Scalable Semantic Web Data Management Using Vertical Partitioning

Daniel Abadi\textsuperscript{2→1}, Adam Marcus\textsuperscript{2}, Samuel Madden\textsuperscript{2}, and Kate Hollenbach\textsuperscript{2}

\textsuperscript{1}Yale University \hspace{1cm} \textsuperscript{2}MIT

September 27, 2007
Sam Madden's Publications page, Sam Madden's homepage


Sam Madden's Publications page, Sam Madden's homepage

[PDF] 2007 Michael Stonebraker, Samuel Madden, Daniel Abadi, Stavros Harizopoulos, Nabil Hachem, Pat... 


Sam Madden's Publications page, Sam Madden's homepage

Scheduling. In Proceedings of SOSP, 2007. [PDF] Daniel Abadi, Adam Marcus, Samuel Madden, Katherine... 


VLDB 2007 - 33rd Very Large Data Bases Conference page, 33rd International Conference on Very Large Data Bases website

Page monitored for first time

(Yahoo! Research, USA) Scalable Semantic Web Data Management Using Vertical Partitioning Daniel Abadi, Adam Marcus, Samuel Madden, Katherine Hollenbach (MIT) Research Session 11: Time-Series Data...


VLDB 2007 - 33rd Very Large Data Bases Conference page, 33rd International Conference on Very Large Data Bases website

Page monitored for first time

End of an Architectural Era (It's Time for a Complete Rewrite) Michael Stonebraker, Samuel Madden, Daniel Abadi, Stavros Harizopoulos (MIT, USA). Nabil Hachem (AvanGarde Consulting, USA), Pat Helland...

RDF Data Is Proliferating

- Semantic Web vision: make Web machine-readable
- RDF is the data model behind Semantic Web
- Increasing amount of data published using RDF
  - Swoogle indexes 2,271,350 Semantic Web documents
- Biologists seem sold on Semantic Web
  - Integrated data from Swiss-Prot, TrEMBL, and PIR protein databases available in RDF (500 million statements)
DBFacebook: A New Social Networking Application

RDF Data Model

- Mike Stonebraker
- David DeWitt

Things found in nature (streams, sequoias, auroras)

Elastic/Velcro/Anything, "One-size-fits-all"

Double blind reviewing

The Design of Postgres
- SIGMOD
- Implementation Techniques for Main Memory Database Systems
- VLDB
- GAMMA - A High Performance Dataflow Database Machine
DBFacebook: A New Social Networking Application

- Things found in nature (streams, sequoias, auroras)
- Elastic/Velcro/Anything “One-size-fits-all”
- http://DBFaceBook.com/PersonID1
  - foaf:name
  - foaf:likes
dbfb: dislikes
- Double blind reviewing
- dbfb: authorOf
- dbfb: title
- The Design of Postgres
- dbfb:SIGMOD
- Implementation Techniques for Main Memory Database Systems
- dbfb:VLDB
- GAMMA – A High Performance Dataflow Database Machine

- http://DBFaceBook.com/PersonID2
  - foaf:name
  - foaf:dislikes
dbfb: authorOf
- dbfb: venue
- dbfb: title
- Mike Stonebraker
- David DeWitt
- foaf:name
- rdf:type
- foaf:knows
- dbfb: authorOf
- foaf:knows
- rdf:type
- foaf:Person
- dbfb: dislikes
- http://DBFaceBook.com/Pub101
  - dbfb: title
- http://DBFaceBook.com/Pub102
  - dbfb: venue
- http://DBFaceBook.com/Pub103
  - dbfb: venue
- dbfb: authorOf
- dbfb:SIGMOD
- dbfb:VLDB
- dbfb:SIGMOD
- dbfb:VLDB
- dbfb:SIGMOD
- dbfb:VLDB
- dbfb:SIGMOD
- dbfb:VLDB
- dbfb:SIGMOD
- dbfb:VLDB
- dbfb:SIGMOD
- dbfb:VLDB
RDF Data Management

- Early projects built their own RDF stores
- Trend now towards storing in RDBMSs
- Paper examines 3 approaches for storing RDF data in a RDBMS …
DBFacebook RDF Graph

- Mike Stonebraker
  - likes: Things found in nature (streams, sequoias, auroras)
  - dislikes: Elastic/Velcro/Anything “One-size-fits-all”

