Provenance, Relevance-based Data Management, and the Value of Data
ProvenanceWeek ’23

Boris Glavic

DBGroup
Illinois Institute of Technology

2023-04-30
This Keynote’s Provenance

Have to follow the laws of keynotes

Keynote Checklist

1. Overly grandiose vision
2. Repackage existing work under fancy new name
3. Shamelessly plug your own work
4. Rant about the state of the field
This Keynote’s Provenance

Have to follow the laws of keynotes

Keynote Checklist

1. **Overly grandiose vision**
2. Repackage existing work under fancy new name
3. Shamelessly plug your own work
4. Rant about the state of the field
What is relevance?

- $D_1$
- $D_2$
- $D_3$
- $Q$
- $D_{out}$
What is relevance?

Relevant inputs

D_1

D_2

D_3

Q

D_{out}
Running Example

Revenue of states with > 10000 sales from category toys or food

SELECT state, sum(sales) AS revenue
FROM prod-sales WHERE prod-category IN ('toys','food')
GROUP BY state HAVING sum(sales) > 10000
What data is relevant for this query?

- only rows where `prod_category` IN (toys, food)
- only states where `sum(sales)` > 10000 for these categories

SELECT `state`, `sum(sales)` AS revenue 
FROM `prod-sales` WHERE `prod_category` IN ('toys','food')
GROUP BY `state` HAVING `sum(sales)` > 10000

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>prod-category</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>4000</td>
<td>toys</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>5000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>6000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>2500</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2600</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>1400</td>
<td>alcohol</td>
</tr>
</tbody>
</table>
### Speed-up computations
- Restrict computations to relevant data
- How to reuse relevance information across similar computations?

### Allocating wrangling resources
- Data collection and preparation is expensive
- Why spend effort on data that is not relevant?

### Data life-cycle management
- Relevant data should be kept in fast storage (cache)
- Irrelevant data can be compressed / deleted / moved to slow storage

### Objective metric of data value
- Relevance is an **objective metric of data value**
- **Dark data**: data that is not useful or has not yet been discovered to be useful
Defining Data Relevance

**Definition (Relevance)**

- **Input**: dataset $D$ and computation $Q$
- **Output**: $R(Q, D) = D' \subseteq D$ fulfilling:
  - **sufficiency**: $Q(D') = Q(D)$
  - **minimality**: $\forall D'' \subset D' : Q(D'') \neq Q(D)$

**Relevance vs. Data Access**

- $D$ was accessed $\not\Rightarrow D$ is relevant

**Aggregated Relevance**

- Workload $\mathcal{W} = Q_1, \ldots, Q_n$
- Relevance of $d_i \in D$ for $\mathcal{W}$ is:
  $$\sum_i \frac{1[d_i \in R(Q_i, D)]}{|\mathcal{W}|}$$
Remarks

Problems with minimality

- **not unique** (e.g., disjunctive operations)!
  - for some applications we don’t care
  - otherwise,

\[ \mathcal{R}(Q, D) = \{ d \mid \exists D' \subset D : d \in D' \wedge D' \text{ is sufficient and minimal} \} \]

- **computational complexity**
  - **NP-hard** for \( \text{sum} \Rightarrow \text{subset-sum problem} \)
  - drop minimality requirement (and/or sufficiency)
## Degrees of relevance

- relevance is all-or-nothing
- some applications (e.g., approximation) need degree of relevance

## Data- + computation-centric

- relevance is data-centric
  - specific to $D$
- relevance is computation-centric
  - specific to $Q$ or $\mathcal{W}$
<table>
<thead>
<tr>
<th>Keynote Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Overly grandiose vision</td>
</tr>
<tr>
<td>2. Repackage existing work under fancy new name</td>
</tr>
<tr>
<td>3. Shamelessly plug your own work</td>
</tr>
<tr>
<td>4. Rant about the state of the field</td>
</tr>
</tbody>
</table>
Consider a tuple level provenance model $P(Q, D, t)$
- most provenance models are sufficient (e.g., [GKT07, Gla21])
- some fulfill our fixed minimality notion

For such models:

$$\mathcal{R}(Q, D) = \bigcup_{t \in Q(D)} P(Q, D, t)$$
Relevance vs. Attribution

Degrees of relevance = attribution / responsibility?

- If approximate answers are acceptable, then we want more information
  - how much does a data item contribute to a result
  - e.g., bias sampling in APQ towards more relevant data
- closely related to:
  - causal responsibility [MGMS10]
  - attribution techniques (e.g., Shapley [LBKS20, GZ19, dBLSS21])
Isn’t that just ...?

