Special Topics:
Self-Driving Database Management Systems
Autonomous Systems III

Kushagra Singh // 15-799 // Spring 2022
Discussion so far

- **Database Optimisation**
- Improve upon various aspects of the DB
  - Better physical layout decisions
  - Better knob configurations
Today

- Database Learning
- Not the same as database optimization
  - We’re not “tuning” database internals
  - Rather modelling the underlying data
Motivation

- Work done while executing queries is lost
- Some exceptions:
  - Caching tuples
  - View selection / Adaptive Indexing
Motivation

- Work done while executing queries is lost
- Some exceptions:
  - Caching tuples
  - View selection / Adaptive Indexing
- Can we reuse work done?
Database Learning

- Underlying data follows some unknown distribution
- Each query answer reveals information about the underlying distribution
- If we could perfectly model this distribution, we wouldn’t even need the database!

The answer to each query reveals some fuzzy knowledge about the answers to other queries, even if each query accesses a different subset of tuples and columns.
Database Learning

- Unfortunately, modelling the distribution perfectly is not always feasible.
- Approximate models can still be beneficial!
- Can improve sample based answers in AQP settings

We call the above goal *Database Learning* (DBL), as it is reminiscent of the inferential goal of Machine Learning (ML) whereby past observations are used to improve future predictions [15, 16, 66]. Likewise, our goal in DBL is to enable a similar principle by *learning from past observations, but in a query processing setting*. Specifically, in DBL, we plan to treat approximate answers to past queries as observations, and use them to refine our posterior knowledge of the underlying data, which in turn can be used to speed up future queries.
Each query answer reveals information about the underlying distribution
Database Learning

- Goal – Given previous query answers, improve future query answers
- How – Query answers could potentially be correlated; exploit this!
Database Learning

- Goal – Given previous query answers, improve future query answers
- How – Query answers could potentially be correlated; exploit this!
- No fancy deep learning
- Statistical approach – Principle of Maximum Entropy (1957)
Goal – Given previous query answers, improve future query answers

How – Query answers could potentially be correlated; exploit this!

No fancy deep learning

Statistical approach –
Principle of Maximum Entropy
Maybe contentious

Machine Learning

Watered down Statistics
Definitely contentious

Statistics

Watered down Math
Limited Edition; HMU

Modelling credits: Anon friend
Overall Architecture

- Query Synopsis
- Inference Model
- AQP engine (off the shelf)
Query Synopsis

Break down query into “snippets”

- Snippet – supported SQL query with a scalar result
- Single aggregate function
Break down query into “snippets”

- Snippet – supported SQL query with a scalar result
- Single aggregate function

### 2.2 Supported Queries

Verdict supports aggregate queries that are flat (i.e., no derived tables or sub-queries) with the following conditions:

1. **Aggregates.** Any number of SUM, COUNT, or AVG aggregates can appear in the select clause. The arguments to these aggregates can also be a derived attribute.

2. **Joins.** Verdict supports foreign-key joins between a fact table and any number of dimension tables, exploiting the fact that this type of join does not introduce a sampling bias [3]. For simplicity, our discussion in this paper is based on a denormalized table.

3. **Selections.** Verdict currently supports equality and inequality comparisons for categorical and numeric attributes (including the in operator). Currently, Verdict does not support disjunctions and textual filters (e.g., like ‘%Apple%’) in the where clause.

