join Tree
   - Pushing aggregate computation past joins
   - Using different roots and directional views
3. Avoid Materialization of Data Matrix

**Challenge**:

- Workloads require **many** aggregate queries

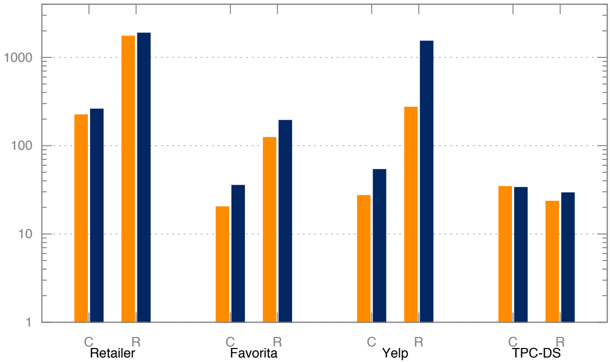
# Aggregates are at the Core of Analytics Workloads

| Workload | Query Batch | # Queries |
|---|---|---|
| Linear Regression<br>Covariance Matrix | `SUM(`$X_i$`*`$X_j$`)`<br>`SUM(`$X_i$`) GROUP BY` $X_j$<br>`COUNT(*) GROUP BY` $X_i, X_j$ | 140 |
| Regression Tree<br>(1 Node) | `VARIANCE(`$Y$`) WHERE` $X_j = c_j$ | 270 |
| Mutual Information<br>Chow-Liu Trees | `COUNT(*) GROUP BY` $X_i$<br>`COUNT(*) GROUP BY` $X_i, X_j$ | 106 |
| Data Cubes | `SUM(`$M$`) GROUP BY` $X_1, \ldots, X_d$ | 40 |

(# Queries shown for Favorita Kaggle dataset)

# Existing DBMSs are NOT Designed for Query Batches

Relative Speedup for **Our Approach** over **DBX** and **MonetDB**

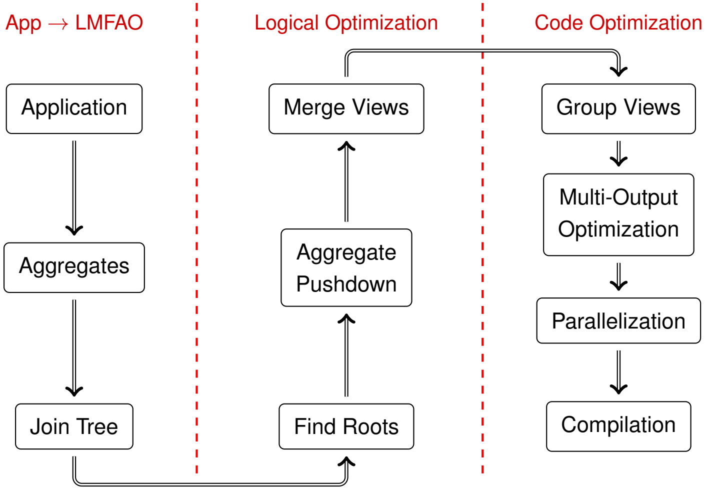



C = Covariance Matrix; R = Regression Tree Node; AWS d2.xlarge (4 vCPUs, 32GB)

# Tools of a Database Researcher

1. Exploit structure in the data
   - Algebraic structure: Factorized aggregate computation
   - Combinatorial structure: Query complexity measures
2. Sharing computation and data access
   - Aggregates decomposed into views over join tree
   - Share data access across views
3. Specialization for workload and data
   - Generate code specific to the query batch and dataset
   - Improve cache locality for hot data
4. Parallelization
   - Task and domain parallelism