### Provenance
- **yes, but ...**
  - only care about sufficiency not completeness for many relevance use cases
  - novel challenges
    - reusing provenance across computations
    - maintaining provenance under updates
    - (over-)approximations

### Heatmaps
- **yes, but ...**
  - heatmaps track data access, not relevance

### Materialized views
- **no, but ...**
  - store relevant inputs instead of outputs
Keynote Checklist

1. Overly grandiose vision
2. Repackage existing work under fancy new name
3. Shamelessly plug your own work
4. Rant about the state of the field
System-centric Provenance Usage

The obligatory keynote rant

- Most provenance work has focused on end user consumption
  - debugging,
  - auditing
  - understanding computations
  - ...

Slide 16 of 79 Boris Glavic - Relevance-based Data Management
Provenance as a supporting technology

- partial result refresh [ISW11]
- reuse [PW18]
- provisioning [AKLT15]
- probabilistic databases [VdBS17]
- reproducibility [MSM⁺22, KGFB22]
- ...
Why is System-centric Provenance Different?

We can now ignore
- visualization of provenance
- can a human comprehend the provenance we create?

Facing objective truths
- optimizing against objective metrics
  - speed
  - storage consumptions
  - accuracy
  - ...
This Keynote’s Provenance

Keynote Checklist

1. Overly grandiose vision
2. Repackage existing work under fancy new name
3. Shamelessly plug your own work
4. Rant about the state of the field
Motivation

Fundamental principle in query processing

- determine upfront (statically) what data is needed (relevant) to answer a query
- exclude irrelevant data as early and as efficiently as possible
How to Filter Irrelevant Data?

- Decades of research & development - efficient storage organization for databases
  - **Index structures** [LLS13, AS13, Gra06, Com79, BS77, Moe98]
    - B-trees, Hash-index, Bitmap-index, zone maps, ...
    - select rows based on attribute values
  - **Partitioned tables** [DGTM17, RJ17, SFWW16, AEHS+14, TMS+14, PCZ12, ANY04, CY90]
    - split tables into horizontal/vertical fragments
  - ...

- Decades of research & development - query optimization and execution
  - **Selection move-around** [LM97]
    - filter data based on selections implied by the query
  - **Semi-join reducers** [Mul90, BC81, BG81]
    - filter input tables before joining
  - ...

Running Example

Revenue of states with > 10000 sales from category toys or food

<table>
<thead>
<tr>
<th>state</th>
<th>revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>11,100</td>
</tr>
</tbody>
</table>

SELECT state, sum(sales) AS revenue
FROM prod-sales WHERE prod-category IN ('toys','food')
GROUP BY state HAVING sum(sales) > 10000

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>prod-category</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>4000</td>
<td>toys</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>5000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>6000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>2500</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2600</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>1400</td>
<td>alcohol</td>
</tr>
</tbody>
</table>
**Running Example (cont.)**

<table>
<thead>
<tr>
<th>What data is relevant for this query?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• only rows where prod_category IN (toys, food)</td>
</tr>
<tr>
<td>• only states where sum(sales) &gt; 10000 for these categories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What data can be skipped by the DBMS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Utilize zone map (index, partitioning, ...) on prod_category</td>
</tr>
<tr>
<td>• however, we cannot filter states because we will only know at runtime which states qualify!</td>
</tr>
</tbody>
</table>

```sql
SELECT state, sum(sales) AS revenue
FROM prod-sales WHERE prod-category IN ('toys', 'food')
GROUP BY state HAVING sum(sales) > 10000
```
### Runtime Relevance Analysis

- Determine **dynamically** at runtime what data is **relevant** for a query
  - pay overhead once to determine what data is relevant
- Use this information to **benefit future queries**
  - **amortize** cost over time

### Synergy with existing technologies

- To use relevance information to efficiently filter data we need to 
  exploit . . .
  - zone maps
  - indexes
  - partitioning
  - in-memory caches
  - . . .
Filtering relevant data

- only CA qualifies, add condition `state IN ('CA')`

```sql
SELECT state, sum(sales) AS revenue
FROM prod-sales
WHERE prod-category IN ('toys', 'food')
    AND state IN ('CA')
GROUP BY state HAVING sum(sales) > 10000
```

