The (Potential) Perks of Integrating Provenance Support into Database Engines

Boris Glavic, Gopal Shankar

Illinois Institute of Technology
DBGroup

GCASR 2013, May 3, 2013

1 Master student
Outline

1 Motivation

2 Research Directions

3 Improving Performance with Provenance-aware Operators

4 Conclusions
Given a piece of data

- How do we know . . .
  - which data it is derived from?
  - which transformations (SQL) were used to create it?
  - who created it?
  - . . .

Example

<table>
<thead>
<tr>
<th>shop</th>
<th>rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migros</td>
<td>125</td>
</tr>
<tr>
<td>Coop</td>
<td>25</td>
</tr>
</tbody>
</table>
Given a piece of data

- How do we know . . .
  - which data it is derived from?
  - which transformations (SQL) where used to create it?
  - who created it?
  - . . .

Example

Compute the revenue for each shop as sum of prices of items sold

```
SELECT shop, 
    sum(price) AS rev
FROM sales, items
WHERE itemId = id
GROUP BY shop
```

<table>
<thead>
<tr>
<th>shop, rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migros 125</td>
</tr>
<tr>
<td>Coop 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>shop</th>
<th>itemId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migros</td>
<td>1</td>
</tr>
<tr>
<td>Migros</td>
<td>3</td>
</tr>
<tr>
<td>Coop</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
</tr>
</tbody>
</table>
Given a piece of data

- How do we know ... 
  - which data it is derived from?
  - which transformations (SQL) were used to create it?
  - who created it?
  - ...

Definition (Data Provenance)

Information about the origin and creation process of data.

Example

```
<table>
<thead>
<tr>
<th>shop</th>
<th>rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>125</td>
</tr>
<tr>
<td>t₂</td>
<td>25</td>
</tr>
</tbody>
</table>
```

```
SELECT shop,
    sum(price) AS rev
FROM sales, items
WHERE itemId = id
GROUP BY shop
```

```
<table>
<thead>
<tr>
<th>shop</th>
<th>itemId</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>1</td>
</tr>
<tr>
<td>s₁</td>
<td>3</td>
</tr>
<tr>
<td>s₁</td>
<td>3</td>
</tr>
<tr>
<td>s₂</td>
<td>1</td>
</tr>
<tr>
<td>s₂</td>
<td>2</td>
</tr>
<tr>
<td>s₃</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>i₁</td>
<td>100</td>
</tr>
<tr>
<td>i₂</td>
<td>10</td>
</tr>
<tr>
<td>i₃</td>
<td>25</td>
</tr>
</tbody>
</table>
```
Given a piece of data

- How do we know . . .
  - which data it is derived from?
  - which transformations (SQL) where used to create it?
  - who created it?
  - . . .

Definition (In this talk)

Data-Dependencies: which input data does query output depend on.

Example

<table>
<thead>
<tr>
<th>shop</th>
<th>rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migros</td>
<td>125</td>
</tr>
<tr>
<td>Coop</td>
<td>25</td>
</tr>
</tbody>
</table>

```
SELECT shop, 
    sum(price) AS rev
FROM sales, items
WHERE itemId = id
GROUP BY shop
```
Use Cases

- Debugging (tracking the sources of errors)
- Propagating annotations
- Gain deeper understanding of data and transformations
  - Estimate quality, trust
- Access Control (PBAC)
- Supporting technology (e.g., Probabilistic databases, Creating test databases)

Application Domains

- Complex database queries, e.g., datawarehousing
- E-science and curated databases
- Data integration
How to compute Provenance?

Annotation Propagation
- Annotate inputs
- Propagate these annotations during query processing
- Combine/filter annotations based on data-dependencies of operators

Example
- Annotate inputs with tuple IDs (singleton set)
- Join: Set union
- Duplicate Removal: Set union

Example
```
SELECT DISTINCT R.a
FROM R, S
WHERE R.a = S.c;
```

<table>
<thead>
<tr>
<th>R</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>r2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
How to compute Provenance?

Example

- Annote inputs with tuple IDs (singleton set)
- Join: Set union
- Duplicate Removal: Set union

\[
\left(\{r_1\} \cup \{s_1\}\right) \cup \left(\{r_1\} \cup \{s_2\}\right) = \{r_1, s_1, s_2\}
\]

Example

Result

\[
\{r_1, s_1, s_2\}
\]

SELECT DISTINCT R.a
FROM R, S
WHERE R.a = S.c;

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
# Provenance Representation

## Relational encoding of provenance

1. **Standard relations**
   - Extend query result tuples with complete input tuples in provenance (or only ID columns)
   - Duplicate tuples to fit in provenance

2. **New data types (DBMS extensibility mechanism)**
   - Use new/existing complex data type to represent provenance

### Example (Provenance)

\[ \{ < r_1, s_1 >, < r_1, s_2 > \} \]