- David DeWitt
  - dislikes: Double blind reviewing

- PersonID1
  - type: Person
  - knows: Mike Stonebraker, David DeWitt

- PersonID2
  - type: Person
  - knows: Mike Stonebraker, David DeWitt

- Pub101
  - title: The Design of Postgres

- Pub102
  - title: Implementation Techniques for Main Memory Database Systems

- Pub103
  - title: GAMMA – A High Performance Dataflow Database Machine

- SIGMOD
  - venue: personID1

- VLDB
  - venue: personID2
Approach 1: Triple Stores

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>type</td>
<td>Person</td>
</tr>
<tr>
<td>PersonID1</td>
<td>name</td>
<td>“Mike Stonebraker”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>likes</td>
<td>“Things found in nature (streams, sequoias, auroras)”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>dislikes</td>
<td>“Elastic/Velcro/Anything ‘One-size-fits-all’”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub101</td>
</tr>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>type</td>
<td>Person</td>
</tr>
<tr>
<td>PersonID2</td>
<td>name</td>
<td>“David DeWitt”</td>
</tr>
<tr>
<td>PersonID2</td>
<td>dislikes</td>
<td>“Double blind reviewing”</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub103</td>
</tr>
<tr>
<td>Pub101</td>
<td>title</td>
<td>“The Design of Postgres”</td>
</tr>
<tr>
<td>Pub101</td>
<td>venue</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub102</td>
<td>title</td>
<td>“Implementation Techniques for Main Memory Databases”</td>
</tr>
<tr>
<td>Pub102</td>
<td>venue</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub103</td>
<td>title</td>
<td>“GAMMA – A High Performance Dataflow Database”</td>
</tr>
<tr>
<td>Pub103</td>
<td>venue</td>
<td>VLDB</td>
</tr>
</tbody>
</table>
DBFacebook RDF Graph

Mike Stonebraker

PersonID1

likes

Elastic/Velcro/Anything “One-size-fits-all”

Dislikes

Things found in nature (streams, sequoias, auroras)

Dislikes

David DeWitt

PersonID2

knows

knows

knows

Double blind reviewing

AuthorOf

AuthorOf

AuthorOf

The Design of Postgres

Pub101

Title

Implementation Techniques for Main Memory Database Systems

Pub102

Venue

VLDB

Venue

GAMMA – A High Performance Dataflow Database Machine

Pub103

Title
# Approach 2: Property Tables

<table>
<thead>
<tr>
<th>Subject</th>
<th>name</th>
<th>likes</th>
<th>dislikes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>Mike Stonebraker</td>
<td>Things found in nature (streams, sequoias, auroras)</td>
<td>Elastic/Velcro/Anything ‘One-size-fits-all’</td>
</tr>
<tr>
<td>PersonID2</td>
<td>David DeWitt</td>
<td>NULL</td>
<td>Double Blind Reviewing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>title</th>
<th>venue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pub101</td>
<td>“The Design of Postgres”</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub102</td>
<td>“Implementation Techniques for Main Memory Databases”</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub103</td>
<td>“GAMMA – A High Performance Dataflow Database”</td>
<td>SIGMOD</td>
</tr>
</tbody>
</table>
DBFacebook RDF Graph

- **PersonID1**
  - **likes**
    - Things found in nature (streams, sequoias, auroras)
  - **dislikes**
    - Elastic/Velcro/Anything “One-size-fits-all”

- **PersonID2**
  - **knows**
    - Mike Stonebraker
    - David DeWitt

- **Person**
  - **type**

- **Mike Stonebraker**
  - **name**
  - **type**
  - **knows**
    - PersonID1
    - PersonID2

- **David DeWitt**
  - **name**
  - **type**
  - **dislikes**
    - Double blind reviewing

- **Pub101**
  - **title**
    - The Design of Postgres
  - **type**
  - **authorOf**
    - PersonID1

- **Pub102**
  - **title**
    - Implementation Techniques for Main Memory Database Systems
  - **venue**
    - SIGMOD
  - **authorOf**
    - PersonID1

- **Pub103**
  - **title**
    - GAMMA – A High Performance Dataflow Database Machine
  - **venue**
    - VLDB
  - **authorOf**
    - PersonID2
### Approach 3: One-table-per-property

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
<th>Subject</th>
<th>Object</th>
<th>Subject</th>
<th>Object</th>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>Mike Stonebraker</td>
<td>PersonID1</td>
<td>Elastic/Velcro/Anything</td>
<td>PersonID1</td>
<td>Things found in nature (streams, sequoias, auroras)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PersonID2</td>
<td>David DeWitt</td>
<td>PersonID1</td>
<td>‘One-size-fits-all’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PersonID2</td>
<td>Double Blind Reviewing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Paper Contributions

- Explores advantages/disadvantages of these approaches
  - Triples stores are the dominant choice
  - Property Tables implemented by Jena and Oracle
  - We propose the one-table-per-property approach

- Shows how a column-store can be extended to implement the one-table-per-property approach

- Introduces benchmark for evaluating RDF stores
Results Synopsis

- Triple-store really slow on benchmark with 50M triples
- Property-tables and one-table-per-property approaches are factor of 3 faster
- One-table-per-property with column-store yields another factor of 10
Querying RDF Data

- SPARQL is the dominant language
- Examples:
  - `SELECT ?name
    WHERE { ?x type Person .
       ?x name ?name }
  
- `SELECT ?likes ?dislikes
  WHERE { ?x title "Implementation Techniques for Main Memory Databases" .
       ?y authorOf ?x .
       ?y likes ?likes .
       ?y dislikes ?dislikes }`
Translation to SQL over triples is easy