4. **Grouping.** Groupby clauses are supported for both stored and derived attributes. The query may also include a having clause. Note that the underlying AQP engine may affect the cardinality of the result set depending on the having clause (e.g., subset/superset error). Verdict simply operates on the result set returned by the AQP engine.
Inference Model - Problem setup

- Query snippet $q_i$ – AQP returns $(\theta_i, \beta_i)$

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_i$</td>
<td>$i$-th (supported) query snippet</td>
</tr>
<tr>
<td>$n + 1$</td>
<td>index number for a new snippet</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>random variable representing our knowledge of the raw answer to $q_i$</td>
</tr>
<tr>
<td>$\hat{\theta}_i$</td>
<td>(actual) raw answer computed by AQP engine for $q_i$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>expected error associated with $\theta_i$</td>
</tr>
<tr>
<td>$\hat{\theta}_i$</td>
<td>random variable representing our knowledge of the exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>exact answer to $q_i$</td>
</tr>
<tr>
<td>$\hat{\theta}_{n+1}$</td>
<td>improved answer to the new snippet</td>
</tr>
<tr>
<td>$\tilde{\beta}_{n+1}$</td>
<td>improved error to the new snippet</td>
</tr>
</tbody>
</table>
Inference Model - Problem setup

- Query snippet $q_i$ – AQP returns $(\theta_i, \beta_i)$
- $Q_n$ (synopsis) – AQP answers and errors for $n$ snippets

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_i$</td>
<td>$i$-th (supported) query snippet</td>
</tr>
<tr>
<td>$n + 1$</td>
<td>index number for a new snippet</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>random variable representing our knowledge of the raw answer to $q_i$</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>(actual) raw answer computed by AQP engine for $q_i$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>expected error associated with $\theta_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>random variable representing our knowledge of the exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_{n+1}$</td>
<td>improved answer to the new snippet</td>
</tr>
<tr>
<td>$\tilde{\beta}_{n+1}$</td>
<td>improved error to the new snippet</td>
</tr>
</tbody>
</table>
Inference Model - Problem setup

- Query snippet $q_i$ – AQP returns $(\theta_i, \beta_i)$
- $Q_n$ (synopsis) – AQP answers and errors for $n$ snippets
- Assumption: same aggregate and attribute e.g. $\text{AVG}(A_k)$

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_i$</td>
<td>$i$-th (supported) query snippet</td>
</tr>
<tr>
<td>$n + 1$</td>
<td>index number for a new snippet</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>random variable representing our knowledge of the raw answer to $q_i$</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>(actual) raw answer computed by AQP engine for $q_i$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>expected error associated with $\theta_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>random variable representing our knowledge of the exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\beta}_{n+1}$</td>
<td>improved answer to the new snippet</td>
</tr>
<tr>
<td>$\hat{\beta}_{n+1}$</td>
<td>improved error to the new snippet</td>
</tr>
</tbody>
</table>
Inference Model - Problem setup

- Query snippet \( q_i \) – AQP returns \((\theta_i, \beta_i)\)

- \( Q^n \) (synopsis) – AQP answers and errors for \( n \) snippets

- Assumption: same aggregate and attribute e.g. \( \text{AVG}(A_k) \)

- Problem: Given \( Q^n \) and \((\theta_{n+1}, \beta_{n+1})\) – compute improved \( q_{n+1} \) answer for \( q_{n+1} \)
Inference Model - Problem setup

- Represent query snippet answers as a random vars
- \( n + 1 \) random vars; 1 for each snippet in synopsis and one for current snippet: \( \theta_1, \theta_2, \ldots, \theta_{n+1} \)

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_i )</td>
<td>( i )-th (supported) query snippet</td>
</tr>
<tr>
<td>( n + 1 )</td>
<td>index number for a new snippet</td>
</tr>
<tr>
<td>( \theta_i )</td>
<td>random variable representing our knowledge of the raw answer to ( q_i )</td>
</tr>
<tr>
<td>( \theta_i )</td>
<td>actual) raw answer computed by AQP engine for ( q_i )</td>
</tr>
<tr>
<td>( \beta_i )</td>
<td>expected error associated with ( \theta_i )</td>
</tr>
<tr>
<td>( \tilde{\theta}_i )</td>
<td>random variable representing our knowledge of the exact answer to ( q_i )</td>
</tr>
<tr>
<td>( \tilde{\theta}_i )</td>
<td>exact answer to ( q_i )</td>
</tr>
<tr>
<td>( \tilde{\beta}_{n+1} )</td>
<td>improved answer to the new snippet</td>
</tr>
<tr>
<td>( \tilde{\beta}_{n+1} )</td>
<td>improved error to the new snippet</td>
</tr>
</tbody>
</table>
Inference Model - Problem setup

