Learning Models over Relational Databases

fdbresearch.github.io  relational.ai

Dan Olteanu
Oxford & relationalAI

FG DB Symposium 2020
Darmstadt, March 2020
Acknowledgments

FDB team, in particular:

Jakub  Max  Milos  Ahmet  Amir

relationalAI team, in particular:

Mahmoud  Hung  Long
Motivation: Relational Data is Ubiquitous

Kaggle Survey: Most Data Scientists use Relational Data at Work!

Overall

<table>
<thead>
<tr>
<th>Type</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>65.5%</td>
</tr>
<tr>
<td>Text</td>
<td>53%</td>
</tr>
<tr>
<td>Image</td>
<td>18.1%</td>
</tr>
<tr>
<td>Video</td>
<td>5.1%</td>
</tr>
<tr>
<td>Other</td>
<td>10.3%</td>
</tr>
</tbody>
</table>

By Industry

<table>
<thead>
<tr>
<th>Industry</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retail</td>
<td>86%</td>
</tr>
<tr>
<td>Marketing</td>
<td>83%</td>
</tr>
<tr>
<td>Insurance</td>
<td>82%</td>
</tr>
<tr>
<td>Financial</td>
<td>77%</td>
</tr>
</tbody>
</table>

• Massive adoption of the Relational Model in last decades

• Many human hours invested in building relational models

• Relational databases are rich with knowledge of the underlying domains
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
The learning time and accuracy of the model can be drastically improved by exploiting the structure and semantics of the underlying multi-relational database.
Current Landscape for ML over DB
No integration

The good:

1. Most DB+ML solutions operate in this space
2. Supports virtually any ML task
3. ML & DB distinct tools on the technology stack

The bad:

1. Materialisation of feature extraction query
2. DB exports data as one table, ML imports it in own format
3. One/multi-hot encoding of categorical variables

Examples:
PostgreSQL + R, Pandas + scikit-learn/TensorFlow, SparkSQL + MLlib, etc.
Loose integration

- DB supports ML tasks as UDF
- Same running process for DB and ML
- DB computes one table, ML works directly on it → No data export/import

Examples:
- MadLib supports comprehensive library of ML UDFs
- Bismark gives unified programming architecture for incremental gradient descent
Tight integration

Feature Extraction Query → DB → materialised output
= data matrix → ML Tool → \( \theta \)

Batch of Queries → Model Reformulation → Optimisation

Query Batch Evaluation → Structure-Aware Learning vs. Structure-Agnostic Learning

- Exploit relational structure and semantics
- Exploit database optimisations, e.g., push parts of ML tasks past joins
- One evaluation plan for mixed DB and ML workload
Structure-aware Learning FASTER even than Feature Extraction Query!
Case in Point (1): A Retailer Use Case

<table>
<thead>
<tr>
<th>Relation</th>
<th>Cardinality</th>
<th>Arity (Keys+Values)</th>
<th>File Size (CSV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory</td>
<td>84,055,817</td>
<td>3 + 1</td>
<td>2 GB</td>
</tr>
<tr>
<td>Items</td>
<td>5,618</td>
<td>1 + 4</td>
<td>129 KB</td>
</tr>
<tr>
<td>Stores</td>
<td>1,317</td>
<td>1 + 14</td>
<td>139 KB</td>
</tr>
<tr>
<td>Demographics</td>
<td>1,302</td>
<td>1 + 15</td>
<td>161 KB</td>
</tr>
<tr>
<td>Weather</td>
<td>1,159,457</td>
<td>2 + 6</td>
<td>33 MB</td>
</tr>
<tr>
<td>Join</td>
<td>84,055,817</td>
<td>3 + 41</td>
<td>23 GB</td>
</tr>
</tbody>
</table>
Train a linear regression model to predict *inventory* given all features

<table>
<thead>
<tr>
<th>PostgreSQL+TensorFlow</th>
<th>Time</th>
<th>Size (CSV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>–</td>
<td>2.1 GB</td>
</tr>
<tr>
<td>Join</td>
<td>152.06 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Export</td>
<td>351.76 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Shuffling</td>
<td>5,488.73 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Query batch</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Grad Descent</td>
<td>7,249.58 secs</td>
<td>–</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td><strong>13,242.13 secs</strong></td>
<td></td>
</tr>
</tbody>
</table>
Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict inventory given all features

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL+TensorFlow</th>
<th>Our approach (SIGMOD’19)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Size (CSV)</td>
</tr>
<tr>
<td>Database</td>
<td>–</td>
<td>2.1 GB</td>
</tr>
<tr>
<td>Join</td>
<td>152.06 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Export</td>
<td>351.76 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Shuffling</td>
<td>5,488.73 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Query batch</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Grad Descent</td>
<td>7,249.58 secs</td>
<td>–</td>
</tr>
<tr>
<td>Total time</td>
<td>13,242.13 secs</td>
<td>6.13 secs</td>
</tr>
</tbody>
</table>

2, 160 \times \text{ faster while being more accurate} (RMSE on 2\% test data)
Train a linear regression model to predict *inventory* given all features.