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>prod-category</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>4000</td>
<td>toys</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>5000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>6000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>2500</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2600</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>1400</td>
<td>alcohol</td>
</tr>
</tbody>
</table>
(1) Capture
- How to compactly over-approximate what data is relevant?
  - just listing relevant rows is inefficient
- How to determine efficiently at runtime what data is relevant?
  - needs to be applicable to relatively complex queries

(2) Use
- How to filter irrelevant data effectively?
  - need to encode relevant data for efficient filtering
  - want synergy with existing physical design and self-tuning technologies

(3) Safety
- How to determine whether an over-approximation of relevant data yields the same result?
(4) Reuse
- How to utilize relevance information for multiple queries?
  - need to determine statically whether the relevant data for query $Q_1$ subsumes relevant data of $Q_2$

(5) Maintenance
- How to maintain relevance information under updates?
  - when data is updated, relevance information becomes stale
When is this beneficial?

Fundamental requirement

- We have |relevant data| « |data that DBMS can filter|

Counterexample

- If in our running example CA is the only state that sells food and toys, then the data returned by the DBMS based on the query’s WHERE clause is exactly what is relevant for the query.
- \( \Rightarrow \) there is no way for us to benefit

Common query types with these characteristics

- HAVING queries
- Top-k queries
1. Compact over-approximation of relevant data
   - Use *(virtual)* range-partitioning to represent subsets of the data
   - **Provenance sketches**: stores which fragments of a partition contain relevant data

2. Determine at runtime what data is relevant for a query
   - Our method utilizes data provenance techniques
   - Instrument queries to capture provenance sketches *(query rewrites)*

3. Instrument queries to filter out irrelevant data early-on
   - Use provenance sketches to speed-up execution of a query
   - **Filter data** based on sketch by adding WHERE conditions

4. Re-use of provenance sketches
   - Use sketch captured for query $Q_1$ to answer $Q_2$
   - Check statically if $Q_1$’s sketch subsumes relevant data for $Q_2$
Revisiting the Running Example

Range partitioning

- We partition on sales
  
  \[
  \{[0, 600], [601, 1200], \\
  [1201, 2700], [2701, 3500], \\
  [3501, 5500], [5501, 10000]\}
  \]

Provenance Sketch

- Fragments in the sketch (that belong to provenance)
  \[
  \{[1201, 2700], [5501, 10000]\}
  \]

Captured data

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>prod-category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>San Francisco</td>
<td>6000</td>
<td>toys</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>2500</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2600</td>
<td>food</td>
</tr>
<tr>
<td>CA</td>
<td>San Francisco</td>
<td>1400</td>
<td>alcohol</td>
</tr>
</tbody>
</table>
Data structures
- Maintain index of sketches that maps parameterized queries + bind parameters to sketches
- Bookkeep costs and benefits

Workflow for a query
- Can use existing sketch
- Should create new sketch for query?
  - YES: Instrument query to use sketch
  - NO: Which attributes to use
    - YES: Instrument query to create sketch
      - NO: Execute query
Outline

1. Relevance-based Data Management
2. Provenance-based Data Skipping
3. Provenance Sketches
4. Sketch Capture and Use
5. Reusing Sketches and Sketch Safety
6. Experiments
7. Relevance and The Value of Data
8. Conclusions
Overview

What do we want?

1. compact over-approximation of provenance
2. fast to compute
3. can be exploited by the DBMS to skip data

Horizontal Partitioning

Horizontal partitioning of a table $R = \text{set of fragments } f_1, \ldots, f_n$ such that:

1. $\forall i : f_i \subseteq R$
2. $\bigcup_{i=1}^{n} f_i = R$
3. $\forall i, j : f_i \cap f_j = \emptyset$
A **provenance sketch** \( \mathcal{P} \) for a query \( Q \) and database \( D \) contains for each table \( R \) in \( D \) accessed by \( Q \) a pair:

- a *horizontal partitioning* \( \mathcal{F}_R \) for \( R \)
- a set of fragments: \( \mathcal{P}(R) \subseteq \mathcal{F}_R \)

Data contained in a sketch:

\[
D_\mathcal{P} = \bigcup_{f \in \mathcal{P}} f
\]

- over-approximates provenance:

\[
\text{Prov}(Q, D) \subseteq D_\mathcal{P}
\]
A provenance $\mathcal{P}$ for a query $Q$ and database $D$ is accurate iff:

$$\forall f \in \mathcal{P} : \text{Prov}(Q, D) \cap f \neq \emptyset$$
**Provenance Sketches (Example)**

```sql
SELECT state, sum(sales) AS rev
FROM sal
GROUP BY state
HAVING sum(sales) > 5000
```