### Example

```sql
SELECT DISTINCT R.a
FROM R, S
WHERE R.a = S.c;
```

<table>
<thead>
<tr>
<th>R</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>r2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
</tbody>
</table>

↑
Relation with Provenance + Query Result Data

- **Data**: result tuple + all tuples from input tables in provenance
  - Result tuple might have to be duplicated!
- **Schema**: + input attributes (renamed)

Example (Query Result)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result q</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Relation with Provenance + Query Result Data

- **Data**: result tuple + all tuples from input tables in provenance
  - Result tuple might have to be duplicated!
- **Schema**: + input attributes (renamed)

Example (Provenance Representation)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result + Provenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1 1 2 1 2 1 2 1 3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1 1 2 1 3</td>
</tr>
</tbody>
</table>
Relation with Provenance + Query Result Data

- **Data**: result tuple + all tuples from input tables in provenance
  - Result tuple might have to be duplicated!
- **Schema**: + input attributes (renamed)

Example (Provenance Representation)

\[
\begin{array}{cc}
R & S \\
\hline
a & b \\
1 & 2 \\
3 & 4 \\
\end{array}
\rightarrow
\begin{array}{cc}
q \\
\hline
\end{array}
\begin{array}{cccc}
\text{Result + Provenance} \\
a & P(a) & P(b) & P(c) & P(d) \\
1 & 1 & 2 & 1 & 2 \\
3 & 1 & 2 & 1 & 3 \\
\end{array}
\]
**Representation - Single Value**

**Relation with Query Result Data + Extra Attribute**

- **Data**: result tuple + new attribute to store provenance
  - Value of new attribute: complete provenance for this output
- **Schema**: + new provenance attribute

**Example (Query Result)**

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
<th>q</th>
<th>Result q</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( b )</td>
<td>( c )</td>
<td>( d )</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Relation with Query Result Data + Extra Attribute

- **Data**: result tuple + new attribute to store provenance
  - Value of new attribute: complete provenance for this output
- **Schema**: + new provenance attribute

Example (Provenance Representation)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result + Provenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

q

Result + Provenance

<table>
<thead>
<tr>
<th>a</th>
<th>Prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{r_1, s_1, s_2}</td>
</tr>
</tbody>
</table>
Query rewrite - Standard relations

- Take original query $q$ and rewrite into $q^+$:
  - Attach provenance to inputs (on-the-fly or join existing)
  - Propagate through operations
  - $\Rightarrow$ Compute original results + provenance
State-of-the-art - Provenance Generation

**Advantages**
- Reuse of existing DB technology
- DBMS independence (to some extend)

**Disadvantages**
- Unnecessarily complicated and inefficient computation
  - Impossible to propagate provenance through some operators
    - need to duplicate parts of the query
- Hard to benefit from overlap and structure in provenance
- Complex queries that are hard to optimize
Query rewrite - Single Value

- Take original query $q$ and rewrite into $q^+$:
  - Attach provenance to inputs (on-the-fly or join existing)
  - User-defined functions (UDF) to combine propagated provenance
Advantages
- Reuse of existing DB technology
- DBMS independence (to some extend)

Disadvantages
- Unnecessarily complicated and inefficient computation
  - Impossible to propagate provenance through some operators
  - DBMS not optimized for very large attribute values
  - Optimizer has to handle provenance computations as black-boxes
- Hard to benefit from overlap and structure in provenance
**Fundamental Problem - Performance**

### Implications of loose integration of provenance

- Severe performance limitations
  - SQL and DBMS not designed for provenance \(\Rightarrow\) complex queries
  - Different data-flow (mostly append only, less reduction) \(\Rightarrow\) relational execution engine not a good fit
  - Provenance structure predictable and overlapping \(\Rightarrow\) no exploited
- \(\Rightarrow\) hinder broad adaptation
  - Access Control (**performance critical**)
  - Supporting technology, e.g., probabilistic data (**performance critical**)

### What can be gained by full integration of provenance in DBMS engine?

- Provenance-aware operators
- Provenance optimized data-flow during query processing
- Performance optimized compression techniques
Outline

1. Motivation
2. Research Directions
3. Improving Performance with Provenance-aware Operators
4. Conclusions
Tight Integration of Provenance Computation in the DBMS core

- Provenance-aware operators
  - Simplify queries ⇒ easier to optimize
  - Direct propagation without “rerouting” ⇒ less redundancy

- Compression schemes exploiting provenance structure
  - Faster compression/decompression ⇒ applicable during provenance computation
  - Execute operations on compressed provenance directly
  - Approximate provenance

- Investigate non-traditional data-flows for provenance
  - Cache provenance between queries
  - Share provenance between operators ⇒ divert from traditional operator base processing
  - . . .
Research Approach

Tight Integration of Provenance Computation in the DBMS core

- **Provenance-aware operators**
  - Simplify queries ⇒ easier to optimize
  - Direct propagation without “rerouting” ⇒ less redundancy

- **Compression schemes exploiting provenance structure**
  - Faster compression/decompression ⇒ applicable during provenance computation
  - Execute operations on compressed provenance directly
  - Approximate provenance

- **Investigate non-traditional data-flows for provenance**
  - Cache provenance between queries
  - Share provenance between operators ⇒ divert from traditional operator base processing
  - ...
Declarative Summaries

Rational

- Overlap in provenance
- Avoid generating large provenance for intermediates if not used in final provenance
  - Performance gain!