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL+TensorFlow</th>
<th>Our approach (SIGMOD'19)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Size (CSV)</td>
</tr>
<tr>
<td>Database</td>
<td>–</td>
<td>2.1 GB</td>
</tr>
<tr>
<td>Join</td>
<td>152.06 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Export</td>
<td>351.76 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Shuffling</td>
<td>5,488.73 secs</td>
<td>23 GB</td>
</tr>
<tr>
<td>Query batch</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Grad Descent</td>
<td>7,249.58 secs</td>
<td>–</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td><strong>13,242.13 secs</strong></td>
<td></td>
</tr>
</tbody>
</table>

2, 160× faster while being more accurate (RMSE on 2% test data)

TensorFlow trains one model. Our approach takes < 0.1 sec for any extra model over a subset of the given feature set.
Similar behaviour (or outright failure) for more:

- **datasets**: Favorita, TPC-DS, Yelp, Housing
- **systems**:
  - used in industry: R, scikit-learn, Python StatsModels, mlib, XGBoost, MADlib
  - academic prototypes: Morpheus, libFM
- **models**: decision trees, factorisation machines, $k$-means, ..

This is to be contrasted with the scalability of DBMSs!
How to achieve this performance improvement?
Idea 1: Turn the ML Problem into a DB Problem
<table>
<thead>
<tr>
<th>Workload</th>
<th>Query Batch</th>
<th># Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>$\text{SUM}(X_i \times X_j)$</td>
<td>814</td>
</tr>
<tr>
<td>Covariance Matrix</td>
<td>$\text{SUM}(X_i) \text{ GROUP BY } X_j$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{SUM}(1) \text{ GROUP BY } X_i, X_j$</td>
<td></td>
</tr>
<tr>
<td>Decision Tree (Regression, 1 Node)</td>
<td>$\text{VARIANCE}(Y) \text{ WHERE } X_j = c_j$</td>
<td>3,141</td>
</tr>
<tr>
<td>R$k$-means</td>
<td>$\text{SUM}(1) \text{ GROUP BY } X_j$</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>$\text{SUM}(1) \text{ GROUP BY } \text{Center}_1, \ldots, \text{Center}_k$</td>
<td></td>
</tr>
</tbody>
</table>

(# Queries shown for Retailer dataset with 39 attributes)

Queries in a batch:

- Same aggregates but over different attributes
- Expressed over the same join of the database relations

**AMPLE opportunities for sharing computation in a batch.**
Models under Consideration

So far:
- Polynomial regression
- Factorisation machines
- Classification/regression trees
- Mutual information
- Chow Liu trees
- $k$-means clustering
- $k$-nearest neighbours
- (robust, ordinal) PCA
- SVM

On-going:
- Boosting regression trees
- AdaBoost
- Sum-product networks
- Random forests
- Logistic regression
- Linear algebra:
  - QR decomposition
  - SVD
  - Low-rank matrix factorisation

All these cases can benefit from **structure-aware computation**
Ridge Linear Regression

Query Batch
Recap: Ridge Linear Regression

Linear regression model:

$$f_\theta(x) = \langle \theta, x \rangle = \theta_0 x_0 + \theta_1 x_1 + \ldots$$

- Training dataset $D$ defined by *feature extraction query*
  - A tuple $(x, y) \in D$ consists of feature vector $x$ and response $y$

- Parameters $\theta$ obtained by minimising the objective function:

$$J(\theta) = \frac{1}{2|D|} \sum_{(x, y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \|\theta\|^2_2$$
We can solve $\theta^* := \arg \min_{\theta} J(\theta)$ with batch-gradient descent:

repeat until convergence:

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

**Model reformulation idea:** Decouple

- data-dependent ($x, y$) computation from
- data-independent ($\theta$) computation

in the formulations of the objective $J(\theta)$ and its gradient $\nabla J(\theta)$. 
\[ J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \| \theta \|^2 \]

\[ = \frac{1}{2|D|} \left( \theta^\top \left( \sum_{(x,y) \in D} xx^\top \right) \theta - 2 \langle \theta, \sum_{(x,y) \in D} y \cdot x \rangle + \left( \sum_{(x,y) \in D} y^2 \right) \right) + \frac{\lambda}{2} \| \theta \|^2 \]

\[ = \frac{1}{2|D|} \left( \theta^\top \Sigma \theta - 2 \langle \theta, c \rangle + s_Y \right) + \frac{\lambda}{2} \| \theta \|^2 \]

\[ \nabla J(\theta) = \frac{1}{|D|} \left( \Sigma \theta - c \right) + \lambda \theta \]
\[ \Sigma, \mathbf{c}, s_Y \text{ can be Expressed as Batch of Queries} \]

Compute one query for each entry \[ \sum_{(x,y) \in D} x_i x_j^T \] in \( \Sigma \):

- For continuous variables, \( x_i, x_j \):
  
  \[ \text{SELECT SUM (x_i * x_j) FROM D}; \]

- For categorical \( x_i \) and continuous \( x_j \):
  
  \[ \text{SELECT x_i, SUM(x_j) FROM D GROUP BY x_i}; \]

- For categorical \( x_i, x_j \):
  
  \[ \text{SELECT x_i, x_j, SUM(1) FROM D GROUP BY x_i, x_j}; \]

where \( D \) is the feature extraction query over the input DB.
$\Sigma, c, s_Y$ can be Expressed as Batch of Queries