- Represent query snippet answers as a random vars

- $n + 1$ random vars; 1 for each snippet in synopsis and one for current snippet: $\theta_1, \theta_2, \ldots, \theta_{n+1}$

- These vars form a joint PDF

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_i$</td>
<td>$i$-th (supported) query snippet</td>
</tr>
<tr>
<td>$n + 1$</td>
<td>index number for a new snippet</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>random variable representing our knowledge of the raw answer to $q_i$</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>(actual) raw answer computed by AQP engine for $q_i$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>expected error associated with $\theta_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>random variable representing our knowledge of the exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\theta}_i$</td>
<td>exact answer to $q_i$</td>
</tr>
<tr>
<td>$\tilde{\beta}_{n+1}$</td>
<td>improved answer to the new snippet</td>
</tr>
<tr>
<td>$\tilde{\beta}_{n+1}$</td>
<td>improved error to the new snippet</td>
</tr>
</tbody>
</table>
If we have \( f(\theta_1 = \theta'_1, ..., \theta_{n+1} = \theta'_{n+1}, \bar{\theta}_{n+1} = \bar{\theta}'_{n+1}) \) then the prediction is the value of \( \bar{\theta}'_{n+1} \) that maximizes
\[
f(\bar{\theta}_{n+1} = \bar{\theta}'_{n+1} \mid \theta_1 = \theta_1, ..., \theta_{n+1} = \theta_{n+1})
\]

\[
\bar{\theta}_{n+1} = \operatorname{ArgMax}_{\bar{\theta}'_{n+1}} f(\bar{\theta}'_{n+1} \mid \theta_1 = \theta_1, ..., \theta_{n+1} = \theta_{n+1})
\]
Inference Model

- Joint PDF derived from answer statistics – means, variances, and covariances

- ME principle: Joint PDF will maximise entropy

- Constraint: satisfy means, variances, and covariances (derived from observations)

\[
h(f) = - \int f(\theta) \cdot \log f(\theta) \, d\theta
\]

\(\theta\)s are the observations (query answers)
Inference Model

- Leap: joint PDF that maximises entropy while satisfying means, variances, and covariances is a **normal distribution** with same statistics

- So if we have statistics, we can get the distribution!

\[
h(f) = -\int f(\vec{\theta}) \cdot \log f(\vec{\theta}) \, d\vec{\theta}
\]

\[
f(\vec{\theta}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(\frac{-1}{2} (\vec{\theta} - \vec{\mu})^\top \Sigma^{-1} (\vec{\theta} - \vec{\mu})\right)
\]

\(\theta_s\) are the observations (query answers)
Inference Model

- For a normal pdf, the value that maximises it is the mean – which would be the new “improved” answer

- Similarly, the variance is the error bound

\[
\begin{align*}
\mu_c &= \tilde{\mu}_{n+1} + \tilde{k}_{n+1}^\top \Sigma_{n+1}^{-1} (\tilde{\theta}_{n+1} - \tilde{\mu}_{n+1}) \\
\sigma_c^2 &= \tilde{k}_{n+1}^2 - \tilde{k}_{n+1}^\top \Sigma_{n+1}^{-1} \tilde{k}_{n+1}
\end{align*}
\]

where:
- $\tilde{k}_{n+1}$ is a column vector of length $n + 1$ whose $i$-th element is $(i, n + 2)$-th entry of $\Sigma$;
- $\Sigma_{n+1}$ is a $(n + 1) \times (n + 1)$ submatrix of $\Sigma$ consisting of $\Sigma$’s first $n + 1$ rows and columns;
- $\tilde{\theta}_{n+1} = (\theta_1, \ldots, \theta_{n+1})^\top$;
- $\tilde{\mu}_{n+1} = (\mu_1, \ldots, \mu_{n+1})^\top$; and
- $\tilde{k}^2$ is the $(n + 2, n + 2)$-th entry of $\Sigma$
Inference Model - Computing Stats