# LMFAO: Layered Multi Functional Aggregate Optimization

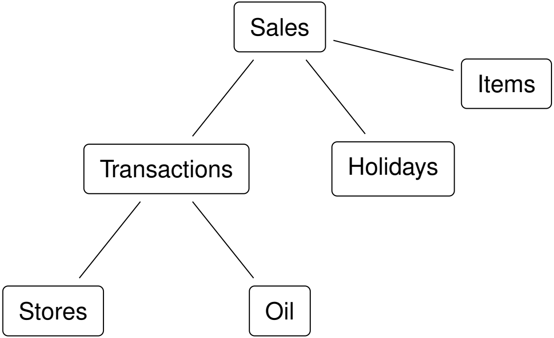


# The Layers of LMFAO: Logical Optimization

$Q_1$: $\texttt{SUM}\,(\texttt{units})$

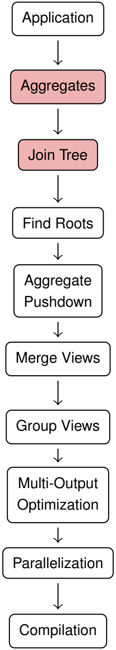
$Q_2$: $\texttt{SUM}\,(\texttt{item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM}\,(\texttt{units} \cdot \texttt{item})$ GROUP BY $\texttt{color}$




**Favorita Kaggle Dataset:**

Units sold for different items, stores, date.

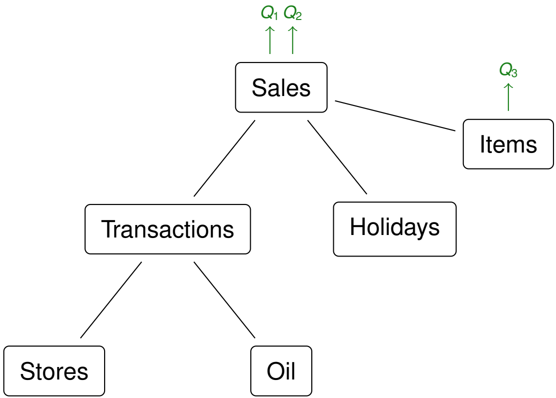


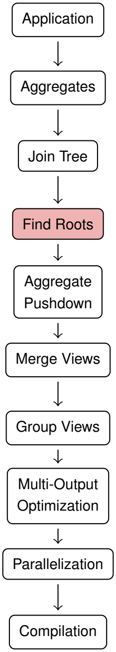
# The Layers of LMFAO: Logical Optimization

$Q_1$: $\texttt{SUM}\,(\texttt{units})$

$Q_2$: $\texttt{SUM}\,(\texttt{item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM}\,(\texttt{units} \cdot \texttt{item})$ GROUP BY $\texttt{color}$




**Find Roots Layer:**

For each query, decide its output (root) node.

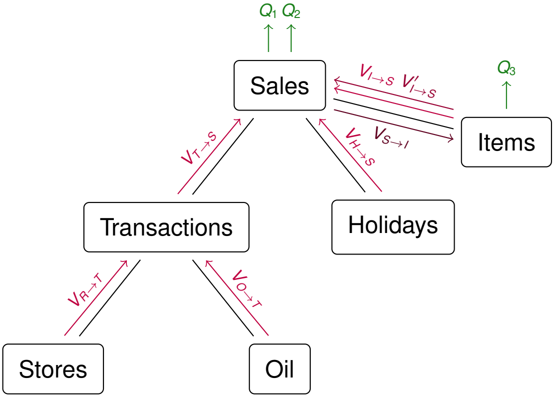
Choose root which minimizes sizes of views.

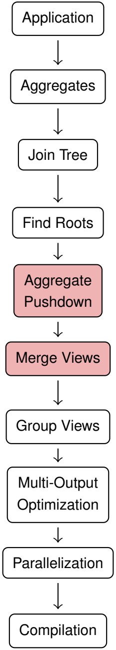
# The Layers of LMFAO: Logical Optimization

$Q_1$: $\texttt{SUM}\,(\texttt{units})$

$Q_2$: $\texttt{SUM}\,(\texttt{item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM}\,(\texttt{units} \cdot \texttt{item})$ GROUP BY $\texttt{color}$




**Aggregate Pushdown Layer:**

Break down each query into *directional views* over the join tree.