Compute one query for each entry $\sum_{(x,y) \in D} x_i x_j^T$ in $\Sigma$:

- $x_i, x_j$ continuous
  
  \[
  \text{SELECT} \ \text{SUM} (x_i \ast x_j) \ \text{FROM} \ D;
  \]

where $D$ is the feature extraction query over the input DB.
Compute one query for each entry \( \sum_{(x,y) \in D} x_i x_j^T \) in \( \Sigma \):

- \( x_i, x_j \) continuous
  
  \[
  \text{SELECT } \text{SUM} (x_i \ast x_j) \text{ FROM } D;
  \]

- \( x_i \) categorical, \( x_j \) continuous
  
  \[
  \text{SELECT } x_i, \text{SUM}(x_j) \text{ FROM } D \text{ GROUP BY } x_i;
  \]

where \( D \) is the feature extraction query over the input DB.
\[ \Sigma, \ c, \ s_Y \ \text{can be Expressed as Batch of Queries} \]

Compute one query for each entry \( \sum_{(x,y) \in D} x_i x_j^\top \) in \( \Sigma \):

- \( x_i, x_j \) continuous
  
  \[
  \text{SELECT } \text{SUM} (x_i \ast x_j) \text{ FROM } D;
  \]

- \( x_i \) categorical, \( x_j \) continuous
  
  \[
  \text{SELECT } x_i, \text{SUM}(x_j) \text{ FROM } D \text{ GROUP BY } x_i;
  \]

- \( x_i, x_j \) categorical
  
  \[
  \text{SELECT } x_i, x_j, \text{SUM}(1) \text{ FROM } D \text{ GROUP BY } x_i, x_j;
  \]

where \( D \) is the feature extraction query over the input DB.
Natural Attempt:

Use Existing DB System to Compute Query Batch
Existing DBMSs are **NOT** Designed for Query Batches

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

<table>
<thead>
<tr>
<th></th>
<th>Retailer</th>
<th>Favorita</th>
<th>Yelp</th>
<th>TPC-DS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td></td>
<td></td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
<td></td>
<td>C</td>
<td>R</td>
</tr>
</tbody>
</table>

*C = Covariance Matrix; R = Regression Tree Node; AWS d2.xlarge (4 vCPUs, 32GB)*
Idea 2: Exploit Problem Structure to Lower Complexity
Algebraic structure: (semi)rings \((\mathcal{R}, +, \ast, 0, 1)\)

- Distributivity law ➔ **Factorisation**
  - Factorised Databases [VLDB’12+, TODS’15, SIGREC’16]
  - Factorised Machine Learning [SIGMOD’16+’19, DEEM’18, PODS’18+’19, TODS’20]

- Additive inverse ➔ **Uniform treatment of updates**
  - Factorised Incremental Maintenance [SIGMOD’18+’20]

- Sum-Product abstraction ➔ **Same processing for distinct tasks**
  - DB queries, Covariance matrix, PGM inference, Matrix chain multiplication [SIGMOD’18+’19]
Combinatorial structure: query width and data degree measures

- **Width measure** $w$ for FEQ → **Low complexity** $\tilde{O}(N^w)$
  
  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  
  worst-case optimal size and time for factorised joins
  
  [ICDT'12+’18, TODS’15, PODS’19, TODS’20]

- **Degree** → **Adaptive processing** depending on high/low degrees
  
  worst-case optimal incremental maintenance

  [ICDT’19a, PODS’20]

  evaluation of queries with negated relations of bounded degree

  [ICDT’19b]

- **Functional dependencies** → **Learn simpler, equivalent models**

  reparameterisation of polynomial regression models and factorisation machines

  [PODS’18, TODS’20]
Idea 3: Lower the Constant Factors
1. **Specialisation** for workload and data
   - Generate code specific to the query batch and dataset
   - Improve cache locality for hot data path

2. **Sharing low-level data access**
   - Aggregates decomposed into views over join tree
   - Share data access across views with different output schemas

3. **Parallelisation**: multi-core (SIMD & distribution to come)
   - Task and domain parallelism

[DEEM’18, SIGMOD’19, CGO’20]
Case in Point (3)

Code Optimisations

→

Non-trivial Speedup
IFAQ: Iterative Functional Aggregate Queries

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

Collections are Dictionaries or Sets

- Database relations are modeled as dictionaries

<table>
<thead>
<tr>
<th>Relation R(A,B)</th>
<th>Relation R(A,B) in IFAQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
</tr>
</tbody>
</table>

Inspired by the FAQ framework [PODS’16]
IFAQ: Iterative Functional Aggregate Queries

- $\sum$ for stateful computation over collection elements:

  \[ \sum_{e \in \text{set}} f(e) \quad \xrightarrow{\text{Compile}} \quad \text{for(auto& e : set)} \quad \text{res} += f(e); \]

- $\lambda$ for constructing dictionaries:

  \[ \lambda_{e \in \text{set}} f(e) \quad \xrightarrow{\text{Compile}} \quad \text{for(auto& e : set)} \quad \text{res}[e] = f(e); \]

- Supports while loops and conditionals
Running Example

Dataset with three relations:

- Sales(item, store, unit sales)
- Item(item, price)
- Store(store, city)

Learning Task:

Learn Linear Regression model to predict number of unit sales.