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td></td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>3000</td>
<td>$P_1$</td>
</tr>
<tr>
<td>IL</td>
<td>Schaumburg</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2000</td>
<td>$P_2$</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>San Francisco</td>
<td>8000</td>
<td>$P_3$</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

**Provenance Sketches**

- **Page-based**
  \{ $P_2$, $P_3$ \}

- **Hash-based (on sales)**
  \{ 1, 2 \}

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>8000</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Range-based (on sales)**
  \{ [801, 5000], [5001, 100000] \}

For ranges:
  \{ [0, 500], [501, 800], [801, 5000], [5001, 20000] \}
1. Any partitioning can be utilized to create a provenance sketch
   - Sketch contains all fragments that contains provenance
   - e.g., range - divide the input table into fragments based on a partitioning of an attribute domain (can exploit histograms [CGHJ12, Ioa03])

2. Physical vs. Virtual
   - **Physical** = partitioning aligns with physical design
   - **Virtual** = partitioning does not align with physical design
Outline

1. Relevance-based Data Management
2. Provenance-based Data Skipping
3. Provenance Sketches
4. Sketch Capture and Use
5. Reusing Sketches and Sketch Safety
6. Experiments
7. Relevance and The Value of Data
8. Conclusions
Query Instrumentation

- Given query $Q$
- Construct query $Q_{sketch}$ that generates a provenance sketch

Relational Algebra Rewrite Rules

- One rewrite rule per algebra operator
  - $\Rightarrow$ composable
- Queries are rewritten by recursive application of these rules

Extension of our previous work on query instrumentation [AFG+18]

- Represent sets of fragments as bitvectors
- Combine used user-defined aggregation functions
Input Query

```sql
SELECT dept, avg(salary) AS avgsal
FROM emp
GROUP BY dept
```

Instrumented Query

```sql
SELECT dept, avgsal, provName, provSalary, provDept
FROM (SELECT dept, avg(salary) AS avgsal
    FROM emp
    GROUP BY dept) o,
    (SELECT name as provName,
       salary AS provSalary,
       dept AS provDept
    FROM emp) p
WHERE o.dept = p.provDept
```
Using the instrumented query from the previous slide

<table>
<thead>
<tr>
<th>dept</th>
<th>avgsal</th>
<th>provName</th>
<th>provSalary</th>
<th>provDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>15</td>
<td>Peter</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>HR</td>
<td>15</td>
<td>Bob</td>
<td>20</td>
<td>HR</td>
</tr>
<tr>
<td>IT</td>
<td>5</td>
<td>Alice</td>
<td>5</td>
<td>IT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>salary</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>HR</td>
</tr>
<tr>
<td>Alice</td>
<td>5</td>
<td>IT</td>
</tr>
</tbody>
</table>
Input Query

```sql
SELECT dept, avg(salary) AS avgsal
FROM emp
GROUP BY dept
```

Instrumented Query

for ranges: \{[0, 500], [501, 800], [801, 5000], [5001, 20000]\}

```sql
SELECT bit_or_agg(provsketch)
FROM (SELECT salary, dept,
    CASE WHEN salary BETWEEN 0 AND 500 THEN 1 << 0
    WHEN salary BETWEEN 0 AND 500 THEN 1 << 1
    ... END AS provsketch
    FROM empl)
GROUP BY dept
```
To benefit from using a provenance sketch we have to instruct the database what data to access.

Most sketch types: compile sketch into selection condition.

Database systems are good at filtering data based on conditions!

We compile the compact representation of what is relevant provided by the provenance sketch into a format understood by the DBMS!
Provenance Sketch Use (Example)

SELECT state, sum(sales) AS rev
FROM sal
WHERE salary BETWEEN 801 AND 5000
  OR salary BETWEEN 5001 AND 100000
GROUP BY state
HAVING sum(sales) > 5000

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>3000</td>
<td>$P_1$</td>
</tr>
<tr>
<td>IL</td>
<td>Schaumburg</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2000</td>
<td>$P_2$</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>San Francisco</td>
<td>8000</td>
<td>$P_3$</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Provenance Sketches
- **Range-based** (on sales)
  - $[[801, 5000], [5001, 100000]]$

  for ranges:
  - $[[0, 500], [501, 800], [801, 5000], [5001, 20000]]$
**Instrumented query**

```sql
SELECT state, sum(sales) AS revenue
FROM prod-sales
WHERE prod-category IN ('toys', 'food')
    AND (salary BETWEEN 1201 AND 2700
        OR salary BETWEEN 5500 AND 10000)
GROUP BY state HAVING sum(sales) > 10000
```