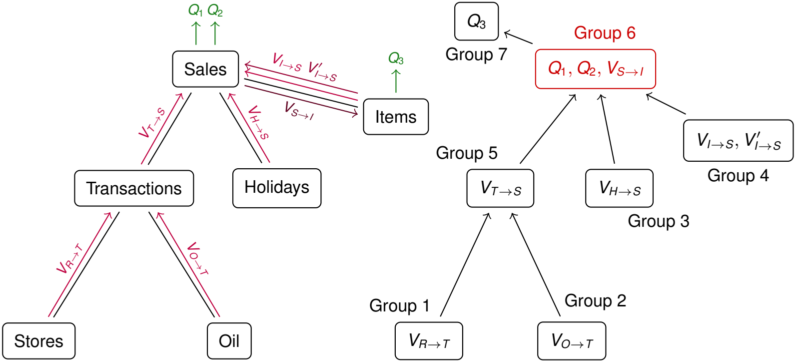
Reuse Partial Aggregates & **Merge Views** with same group-by attributes.

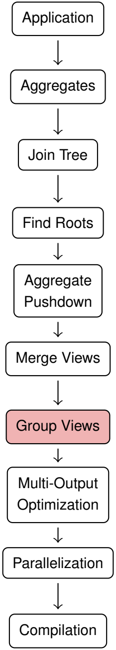
# The Layers of LMFAO: Code Optimization

$Q_1$: $\texttt{SUM}\,(\texttt{units})$

$Q_2$: $\texttt{SUM}\,(\texttt{item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM}\,(\texttt{units} \cdot \texttt{item})$ GROUP BY $\texttt{color}$




**Group Views Layer:**

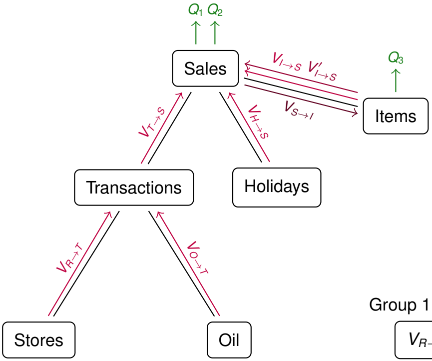
1. Construct Dependency Graph
2. Group Views that are computed over same relation

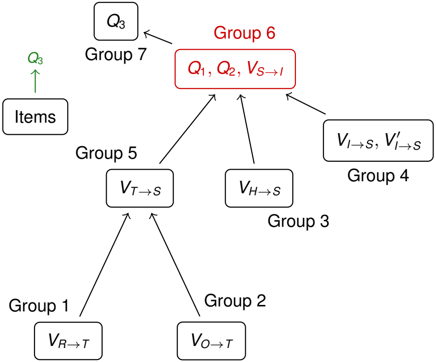
# The Layers of LMFAO: Code Optimization

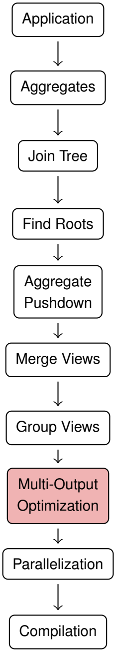
$Q_1$: $\texttt{SUM (units)}$

$Q_2$: $\texttt{SUM (item} \cdot f(\texttt{date, color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM (units} \cdot \texttt{item)}$ GROUP BY $\texttt{color}$




**Multi-Output Optimization Layer:**

View Group is a **computational unit** in LMFAO.

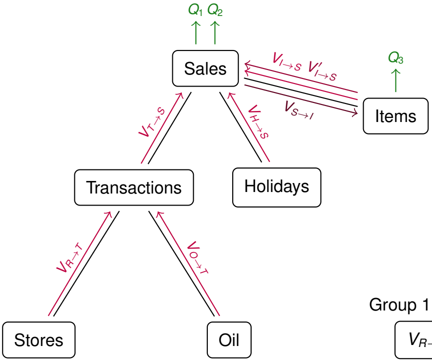
All views in one group are computed in one scan over the relation.