Training Dataset:

\[ Q(x) = S(x_S) \bowtie R(x_R) \bowtie I(x_I) \]
(Simplified) Linear Regression in IFAQ

Batch Gradient Descent:
Update $\theta$ in direction of gradient of square loss

let $F = [[i, s, p, c]]$ in

$\theta \leftarrow \theta_0$

while( not converged ) {

$\theta = \max_{f_1 \in F} \left( \theta(f_1) - \frac{\alpha}{|Q|} \sum_{x \in \sup(Q)} Q(x) \times \left( \sum_{f_2 \in F} \theta(f_2) \times x[f_2] - x[u] \right) \times x[f_1] \right)$

}

$\theta$
(Simplified) Linear Regression in IFAQ

Batch Gradient Descent:
Enter a description here.

Let \( F = [[i, s, p, c]] \) in

\[
\theta \leftarrow \theta_0
\]

While (not converged) {

\[
\theta = \sum_{f_1 \in F} \left( \theta(f_1) - \frac{\alpha}{|Q|} \sum_{x \in \text{sup}(Q)} Q(x) \left( \sum_{f_2 \in F} \theta(f_2) \cdot x[f_2] - x[u] \right) \cdot x[f_1] \right)
\]

Gradient of square loss

}\}

\( \theta \)

For simplicity and WLOG, we

1. set \( \frac{\alpha}{|Q|} = 1 \)
2. ignore \( x[u] \)
let \( F = [[i, s, p, c]] \) in

\[
\theta \leftarrow \theta_0
\]

while( not converged ) {

\[
\theta = \lambda \left( \theta(f_1) - \sum_{x \in \text{sup}(Q)} Q(x) * \left( \sum_{f_2 \in F} \theta(f_2) * x[f_2] \right) * x[f_1] \right)
\]

}

\( \theta \)
Transformation Rule: **Normalisation**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{x \in \sup(Q)} Q(x) \ast \sum_{f_2 \in F} \left( \theta(f_2) \ast x[f_2] \right) \ast x[f_1] \right)
\]
Transformation Rule: **Normalisation**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{x \in \text{sup}(Q)} Q(x) \sum_{f_2 \in F} \left( \theta(f_2) \ast x[f_2] \right) \ast x[f_1] \right)
\]
Transformation Rule: **Normalisation**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{x \in \text{sup}(Q)} \sum_{f_2 \in F} \left( Q(x) \ast \theta(f_2) \ast x[f_2] \ast x[f_1] \right) \right)
\]
Transformation Rule: **Loop Scheduling**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{x \in \text{sup}(Q)} \sum_{f_2 \in F} (Q(x) \ast \theta(f_2) \ast x[f_2] \ast x[f_1]) \right)
\]

Order loops by size of support
Transformation Rule: **Loop Scheduling**

$$\theta = \lambda \left( \theta(f_1) - \sum_{f_2 \in F} \sum_{x \in \text{sup}(Q)} (Q(x) \ast \theta(f_2) \ast x[f_2] \ast x[f_1]) \right)$$

Order loops by size of support
Transformation Rule: **Factorisation**

\[ \theta = \lambda \left( \theta(f_1) - \sum_{f_2 \in F} \sum_{x \in \text{sup}(Q)} (Q(x) * \theta(f_2) * x[f_2] * x[f_1]) \right) \]
Transformation Rule: **Factorisation**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{f_2 \in F} \sum_{x \in \text{sup}(Q)} \left( Q(x) \ast \theta(f_2) \ast x[f_2] \ast x[f_1] \right) \right)
\]

Less arithmetic operations
High-Level Optimisations

Transformation Rule: **Factorisation**

\[
\theta = \lambda \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \right) \right) \left( \sum_{x \in \text{sup}(Q)} \left( Q(x) \right) \right) \left( x[f_2] \right) \left( x[f_1] \right)
\]

Less arithmetic operations
Transformation Rule: **Static Memoisation**

\[
\theta \leftarrow \theta_0
\]

\[
\text{while( not converged )} \{
\theta = \lambda \left( \sum_{f_1 \in F} \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \cdot \sum_{x \in \sup(Q)} (Q(x) \cdot x[f_2] \cdot x[f_1]) \right)
\}
\]

\[
\theta
\]
Transformation Rule: **Static Memoisation**

\[
\theta \leftarrow \theta_0
\]

\[
\text{while( not converged )}\{
\theta = \lambda \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \ast \sum_{x \in \text{sup}(Q)} (Q(x) \ast x[f_2] \ast x[f_1]) \right)
\}
\]

\[
\theta
\]
Transformation Rule: **Code Motion**

\[
\theta \leftarrow \theta_0
\]

while( not converged ){

\[
\text{let } M = \lambda_{f_1 \in F} \lambda_{f_2 \in F} \sum_{x \in \text{sup}(Q)} Q(x) \ast x[f_2] \ast x[f_1] \text{ in }
\]

\[
\theta = \lambda_{f_1 \in F} \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \ast M(f_1)(f_2) \right)
\]

}\n
\[
\theta
\]