**Sketch**

- **Partition** sales
  - `[[0, 600], [601, 1200],
    [1201, 2700], [2701, 3500],
    [3501, 5500], [5501, 10000]]`
- **Sketch**
  - `[[1201, 2700], [5501, 10000]]`

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
<th>prod-category</th>
<th>fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>4000</td>
<td>toys</td>
<td>[3501, 5500]</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>5000</td>
<td>toys</td>
<td>[3501, 5500]</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>6000</td>
<td>toys</td>
<td>[5501, 10000]</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>2500</td>
<td>food</td>
<td>[1201, 2700]</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2600</td>
<td>food</td>
<td>[1201, 2700]</td>
</tr>
<tr>
<td>CA</td>
<td>San Fransico</td>
<td>1400</td>
<td>alcohol</td>
<td>[1201, 2700]</td>
</tr>
</tbody>
</table>
Integrating everything

Problem Definition

- A database $D$
- A workload of queries $Q_1, \ldots, Q_n$ which is unveiled one query at a time
- Decide for each query ... 
  - what (if any) relevance information to capture
  - whether to reuse previously captured relevance information to speed-up the query

Bookkeeping

- track what queries we have seen so far
- store provenance sketch(es) for queries
- fetch sketch if needed
Project page
Available on github
- https://github.com/IITDBGroup/gprom
GProM [?] one sentence:
- A SQL+X to SQL optimizing compiler
Implemented in C
- Clients: CLI, shared library, Java bindings
- Frontend languages: SQL, Datalog
- Backends (use native C libraries or ODBC): Oracle, Postgres, SQLite, MonetDB, MSSQL
<table>
<thead>
<tr>
<th>GProM</th>
<th>SQL</th>
<th>Datalog</th>
<th>Java Bindings</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>JDBC</td>
<td>libgprom</td>
<td></td>
</tr>
<tr>
<td>Provenance Rewriter</td>
<td>Transaction Reenactor</td>
<td>Optimizer</td>
<td>Frontend</td>
</tr>
<tr>
<td>Oracle</td>
<td>Postgres</td>
<td>SQLite</td>
<td>LogicBlox</td>
</tr>
<tr>
<td>Oracle</td>
<td>Postgres</td>
<td>SQLite</td>
<td>MonetDB</td>
</tr>
</tbody>
</table>

Q(X) :- R(X,Y).
WHY(Q(Peter))

GProM Architecture

PROVENANCE OF (SELECT * FROM ...

Transaction Reenactor
Optimizer

Target Code Generation
Backend Connector

C client library

Oracle Postgres SQLite

Oracle Postgres SQLite MonetDB

CLI JDBC
Outline

1. Relevance-based Data Management
2. Provenance-based Data Skipping
3. Provenance Sketches
4. Sketch Capture and Use
5. Reusing Sketches and Sketch Safety
6. Experiments
7. Relevance and The Value of Data
8. Conclusions
### Sketch Reuse

- Given query $Q_1$ and its sketch $\mathcal{P}_1$
  - can it be reused to answer $Q_2$
- Prove that sketch for $Q_1$ contains all relevant data for $Q_2$
  - *Provenance containment* [Gre11]

### Results

- Limited to parameterized queries
- Sound static method utilizing database statistics
  - inspired by work for proving query equivalence ([ZAN+19])
Parameterized Queries

- Query whose selection conditions may contain placeholders
- **Instance** of a parameterized query:
  - Query we get for a given parameter setting

Example

```
SELECT sum(popden) AS totalPop, country
FROM city WHERE popden < $1
GROUP BY country
```

- $1 = 100000

```
SELECT sum(popden) AS totalPop, country
FROM city WHERE popden < 100000
GROUP BY country
```
Theorem (Provenance Containment)

Given a sketch $\mathcal{P}$ for an instance $Q_1$ a parameterized query $\mathcal{T}$ and a database $D$, $\mathcal{P}$ is sufficient for any instance $Q_2$ of $\mathcal{T}$ if\(^a\):

$$\text{Prov}(Q_1, D) \supseteq \text{Prov}(Q_2, D)$$

\(^a\)Under the assumption that (i) provenance is sufficient and (ii) sufficiency is preserved under $\subseteq$
Our sound condition