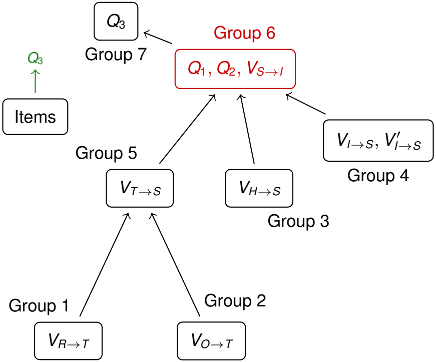
# The Layers of LMFAO: Code Optimization

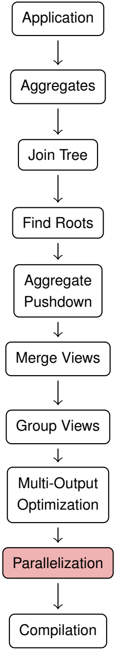
$Q_1$: $\texttt{SUM (units)}$

$Q_2$: $\texttt{SUM (item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM (units} \cdot \texttt{item)}$ GROUP BY $\texttt{color}$




**Parallelization Layer:**

Task parallelism: Evaluate independent groups in parallel

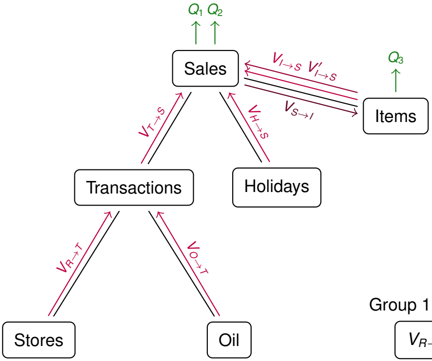
Domain parallelism: Partition the large relation used by each group

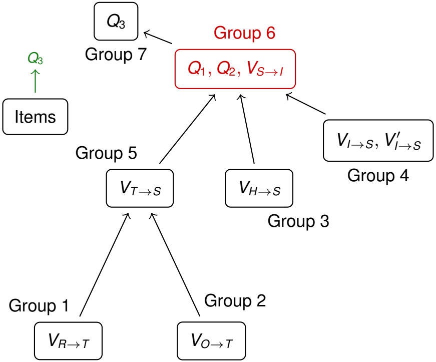
# The Layers of LMFAO: Code Optimization

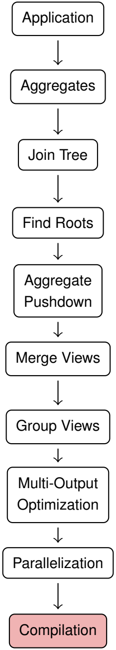
$Q_1$: $\texttt{SUM (units)}$

$Q_2$: $\texttt{SUM (item} \cdot f(\texttt{date}, \texttt{color}))$ GROUP BY $\texttt{store}$

$Q_3$: $\texttt{SUM (units} \cdot \texttt{item)}$ GROUP BY $\texttt{color}$




**Compilation Layer:**

Generate C++ code to compute each View Group.

# Code Generation for Executing View Group 6 over Sales


$Q_1$: `SUM (units)`

Traverse Sales as a trie following an order of its join attributes

# Code Generation for Executing View Group 6 over Sales

$V_I \rightarrow$ item, $V_I' \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V_I')$

    foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$

        foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$

$Q_1$: `SUM(units)`

Lookup into incoming views, e.g., $V_H$, as early as possible

# Code Generation for Executing View Group 6 over Sales

$V_I \rightarrow$ item
$V'_I \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

$\alpha_0 = 0;$
foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V'_I)$
  $\alpha_1 = V_I(i)$
  $\alpha_3 = 0;$
  foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$
    $\alpha_4 = V_H(d);$
    $\alpha_6 = 0;$
    foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$
      $\alpha_8 = V_T(d, s);\quad \alpha_9 = 0;$
      foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S : \alpha_9 \mathrel{+}= u;$
      $\alpha_6 \mathrel{+}= \alpha_8 \cdot \alpha_9;$
    $\alpha_3 \mathrel{+}= \alpha_4 \cdot \alpha_6;$
  $\alpha_0 \mathrel{+}= \alpha_1 \cdot \alpha_3$
$Q_1 = \alpha_0;$