\textbf{M} defines the covariance matrix
High-Level Optimisations

Transformation Rule: **Code Motion**

\[ \theta \leftarrow \theta_0 \]

\[ \text{while( not converged )} \{ \]

\[ \text{let } M = \prod_{f_1 \in F} \prod_{f_2 \in F} \sum_{x \in \text{sup}(Q)} Q(x) \ast x[f_2] \ast x[f_1] \text{ in} \]

\[ \theta = \prod_{f_1 \in F} \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \ast M(f_1)(f_2) \right) \]

\[ \} \]

\[ \theta \]
Transformation Rule: **Code Motion**

$$
\text{let } M = \lambda f_1 \in F \lambda f_2 \in F \sum_{x \in \sup(Q)} Q(x) \cdot x[f_1] \cdot x[f_2] \text{ in}

\theta \leftarrow \theta_0

\text{while( not converged )}\

\quad \theta = \lambda f_1 \in F \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \cdot M(f_1)(f_2) \right)

\}

\theta$$
Expression after High-Level Optimisations:

\[
\text{let } M = \lambda f_1 \in F \lambda f_2 \in F \sum_{x \in \text{sup}(Q)} Q(x) \ast x[f_1] \ast x[f_2] \text{ in }
\]

\[
\theta \leftarrow \theta_0
\]

\[
\text{while( not converged )} \{
\theta = \lambda f_1 \in F (\theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \ast M(f_1)(f_2))
\}
\]

\[
\theta
\]
Transformation Rule: **Loop Unrolling**

\[ \text{let } M = \lambda f_1 \in F \lambda f_2 \in F \sum_{x \in \text{sup}(Q)} Q(x) \ast x[f_1] \ast x[f_2] \text{ in} \]

\[ \theta \leftarrow \theta_0 \]

while( not converged ){

\[ \theta = \lambda f_1 \in F \left( \theta(f_1) - \sum_{f_2 \in F} \theta(f_2) \ast M(f_1)(f_2) \right) \]

}

\[ \theta \]

Unroll Loops over statically known features \( F \)
Transformation Rule: **Loop Unrolling**

\[
\text{let } M = \prod_{f_1 \in F} \prod_{f_2 \in F} \sum_{x \in \text{sup}(Q)} Q(x) * x[f_1] * x[f_2] \text{ in }
\]

\[
\theta \leftarrow \theta_0
\]

\[
\text{while( not converged )} \{ 
\begin{array}{l}
\theta = \left\{ c \rightarrow (\theta(c) - (\ldots + \theta(c) * M(c)(c) + \theta(p) * M(c)(p)\ldots)) \right\}, \ldots \right\}
\end{array}
\]

\[
\theta
\]

Unroll Loops over statically known features \( F \)
Transformation Rule: **Loop Unrolling**

\[
\begin{align*}
\text{let } & \quad M = \left\{ \left\{ c \rightarrow \left\{ \ldots, p \rightarrow \sum_{x \in \text{sup}(Q)} Q(x) \times x[c] \times x[p], \ldots \right\}, \ldots \right\} \right\} \text{ in} \\
& \quad \theta \leftarrow \theta_0 \\
& \quad \text{while( not converged )} \\
& \quad \quad \theta = \left\{ \left\{ c \rightarrow \left( \theta(c) - \left( \ldots + \theta(c) \times M(c)(c) + \theta(p) \times M(c)(p) \ldots \right) \right), \ldots \right\} \right\} \\
& \quad \theta
\end{align*}
\]

- Convert dictionaries over \( F \) into records
- Dynamic accesses into static accesses
Transformation Rule: **Static Field Access**

\[
\text{let } M = \left\{ \ldots, p = \sum_{x \in \text{sup}(Q)} Q(x) \ast x.c \ast x.p, \ldots, \right\}, \ldots \right\} \text{ in }
\]

\[
\theta \leftarrow \theta_0
\]

\[
\text{while( not converged )}{
\theta = \left\{ c = \theta.c - \left( \ldots + \theta.c \ast M.c.c + \theta.p \ast M.c.p \ldots \right), \ldots \right\}
\}
\]

\[
\theta
\]

- Convert dictionaries over \( F \) into records
- Dynamic accesses into static accesses
Transformation Rule: **Aggregate Extraction**

let $M =$

$$\{ c = \{ ..., c = \sum_{x \in \sup(Q)} Q(x) \ast x \cdot c \ast x \cdot c, \ p = \sum_{x \in \sup(Q)} Q(x) \ast x \cdot c \ast x \cdot p, ... \} \}$$