- Goal: compute means, variances, and covariances of the random vars $\theta_1, \ldots, \theta_{n+1}$
- Means: historical average
- Covariance?
Inference Model - Computing Stats

- Covariance \((\theta_i, \theta_j)\)?
- Can be decomposed down to computing inter-tuple covariance

As a result, the covariance between query answers can be broken into an integration of the covariances between tuple-level function values, which we call *inter-tuple covariances.*

\[
\text{cov}(\tilde{\theta}_i, \tilde{\theta}_j) = \text{cov}\left(\int_{t \in F_i} v_g(t) \, dt, \int_{t' \in F_j} v_g(t') \, dt'\right) \\
= \int_{t \in F_i} \int_{t' \in F_j} \text{cov}(v_g(t), v_g(t')) \, dt \, dt'
\]
Inference Model - Validation

- Negative Estimates for FREQ(\(\ast\)) – discard
- Unlikely improved answer
  - If not in a certain range of AQP answer, discard

Formally, let \(t \geq 0\) be the value for which the AQP engine’s answer would fall within \((\hat{\theta}_{n+1} - t, \hat{\theta}_{n+1} + t)\) with probability \(\delta_v\) (0.99 by default) if \(\hat{\theta}_{n+1}\) were the exact answer. We call the \((\hat{\theta}_{n+1} - t, \hat{\theta}_{n+1} + t)\) range the likely region. To compute \(t\), we must find the value closest to \(\hat{\theta}_{n+1}\) that satisfies the following expression:

\[
\text{Pr}\left(|X - \hat{\theta}_{n+1}| < t\right) \geq \delta_v
\]

(14)
Deployment scenarios

- **Accuracy bounded**
  
  Keep processing till error within certain bound

- **Time bounded**
  
  Give the best answer in limited time
Experiment 1 setup – Workloads

Customer1

- 310 tables, 15.5K queries – 3.3K analytical supported by Spark SQL
- Only access to data distribution – used to generate a 536 GB dataset
Experiment 1 setup – Workloads

310 tables, 15,500 queries – 3,300 analytical supported by Spark SQL

Only access to data distribution – used to generate a 536 GB dataset

Customer 1

These are only some of the companies who have successfully used VerdictDB. Contact us if you wish to find more.

VerdictDB

Locally

Making data analytics decisions based on its ability to understand data operations. Unfortunately, due to its scale of operations, VerdictDB is not the only solution.

Database Learning: Toward a Database that Becomes Smarter Every Time
Park et al., SIGMOD 2017

Walmart
Experiment 1 setup – Workloads

TPC-H

- Scale factor 100 – 100 GB
- 21 templates have aggregations
- 500 queries generated
Table 3: Generality of Verdict. Verdict supports a large fraction of real-world and benchmark queries.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total # of Queries with Aggregates</th>
<th># of Supported Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer1</td>
<td>3,342</td>
<td>2,463</td>
<td>73.7%</td>
</tr>
<tr>
<td>TPC-H</td>
<td>21</td>
<td>14</td>
<td>63.6%</td>
</tr>
</tbody>
</table>
Experiment 1 setup – Implementations

NoLearn

- Online aggregation engine
- Randomly samples and splits data
- Error bound improves as more batches get processed
Experiment 1 setup – Implementations

Verdict

- Uses NoLearn as AQP engine
- Takes NoLearn’s answer and improves
Experiment 1 setup – Two Flavours

- Cached – All samples exist in memory
- Not cached – Samples read from SSD
Experiment 1 – Key takeaways