- Universally quantified formula based on per operator rules
- Encodes constraints that have to hold for (intermediate) result query results
  - for each $t$ of $Q_2$ we exists corresponding $t'$ in $Q_1$:
    - $\text{Prov}(Q_2, t) \subseteq \text{Prov}(Q_1, t')$
- Formula is valid $\Rightarrow$ provenance containment for every database $D$

\[^a\text{Can add statistics of } D \text{ as additional constraints.}\]

Testing the condition

- We use an SMT solver (Z3 [dMB08])
$\alpha_{\text{sum}(\text{popden}); \text{state}}$

$\sigma_{\text{popden}<5000}$

$cities$

$\alpha_{\text{sum}(\text{popden}'); \text{state}}$

$\sigma_{\text{popden}' < 3000}$

$cities$

$\text{popden} = \text{popden}' \land \text{popden}' < 3000 \rightarrow \text{popden} < 5000$
Problem

- Provenance sketches encode super sets of a query’s provenance
- This may lead to incorrect results for some queries
- How to determine which sketches are safe to use?

Definition (Sketch Safety)

We call a sketch $\mathcal{P}$ safe for a query $Q$ and database $D$ if

$$Q(D) = Q(D_{\mathcal{P}})$$
Example (Unsafe Sketch)

\[
\begin{array}{|c|c|}
\hline
\text{state} & \text{rev} \\
\hline
\text{IL} & 3510 \\
\hline
\end{array}
\]

```
SELECT state, sum(sales) AS rev 
FROM sal 
GROUP BY state 
HAVING sum(sales) < 5000
```

Provenance Sketches

- Range-based (on sales)

\{[0, 500], [501, 4000]\}

for ranges:

\{[0, 500], [501, 4000], [4001, 20000]\}
Example (Unsafe Sketch)

<table>
<thead>
<tr>
<th>state</th>
<th>rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>3510</td>
</tr>
<tr>
<td>CA</td>
<td>3000</td>
</tr>
</tbody>
</table>

```
SELECT state, sum(sales) AS rev
FROM sal
GROUP BY state
HAVING sum(sales) < 5000
```

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>3000</td>
</tr>
<tr>
<td>IL</td>
<td>Schaumburg</td>
<td>500</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2000</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>10</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>1000</td>
</tr>
</tbody>
</table>

Provenance Sketches

- Range-based (on sales)
  ```
  {{[0, 500], [501, 4000]}}
  ```
  for ranges:
  ```
  {{[0, 500], [501, 4000], [4001, 20000]}}
  ```
Safety is Data Dependent!

```
state  rev
IL     3510
```

```sql
SELECT state, sum(sales) AS rev
FROM sal
GROUP BY state
HAVING sum(sales) < 5000
```

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>Chicago</td>
<td>3000</td>
</tr>
<tr>
<td>IL</td>
<td>Schaumburg</td>
<td>500</td>
</tr>
<tr>
<td>CA</td>
<td>Sacramento</td>
<td>2000</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>10</td>
</tr>
<tr>
<td>CA</td>
<td>San Fansico</td>
<td>4000</td>
</tr>
<tr>
<td>CA</td>
<td>Santa Cruz</td>
<td>1000</td>
</tr>
</tbody>
</table>

Provenance Sketches

- Range-based (on sales)
  ```
  \[
  ([0, 500], [501, 4000])
  \]
  ```
  for ranges:
  ```
  ([0, 500], [501, 4000], [4001, 20000])
  ```
Requirements for Safety Checks

- Avoid generating incorrect sketches
  - ⇒ determine sketch safety at query compile time
  - ⇒ don’t have full access to the data or intermediate results

- Gather information about the data as long as this is cheap
  - access stats the DBMS keeps anyways

- Ensure Soundness
  - have to give up completeness
  - ⇒ we may fail to find safe sketches but never erroneously declare a sketch as safe
Definition (Monotone Queries)

A query $Q$ is **monotone** if for any pair of databases $D \subseteq D'$:

$$Q(D) \subseteq Q(D')$$

Theorem (Any Sketch is Safe for Monotone Queries)

- Let $Q$ be a monotone query and $D$ a database
- $\Rightarrow$ any provenance sketch for $Q$ and $D$ is safe

Proof.

- $Prov(Q, D) \subseteq D_P \subseteq D$ (provenance sketch definition)
- $Q(Prov(Q, D)) = Q(D)$ (sufficiency of provenance)
- $\Rightarrow Q(Prov(Q, D)) \subseteq Q(D_P) \subseteq Q(D)$ (monotonicity)
- $\Rightarrow Q(D_P) = Q(D)$
Definition (Column Safety)

A set of columns $X$ is **safe** for a query $Q$ if for all databases $D$, a sketch created on $X$ for any horizontal partitioning of tables on $X$ is safe.