,... in ...
Transformation Rule: **Aggregate Extraction**

\[
\text{let } M = \{ c = \{ \ldots, c = \sum_{x \in \sup(Q)} Q(x) \ast x \cdot c \ast x \cdot c, \quad p = \sum_{x \in \sup(Q)} Q(x) \ast x \cdot c \ast x \cdot p, \ldots \} \}\text{ in } \ldots
\]
Transformation Rule: **Aggregate Extraction**

\[
\text{let } M_{cc} = \sum_{x \in \text{sup}(Q)} Q(x) \ast x.c \ast x.c \text{ in }
\]

\[
\text{let } M_{cp} = \sum_{x \in \text{sup}(Q)} Q(x) \ast x.c \ast x.p \text{ in }
\]

\[
\text{let } M = \{ c = \{ \ldots, c = M_{cc}, p = M_{cp}, \ldots \}, \ldots \} \text{ in } \ldots
\]
Transformation Rule: **Aggregate Extraction**

\[
\text{let } M_{cc} = \sum_{x \in \text{sup}(Q)} \text{Q}(x) \ast x.c \ast x.c \text{ in}
\]

\[
\text{let } M_{cp} = \sum_{x \in \text{sup}(Q)} \text{Q}(x) \ast x.c \ast x.p \text{ in}
\]

\[
\text{let } M = \{ c = \{..., c = M_{cc}, p = M_{cp}, ...\}, ... \} \text{ in } ...
\]
Aggregate Query Optimisations

Transformation Rule: **Aggregate Extraction**

let $M_{cc} = \sum_{x \in \text{sup}(Q)} Q(x) \ast x.c \ast x.c$ in

let $M_{cp} = \sum_{x \in \text{sup}(Q)} Q(x) \ast x.c \ast x.p$ in

let $M = \{ c = \{ ..., c = M_{cc}, p = M_{cp}, ... \} \}, ... \}$ in ...

We can:
- avoid materialisation of $Q(x)$
- inline code for join computation
Fast Join Recap

To compute $S(x_S) \bowtie R(x_R)$ on variable $s$:

1. Construct nested dictionaries over $R$:

$$H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow \{\{x_R \rightarrow 1\}\}\}$$

For join value $s$, $H_R(s)$ maps to partition of $R$ with $s = x_R.s$
Fast Join Recap

To compute $S(x_S) \bowtie R(x_R)$ on variable $s$:

1. Construct nested dictionaries over $R$:

$$H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow \{x_R \rightarrow 1\}\}$$

For join value $s$, $H_R(s)$ maps to partition of $R$ with $s = x_R.s$

2. Iterate over $S$, and probe $H_R$ for joining tuples:

$$J_{S \bowtie R} = \sum_{x_S \in \text{sup}(S)} \sum_{x_R \in \text{sup}(H_R(\{s = x_S.s\}))} \{\{k \rightarrow S(x_S) \ast H_R(\{s = x_S.s\})(x_R)\}\}$$

let $k = \{s = x_S.s, \ i = x_S.i, \ u = x_S.u, \ c = x_R.c\}$ in

$$\{\{k \rightarrow S(x_S) \ast H_R(\{s = x_S.s\})(x_R)\}\}$$
Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

\[
\begin{align*}
H_R &= \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow \{x_R \rightarrow 1\}\}\} \\
H_I &= \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{\{i = x_I.i\} \rightarrow \{x_I \rightarrow 1\}\}\} \\
M_{cp} &= \sum_{x_S \in \text{sup}(S)} \sum_{x_R \in \text{sup}(H_R(\{s = x_S.s\}))} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \ast H_R(\{s = x_S.s\})(x_R) \ast H_I(\{i = x_S.i\})(x_I) \ast x_R.c \ast x_I.p
\end{align*}
\]
Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow \{x_R \rightarrow 1\}\}\]

\[
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{\{i = x_I.i\} \rightarrow \{x_I \rightarrow 1\}\}\]

\[
M_{cp} = \sum_{x_S \in \text{sup}(S)} \sum_{x_R \in \text{sup}(H_R(\{s = x_S.s\}))} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \ast H_R(\{s = x_S.s\})(x_R) \ast H_I(\{i = x_S.i\})(x_I) \ast x_R.c \ast x_I.p
\]

**Push aggregate** \( \sum_{x_R} x_R.c \) **into** \( H_R \)
Transformation Rule: **Aggregate Pushdown**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \times \{\{s = x_R.s\} \rightarrow x_R.c\}
\]

\[
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \times \{\{i = x_I.i\} \rightarrow \{x_I \rightarrow 1\}\}\}
\]

\[
M_{cp} = \sum_{x_S \in \text{sup}(S)} \sum_{x_R \in \text{sup}(H_R(\{s = x_S.s\}))} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \times H_R(\{s = x_S.s\})(x_R) \times H_I(\{i = x_S.i\})(x_I) \times x_I.p
\]
Aggregate Query Optimisations

**Transformation Rule:** Aggregate Pushdown

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \mapsto x_R.c\}
\]

\[
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{\{i = x_I.i\} \mapsto \{x_I \mapsto 1\}\}\}
\]

\[
M_{cp} = \sum_{x_S \in \text{sup}(S)} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \ast H_R(\{s = x_S.s\}) \ast H_I(\{i = x_S.i\})(x_I) \ast x_I.p
\]
Transformation Rule: **Aggregate Pushdown**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow x_R.c\}
\]

\[
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{\{i = x_I.i\} \rightarrow \{x_I \rightarrow 1\}\}\}
\]

\[
M_{cp} = \sum_{x_S \in \text{sup}(S)} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \ast H_R(\{s = x_S.s\}) \ast H_I(\{i = x_S.i\})(x_I) \ast x_I.p
\]