Approach

- Use declarative descriptions as placeholders for actual provenance
- Only expand when necessary
- Suited to present provenance to user?
Example

```
SELECT count(*), R.b
  (SELECT R.a
   FROM R
   WHERE EXISTS (SELECT * FROM S WHERE S.c > 5) AND R.a < 10)
GROUP BY R.b
HAVING count(*) > 1000
```

- Each output will have all tuples from \( S \) with \( S.c > 5 \) in its provenance.
- Propagation will attach this potentially large set of tuples
  - to every tuple from \( R \)
  - many of these tuples may belong to groups that will be filtered eventually
- ⇒ unnecessary work
Declarative Summaries

Example

```
SELECT count(*) , R.b
  (SELECT R.a
   FROM R
   WHERE EXISTS (SELECT * FROM S WHERE S.c > 5) AND R.a < 10)
GROUP BY R.b
HAVING count(*) > 1000
```

Provenance

<table>
<thead>
<tr>
<th>a</th>
<th>Prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{r_1, s_3, s_5, ...}</td>
</tr>
<tr>
<td>13</td>
<td>{r_5, s_3, s_5, ...}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Using Declarative Summarization

<table>
<thead>
<tr>
<th>a</th>
<th>Prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{r_1, \sigma_{c&gt;5}(S)}</td>
</tr>
<tr>
<td>13</td>
<td>{r_5, \sigma_{c&gt;5}(S)}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Non-traditional data-flows
1 Motivation
2 Research Directions
3 Improving Performance with Provenance-aware Operators
4 Conclusions
Problem - Performance

Data-flow and access patterns

- Provenance computation $\neq$ Regular queries
  - Tuple breadth constantly increasing
  - Coupling of input data with intermediate data
  - Overlap

Direct propagation through operators not always possible

- Adding additional rows to input of an aggregation
- $\Rightarrow$ change result!
## Provenance-aware Operators

### Approach
- Operators optimized for provenance specific tasks
- Support direct propagation of provenance
- Implement and evaluate new operators in **Perm**
  - Perm is a relational provenance system based on query rewrite
  - Implemented as an extension to Postgres

### Rationale
- Less complex queries
- Specialized code
- Less redundancy in computation
Case study: AggProject - A Provenance-aware Aggregation Operator

Problem

- Cannot propagate provenance through aggregation directly
  - Would change aggregation results
- Compute aggregation provenance
  - Compute the original aggregation
  - Compute provenance for query below aggregation
  - Join these two
- Complexity
  - Each level of aggregation
    - Doubles query size
    - Adds additional join
  - Exponential in levels of aggregations!
Case study: AggProject - A Provenance-aware Aggregation Operator

Solution

- Develop a new type of aggregation that caches input provenance and attaches it to aggregation results
Preliminary Results

Implementation in Perm

- Add AGGPROJECT as new clause to SQL
- Physical aggproject operator using aggregation by sorting
- “Hack” integration with query optimizer
- Integration with provenance query rewrites
- Support multiple levels of aggregation
Anecdotal Performance Results

Setup

- TPC-H Benchmark Dataset (1GB)
- Perm vs. Perm with Aggproject

![Graph showing runtime and levels of aggregation with and without Aggproject]

![Graph showing runtime factor for Aggproject vs. regular rewrite for TPC-H Queries with and without Aggproject]
Next Steps

**AggProject**
- Develop cost-model
- Implemented Hash-based variant
  - In-memory
  - With disk caching

**Provenance-aware Operators**
- Set operations
- Nested subqueries
Conclusions

- Provenance become more and more important
- State-of-the-art of relational provenance systems unsatisfactory
  - Query rewrite, custom data types, stored procedures
  - Seriously performance limitations
    - Reduces adaptation for use cases that benefit from provenance
- Integration of provenance in DBMS engine core
  - Potential for addressing the performance bottleneck
Future Work

- Continue provenance-aware operators approach
- Efficient provenance representations and declarative summarizations
  - Develop model and equivalences
  - Cost model and performance analysis
- Non-traditional data flows
  - Caching provenance
  - Operators shared provenance data structures
  - Indexing
Questions?

- **Homepage**: http://www.cs.iit.edu/~glavic/
- **DBGroup**: http://www.cs.iit.edu/~dbgroup/
- **Perm**:
  - Project page: http://www.cs.iit.edu/~dbgroup/research/perm.php
  - Svn repository: https://permdbms.svn.sourceforge.net/svnroot/permdbms/trunk