- Verdict produced smaller errors even when runtime was very large
- Verdict showed faster runtime for the same target errors
Experiment 1 – Results (Customer1)

Database Learning: Toward a Database that Becomes Smarter Every Time
Park et al., SIGMOD 2017
Experiment 1 – Results (TCP-H)

(c) Cached, TPC-H
(d) Not Cached, TPC-H

Park et al., SIGMOD 2017
## Experiment 1 – Results summary

<table>
<thead>
<tr>
<th></th>
<th>Cached?</th>
<th>Error Bound</th>
<th>Time Taken</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NoLearn</td>
<td>Verdict</td>
</tr>
<tr>
<td>Customer 1</td>
<td>Yes</td>
<td>2.5%</td>
<td>4.34 sec</td>
<td>0.57 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0%</td>
<td>6.02 sec</td>
<td>2.45 sec</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>2.5%</td>
<td>140 sec</td>
<td>6.1 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0%</td>
<td>211 sec</td>
<td>37 sec</td>
</tr>
<tr>
<td>TPC-H</td>
<td>Yes</td>
<td>4.0%</td>
<td>26.7 sec</td>
<td>2.9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0%</td>
<td>34.2 sec</td>
<td>12.9 sec</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>4.0%</td>
<td>456 sec</td>
<td>72 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0%</td>
<td>524 sec</td>
<td>265 sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cache?</th>
<th>Runtime</th>
<th>Achieved Error Bound</th>
<th>Error Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NoLearn</td>
<td>Verdict</td>
</tr>
<tr>
<td>Customer 1</td>
<td>Yes</td>
<td>1.0 sec</td>
<td>21.0%</td>
<td>2.06%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0 sec</td>
<td>1.98%</td>
<td>0.48%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>10 sec</td>
<td>21.0%</td>
<td>2.06%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 sec</td>
<td>6.55%</td>
<td>0.87%</td>
</tr>
<tr>
<td>TPC-H</td>
<td>Yes</td>
<td>5.0 sec</td>
<td>13.5%</td>
<td>2.13%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30 sec</td>
<td>4.87%</td>
<td>1.04%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>3.0 min</td>
<td>11.8%</td>
<td>1.74%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 min</td>
<td>4.49%</td>
<td>0.92%</td>
</tr>
</tbody>
</table>
Experiment 2 setup

- RQ: Impact of having queries with diverse set of columns in selection predicates
Experiment 2 setup

- RQ: Impact of having queries with diverse set of columns in selection predicates
- Synthetic dataset – 50 columns (10% categorical), 5M rows
Experiment 2 setup

- RQ: Impact of having queries with diverse set of columns in selection predicates
- Synthetic dataset – 50 columns (10% categorical), 5M rows
- 4 synthetic workloads – varying proportions of frequently accessed columns
Experiment 2 results

- As % of frequently accessed columns increases, Verdict’s relative error reduction over NoLearn gradually decreases.

- Expected – Verdict constructs its model based on historical data.
Experiment 3 setup

- RQ: Verdict’s sensitivity to underlying data distribution
- Synthetic dataset – tables generated using uniform, Gaussian, and log-normal
Experiment 3 results

- Performance remains roughly consistent
- Expected – No assumptions were made about the underlying data.
- Maximum Entropy Principle works for all
Experiment 4 setup

- RQ: Change in Verdict’s performance w.r.t. past queries
Experiment 4 results

- Error bound improves as the number of queries; eventually tapers off
- Takeaway – Small number of queries are required to deliver a good performance

![Error Reduction vs # of past queries graph]
Experiment 4 results

- Error bound improves as the number of queries; eventually tapers off

- Takeaway – Small number of queries are required to deliver a good performance

- A little sketchy – this would be highly data dependent
Some questions

Verdict rejects the improved answer when it is not in the 99% interval of the AQP answer

- How often does this happen?
- Do the graphs include these queries, or is it plotted from cherry picked results?