Push aggregate \(\sum_{x_I} x_I.p\) into \(H_I\)
Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s \rightarrow x_R.c\}\}
\]

\[
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{\{i = x_I.i \rightarrow x_I.p\}\}
\]

\[
M_{cp} = \sum_{x_S \in \text{sup}(S)} \sum_{x_I \in \text{sup}(H_I(\{i = x_S.i\}))} S(x_S) \ast H_R(\{s = x_S.s\}) \ast H_I(\{i = x_S.i\})
\]
Transformation Rule: **Aggregate Pushdown**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{ \{ s = x_R.s \} \rightarrow x_R.c \} \\
H_I = \sum_{x_I \in \text{sup}(I)} I(x_I) \ast \{ \{ i = x_I.i \} \rightarrow x_I.p \} \\
M_{cp} = \sum_{x_S \in \text{sup}(S)} S(x_S) \ast H_R(\{ s = x_S.s \}) \ast H_I(\{ i = x_S.i \})
\]
Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

\[
\begin{align*}
H_R &= \sum_{x_R \in \text{sup}(R)} R(x_R) \cdot \{\{s = x_R.s \rightarrow x_R.c\}\}
\end{align*}
\]

\[
\begin{align*}
H_I &= \sum_{x_I \in \text{sup}(I)} I(x_I) \cdot \{\{i = x_I.i \rightarrow x_I.p\}\}
\end{align*}
\]

\[
\begin{align*}
M_{cp} &= \sum_{x_S \in \text{sup}(S)} S(x_S) \cdot H_R(\{s = x_S.s\}) \cdot H_I(\{i = x_S.i\})
\end{align*}
\]

\[
\begin{align*}
H'_R &= \sum_{x_R \in \text{sup}(R)} R(x_R) \cdot \{\{s = x_R.s \rightarrow x_R.c \cdot x_R.c\}\}
\end{align*}
\]

\[
\begin{align*}
H'_I &= \sum_{x_I \in \text{sup}(I)} I(x_I) \cdot \{\{i = x_I.i \rightarrow 1\}\}
\end{align*}
\]

\[
\begin{align*}
M_{cc} &= \sum_{x_S \in \text{sup}(S)} S(x_S) \cdot H'_R(\{s = x_S.s\}) \cdot H'_I(\{i = x_S.i\})
\end{align*}
\]

Similarly for \(M_{cc}\)
Transformation Rule: **Aggregate Fusion**

\[
H_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{ \{ s = x_R.S \} \rightarrow x_R.C \} \]

\[
H'_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{ \{ s = x_R.S \} \rightarrow x_R.C \ast x_R.C \} \]

Fuse \( H_R \) and \( H'_R \)
Transformation Rule: **Aggregate Fusion**

\[
H''_R = \sum_{x_R \in \text{sup}(R)} R(x_R) \ast \{\{s = x_R.s\} \rightarrow \{v_R = x_R.c, \ v'_R = x_R.c \ast x_R.c\}\}
\]

\(H''_R\) computes two aggregates
Aggregate Query Optimisations

Transformation Rule: **Aggregate Fusion**

\[
H''_R = \sum_{x_R \in \text{sup}(R)} R(x_R) * \{\{s = x_R.s\} \rightarrow \{v_R = x_R.c, \quad v'_R = x_R.c \times x_R.c\}\}
\]

\[
H''_I = \sum_{x_I \in \text{sup}(I)} I(x_I) * \{\{i = x_I.i\} \rightarrow \{v_I = x_I.p, \quad v'_I = 1\}\}
\]

Fuse \(H_I\) and \(H'_I\)
Transformation Rule: **Aggregate Fusion**

\[
H''_R = \sum_{x_R \in \text{sup}(R)} R(x_R) * \{\{s = x_R.s\} \rightarrow \{v_R = x_R.c, \ v'_R = x_R.c * x_R.c\}\} \\

H''_I = \sum_{x_I \in \text{sup}(I)} I(x_I) * \{\{i = x_I.i\} \rightarrow \{v_I = x_I.p, \ v'_I = 1\}\} \\

M_{cc, cp} = \sum_{x_S \in \text{sup}(S)} S(x_S) * \left( \\
\text{let } w_R = H''_R(\{s = x_S.s\}) \text{ in} \\
\text{let } w_I = H''_I(\{i = x_S.i\}) \text{ in} \\
\{m_{cp} = w_R.v_R * w_I.v_I, \ m_{cc} = w_R.v'_R * w_I.v'_I\} \right)
\]
Transformation Rule: **Trie Conversion**

\[ H''_R = \sum_{x_r \in \text{sup}(R)} R(x_r) \ast \{ \{ s = x_r.s \} \rightarrow \{ v_R = x_r.c, v'_R = x_r.c \ast x_r.c \} \} \]

\[ H''_I = \sum_{x_i \in \text{sup}(I)} I(x_i) \ast \{ \{ i = x_i.i \} \rightarrow \{ v_I = x_i.p, v'_R = 1 \} \} \]

\[ M_{cc, cp} = \sum_{x_S \in \text{sup}(S)} S(x_S) \ast \left( \text{let } w_R = H''_R(\{ s = x_S.s \}) \text{ in} \right.