Overview

- Define generalized containment to reason about relationships of tuples from $Q(D)$ and $Q(D_P)$
- Construct universally quantified condition for a query and set of attributes $X$ that implies safety of $X$
Outline

1. Relevance-based Data Management
2. Provenance-based Data Skipping
3. Provenance Sketches
4. Sketch Capture and Use
5. Reusing Sketches and Sketch Safety
6. Experiments
7. Relevance and The Value of Data
8. Conclusions
Experimental Setup

Workload and setup

- **Machine:**
  - 2 x AMD Opteron CPUs (12 cores)
  - 128 GB RAM
  - 4 x 1TB 7.2K HDs (RAID 5)

- **DBMS**
  - Postgres, MonetDB, System X\(^a\)

- **Workloads:**
  - TPC-H [Tra09]
  - Real-world datasets
    - movie ratings (movie lens)
    - crime (City of Chicago Data portal)
    - Stack overflow

\(^a\)Name redacted because of licensing restrictions
Experiments Overview

- Microbenchmarks
  - Overhead of capturing provenance sketches
  - Benefits of using provenance sketches
- End-to-end experiments
  - Execute a complete workload with and w/o PBDS
## Microbenchmarks

### Use Speed-up

<table>
<thead>
<tr>
<th></th>
<th>TPCH-1GB</th>
<th>TPCH-10GB</th>
<th>Crimes</th>
<th>Movies</th>
<th>StackOverflow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
<td>1.2</td>
<td>1.2</td>
<td>1.4</td>
<td>1.5</td>
<td>32.4</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td>5.5</td>
<td>13.3</td>
<td>5.1</td>
<td>2.6</td>
<td>46.0</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>21.3</td>
<td>33.1</td>
<td>8.7</td>
<td>3.6</td>
<td>66.3</td>
</tr>
</tbody>
</table>

### Capture Overhead

<table>
<thead>
<tr>
<th></th>
<th>TPCH-1GB</th>
<th>TPCH-10GB</th>
<th>Crimes</th>
<th>Movies</th>
<th>StackOverflow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
<td>0.01</td>
<td>0.08</td>
<td>2.27</td>
<td>1.91</td>
<td>-0.02</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td>0.35</td>
<td>0.38</td>
<td>2.68</td>
<td>0.90</td>
<td>0.56</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>0.90</td>
<td>1.06</td>
<td><strong>3.08</strong></td>
<td>2.66</td>
<td>1.20</td>
</tr>
</tbody>
</table>
**Setup**

- **Database:** Postgres
- **Machine:** 2 x AMD Opteron CPUs (12 cores), 128 GB RAM, 4 x 1TB 7.2K HDs (RAID 5)
- **Dataset:** Stackoverflow data (real data) (186 GB) (Brin indexes / Postgres Zonemaps)

**Capture & Use**

**Capture**

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Use**

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Relative overhead**

- PS1k
- PS10k

**Runtime (sec)**

- No-PS
- PS1k
- PS10k
End-to-end experiments

**Setup**

- Query workload consisting of a single template (200 instances)
- Compare DBMS (No-PS) against approach that creates and uses provenance sketches (adaptive)

**0.7% selectivity**

- Runtime (sec) vs. Number of Iterations

**2% selectivity**

- Runtime (sec) vs. Number of Iterations
Outline

1. Relevance-based Data Management
2. Provenance-based Data Skipping
3. Provenance Sketches
4. Sketch Capture and Use
5. Reusing Sketches and Sketch Safety
6. Experiments
7. Relevance and The Value of Data
8. Conclusions

References
What is the value of data

- Organizations access their data through computations (queries)
  - only what is returned by these computations matters
- Data that does not contribute to any result could be deleted without affecting the result of any computation
  - \( \Rightarrow \) irrelevant data has no value!
- **data relevance \( = \) data value**

What is needed?

- Aggregating relevance information across workloads
- Computing relevance for all queries is prohibitive
- Under-approximations could be ok
The Value of a Query

Query value
- Not all computations are equally important!

How to assess the value of a query?
- User-provided metrics
  - queries from management are more important than queries from HR
- Feedback mechanism
  - being accessed by a relevant query makes data relevant
  - accessing relevant data makes a query relevant
- Economic models
Dealing with Dark Data

Dark data
- data that has no or very little relevance
- is this data simply not useful or is its value unknown as of now?