$Q_1$: `SUM (units)`

Insert code for partial aggregates as early as possible

Reduces number of executed instructions

# Code Generation for Executing View Group 6 over Sales

$V_I, V'_I \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

$\alpha_0 = 0;$

foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V'_I)$

$\quad \alpha_1 = V_I(i)$

$\quad \alpha_2 = i;$

$\quad \alpha_3 = 0;$

$\quad$ foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$

$\quad\quad \alpha_4 = V_H(d);$

$\quad\quad \alpha_6 = 0;$

$\quad\quad$ foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$

$\quad\quad\quad \alpha_8 = V_T(d, s); \quad \alpha_9 = 0;$

$\quad\quad\quad$ foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S : \alpha_9 \mathrel{+}= u;$

$\quad\quad\quad \alpha_6 \mathrel{+}= \alpha_8 \cdot \alpha_9;$

$\quad\quad \alpha_3 \mathrel{+}= \alpha_4 \cdot \alpha_6;$

$\quad \alpha_0 \mathrel{+}= \alpha_1 \cdot \alpha_3 \qquad V_{S\rightarrow I}(i) = \alpha_3 \cdot \alpha_2;$

$Q_1 = \alpha_0;$

$V_{S\rightarrow I}$: `SUM (units · item) GROUP BY item`

Different outputs share partial aggregates

# Code Generation for Executing View Group 6 over Sales

$V_I, V_I' \rightarrow$ item

$V_H \rightarrow$ date

$V_T \rightarrow$ store

$\alpha_0 = 0;$

foreach $i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V_I')$

  $\alpha_1 = V_I(i)$

  $\alpha_2 = i;$

  $\alpha_3 = 0;$

  foreach $d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T)$

    $\alpha_4 = V_H(d); \quad \alpha_5 = 0;$

    foreach $c \in \pi_{\text{color}} \sigma_{\text{item}=i} V_I'$ : $\alpha_5 \mathrel{+}= f(d, c) \cdot V_I'(i, c);$

    $\alpha_6 = 0; \quad \alpha_7 = \alpha_5 \cdot \alpha_2 \cdot \alpha_4;$

    foreach $s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T)$

      $\alpha_8 = V_T(d, s); \quad \alpha_9 = 0; \quad \alpha_{10} = |\sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S|;$

      foreach $u \in \pi_{\text{units}} \sigma_{\text{item}=i,\text{date}=d,\text{store}=s} S$ : $\alpha_9 \mathrel{+}= u;$

      $\alpha_6 \mathrel{+}= \alpha_8 \cdot \alpha_9; \quad \alpha_{11} = \alpha_7 \cdot \alpha_8 \cdot \alpha_{10};$

      if $Q_2(s)$ then $Q_2(s) \mathrel{+}= \alpha_{11}$ else $Q_2(s) = \alpha_{11};$

    $\alpha_3 \mathrel{+}= \alpha_4 \cdot \alpha_6;$

  $\alpha_0 \mathrel{+}= \alpha_1 \cdot \alpha_3 \quad V_{S \rightarrow I}(i) = \alpha_3 \cdot \alpha_2;$

$Q_1 = \alpha_0;$

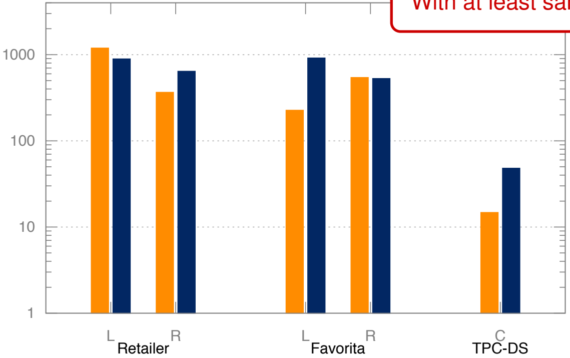
$Q_2$: `SUM (item` $\cdot\, f(\textit{date}, \textit{color})$`)   GROUP BY store`

Different outputs share partial aggregates

# Experimental Evaluation

Relative Speedup for **LMFAO** over **TensorFlow** and **MADlib**

With at least same accuracy!




L = Linear Regression; R = Regression Tree; C = Classification Tree;

TensorFlow learns only 1 Decision Tree Node. Intel i7-4770 (8 CPUs, 32GB)