\text{let } w_I = H''_I(\{ i = x_S.i \}) \text{ in}

\{ m_{cp} = w_R.v_R \ast w_I.v_I, \ m_{cc} = w_R.v'_R \ast w_I.v'_I \} \right) \]

**Turn relations into tries (i.e., nested dictionaries)**
Trie Conversion

Transformation Rule: **Trie Conversion**

\[
M_{cc,cp} = \sum_{x_S \in \text{sup}(S)} S(x_S) \ast \left( \begin{array}{l}
\text{let } w_R = H_{R''}(\{s = x_S.s\}) \text{ in} \\
\text{let } w_I = H_{I''}(\{i = x_S.i\}) \text{ in} \\
\{ m_{cp} = w_R.v_R \ast w_I.v_I, \quad m_{cc} = w_R.v'_R \ast w_I.v'_I \} 
\end{array} \right)
\]

Turn relation S into trie S’
Transformation Rule: **Trie Conversion**

\[
M_{cc, cp} = \sum_{x_S \in \text{sup}(S')} \sum_{x_i \in \text{sup}(S'(x_S))} S'(x_s)(x_i) \ast \left( \begin{array}{c}
\text{let } w_R = H''_R(\{s = x_S.s\}) \text{ in } \\
\text{let } w_I = H''_I(\{i = x_S.i\}) \text{ in } \\
\{m_{cp} = w_R \cdot v_R * w_I \cdot v_I, \quad m_{cc} = w_R \cdot v'_R * w_I \cdot v'_I\} \end{array} \right)
\]

One loop for each join variable
Transformation Rule: **Code Motion**

\[ M_{cc, cp} = \sum_{x_S \in \text{sup}(S')} \sum_{x_i \in \text{sup}(S'(x_S))} S'(x_S)(x_i) \times \left( \begin{array}{l}
\text{let } w_R = H''_R(\{s = x_S.s\}) \text{ in } \\
\text{let } w_I = H''_I(\{i = x_S.i\}) \text{ in } \\
\{ m_{cp} = w_R \cdot v_R \ast w_I \cdot v_I, \quad m_{cc} = w_R \cdot v'_R \ast w_I \cdot v'_I \} \end{array} \right) \]

Move up the look-up into \( H_R \).
Trie Conversion

Transformation Rule: **Code Motion**

\[
M_{cc,cp} = \sum_{x_S \in \text{sup}(S')} S'(x_S)(x_i) \ast \left( \begin{array}{c}
\text{let } w_R = H''_R(\{s = x_S.s\}) \text{ in } \\
\sum_{x_i \in \text{sup}(S'(x_S))} \text{let } w_I = H''_I(\{i = x_S.i\}) \text{ in } \\
\{ m_{cp} = w_R.v_R \ast w_I.v_I, \quad m_{cc} = w_R.v_R' \ast w_I.v_I' \} \end{array} \right)
\]

Move up the look-up into \( H_R \)
Transformation Rule: **Factorisation**

\[ M_{cc,cp} = \sum_{x_S \in \sup(S')} \]

\[ \text{let } w_R = H_R''(\{s = x_S.s\}) \text{ in } \]

\[ \sum_{x_i \in \sup(S'(x_S))} S'(x_S)(x_i) \times \left( \text{let } w_I = H_I''(\{i = x_S.i\}) \text{ in } \sum_{x_i \in \sup(S'(x_S))} S'(x_S)(x_i) \times \left( \text{let } w_R = H_R''(\{s = x_S.s\}) \text{ in } \sum_{x_i \in \sup(S'(x_S))} S'(x_S)(x_i) \times \left( \text{let } w_I = H_I''(\{i = x_S.i\}) \text{ in } \sum_{x_i \in \sup(S'(x_S))} S'(x_S)(x_i) \times \left( \text{let } w_R = H_R''(\{s = x_S.s\}) \text{ in } \sum_{x_i \in \sup(S'(x_S))} S'(x_S)(x_i) \times \left( \text{let } w_I = H_I''(\{i = x_S.i\}) \text{ in } \sum_{x_i \in \sup(S'(x_S))} \right) \right) \right) \right) \right) \]

\[ \{ m_{cp} = w_R.v_R \ast w_I.v_I, \quad m_{cc} = w_R.v_R' \ast w_I.v_I' \} \]
Trie Conversion

Transformation Rule: **Factorisation**

\[
M_{cc,cp} = \sum_{x_S \in \text{sup}(S')} \{ m_{cp} = \text{let } w_R = H''_R(\{ s = x_S.s \}) \text{ in } \{ m_{cp} = w_R \cdot v_R, \ m_{cc} = w_R \cdot v'_R \} \} \]

\[
\sum_{x_i \in \text{sup}(S'(x_s))} S'(x_s)(x_i) \cdot \left( \text{let } w_i = H''_I(\{ i = x_S.i \}) \text{ in } \{ m_{cp} = w_i \cdot v_i, \ m_{cc} = w_i \cdot v'_i \} \right)
\]

Less arithmetic operations
Engineering Tools of a Database Researcher

Relative Speedup of Code Optimisations

Added optimisations for covariance matrix computation:

specialisation $\rightarrow$ + sharing $\rightarrow$ + parallelisation

AWS d2.xlarge (4 vCPUs, 32GB)
Conclusion

Three-step recipe for efficient machine learning over databases:

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

Q.E.D.