Potential data value
- Some data has **potential value**
  - It could help the organization, but this has not been realized yet
- How to improve data management for dark data?
  - Explicit detection and controlled exploration of dark data
Outline

1 Relevance-based Data Management
2 Provenance-based Data Skipping
3 Provenance Sketches
4 Sketch Capture and Use
5 Reusing Sketches and Sketch Safety
6 Experiments
7 Relevance and The Value of Data
8 Conclusions
The Vision

Relevance for Data Management
- Make informed data management decisions based on relevance information
- A fundamentally data-centric approach

Relevance and Data Value
- Relevance = objective metric for the value of data
### Current Work (PBDS)

#### Maintaining Sketches under Updates
- Maintain sketches incrementally / approximately when data is updated

#### Integration with Query Optimization & Self-tuning
- Cost-based decisions for when to create / use what sketches

#### Reuse beyond parameterized queries
- Normalize query structure
- Generalize our sound condition

#### Measure data value using sketches
- Sketches approximate what data is actually needed to answer queries
- Combine with metric for value of queries towards an objective metric for data value
Open Questions (RBDM)

**Value of Queries**
- How to assess the value of queries for an organization

**Sketch reuse beyond parameterized queries**
- Generalize sound condition

**Sketches for semi-structured and graph data**
- Larger space of sketching methods

**Sketches for approximate queries**
- Allow both under- and over-approximations
Questions?

IITDBGroup

- **www**: http://www.cs.iit.edu/~dbgroup/
Lyublena Antova, Amr El-Helw, Mohamed A Soliman, Zhongxian Gu, Michalis Petropoulos, and Florian Waas.
Optimizing queries over partitioned tables in mpp systems.

Bahareh Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng.
GProM - A swiss army knife for your provenance needs.

Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen.
Algorithms for provisioning queries and analytics.

S. Agrawal, V. Narasayya, and B. Yang.
Integrating vertical and horizontal partitioning into automated physical database design.

Daniar Achakeev and Bernhard Seeger.
Efficient bulk updates on multiversion b-trees.

P.A. Bernstein and D. Chiu.
Using semijoins to solve relational queries.
JACM, 28(1), 1981.

Philip A Bernstein and Nathan Goodman.
Power of natural semijoins.

R. Bayer and M. Schkolnick.
Concurrency of operations on b-trees.

Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine.
Synopses for massive data: Samples, histograms, wavelets, sketches.
[Com79] D. Comer.
Ubiquitous b-tree.

An effective approach to vertical partitioning for physical design of relational databases.

On the tractability of SHAP explanations.


[DMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner.
Z3: an efficient SMT solver.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen.
Provenance Semirings.

Data provenance - origins, applications, algorithms, and models.
B-tree indexes for high update rates.  

Containment of conjunctive queries on annotated relations.  

Data shapley: Equitable valuation of data for machine learning.  

[Ioa03] Yannis Ioannidis.  
The history of histograms (abridged).  

Provenance-based refresh in data-oriented workflows.  

[KGFB22] Oliver Kennedy, Boris Glavic, Juliana Freire, and Mike Brachmann.  
The right tool for the job: Data-centric workflows in vizier.  

The shapley value of tuples in query answering.  

The bw-tree: A b-tree for new hardware platforms.  

Query optimization by predicate move-around, August 19 1997.

The Complexity of Causality and Responsibility for Query Answers and non-Answers.

Small Materialized Aggregates: A light weight index structure for data warehousing.

Chex: Multiversion replay with ordered checkpoints.

Optimal semijoins for distributed database systems.

[NLL+21] Xing Niu, Ziyu Liu, Pengyuan Li, Boris Glavic, Dieter Gawlick, Vasudha Krishnaswamy, Zhen Hua Liu, and Danica Porobic.
Provenance-based data skipping.

Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems.

Smoke: Fine-grained lineage at interactive speed.

Query centric partitioning and allocation for partially replicated database systems.


Generalized Containment

Definition (Generalized Containment)

$R \preceq_\Psi R'$ for two relations $R(a_1, \ldots, a_n)$ and $R'(b_1, \ldots, b_n)$ if there exists a mapping $\mathcal{M} \subseteq R \times R'$ such that:

- $\forall t \in R : \exists t' \in R' : \mathcal{M}(t, t')$
- $\forall t, t_1, t_2 : \mathcal{M}(t, t_1) \land \mathcal{M}(t, t_2) \rightarrow t_1 = t_2$
- $\forall (t, t') \in \mathcal{M} : (t, t') \models \Psi$