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>type</td>
<td>Person</td>
</tr>
<tr>
<td>PersonID1</td>
<td>name</td>
<td>“Mike Stonebraker”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>likes</td>
<td>“Things found in nature (streams, sequoias, auroras)”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>dislikes</td>
<td>“Elastic/Velcro/Anything ‘One-size-fits-all’”</td>
</tr>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub101</td>
</tr>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>type</td>
<td>Person</td>
</tr>
<tr>
<td>PersonID2</td>
<td>name</td>
<td>“David DeWitt”</td>
</tr>
<tr>
<td>PersonID2</td>
<td>dislikes</td>
<td>“Double blind reviewing”</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub103</td>
</tr>
<tr>
<td>Pub101</td>
<td>title</td>
<td>“The Design of Postgres”</td>
</tr>
<tr>
<td>Pub101</td>
<td>venue</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub102</td>
<td>title</td>
<td>“Implementation Techniques for Main Memory Databases”</td>
</tr>
<tr>
<td>Pub102</td>
<td>venue</td>
<td>SIGMOD</td>
</tr>
<tr>
<td>Pub103</td>
<td>title</td>
<td>“GAMMA – A High Performance Dataflow Database”</td>
</tr>
<tr>
<td>Pub103</td>
<td>venue</td>
<td>VLDB</td>
</tr>
</tbody>
</table>
SPARQL → SQL (over triple store)

- Query 1 SPARQL:
  ```sparql
  SELECT ?name
  WHERE { ?x type Person .
     ?x name ?name }
  ```

- Query 1 SQL:
  ```sql
  SELECT B.object
  FROM triples AS A, triples as B
  WHERE A.subject = B.subject
     AND A.property = "type"
     AND A.object = "Person"
     AND B.predicate = "name"
  ```
SPARQL → SQL (over triple store)

- **Query 2 SPARQL:**
  ```sparql
  SELECT ?likes ?dislikes
  WHERE { ?x title "Implementation Techniques for Main Memory Databases" .
    ?y authorOf ?x .
    ?y likes ?likes .
    ?y dislikes ?dislikes }
  ```

- **Query 2 SQL:**
  ```sql
  SELECT C.object, D.object
  FROM triples AS A, triples AS B, triples AS C, triples AS D
  WHERE A.subject = B.object
    AND A.property = "title"
    AND A.object = "Implementation Techniques for Main Memory Databases"
    AND B.property = "authorOf"
    AND B.subject = C.subject
    AND C.property = "likes"
    AND C.subject = D.subject
    AND D.property = "dislikes"
  ```
Triple Stores

- Accessing multiple properties for a resource require subject-subject joins
- Path expressions require subject-object joins
- Can improve performance by:
  - Indexing each column
  - Dictionary encoding string data
- Ultimately: Do not scale
Property Tables Can Reduce Joins

<table>
<thead>
<tr>
<th>Subject</th>
<th>name</th>
<th>likes</th>
<th>dislikes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>Mike Stonebraker</td>
<td>Things found in nature (streams, sequoias, auroras)</td>
<td>Elastic/Velcro/Anything ‘One-size-fits-all’</td>
</tr>
<tr>
<td>PersonID2</td>
<td>David DeWitt</td>
<td>NULL</td>
<td>Double Blind Reviewing</td>
</tr>
</tbody>
</table>

Left-over triples

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub101</td>
</tr>
<tr>
<td>PersonID1</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub102</td>
</tr>
<tr>
<td>PersonID2</td>
<td>authorOf</td>
<td>Pub103</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Property Tables

- Complex to design
  - If narrow: reduces nulls, increases unions/joins
  - If wide: reduces unions/joins, increases nulls
- Implemented in Jena and Oracle
  - But main representation of data is still triples
Table-Per-Property Approach

<table>
<thead>
<tr>
<th>name</th>
<th>dislikes</th>
<th>likes</th>
<th>authorOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Object</td>
<td>Subject</td>
<td>Object</td>
</tr>
<tr>
<td>PersonID1</td>
<td>Mike Stonebraker</td>
<td>PersonID1</td>
<td>Elastic/Velcro/Anything ‘One-size-fits-all’</td>
</tr>
<tr>
<td>PersonID2</td>
<td>David DeWitt</td>
<td>PersonID2</td>
<td>Double Blind Reviewing</td>
</tr>
</tbody>
</table>

- + Nulls not stored
- + Easy to handle multi-valued attributes
- + Only need to read relevant properties
- – Still need joins (but they are linear merge joins)
Materialized Paths

- Mike Stonebraker
- David DeWitt

**Likes**:
- Things found in nature (streams, sequoias, auroras)
- Elastic/Velcro/Anything “One-size-fits-all”

**Dislikes**:
- Double blind reviewing

**Publications**:
- Pub101: The Design of Postgres
- Pub102: Implementation Techniques for Main Memory Database Systems
- Pub103: GAMMA – A High Performance Dataflow Database Machine

**Conferences**:
- SIGMOD
- VLDB
Accelerating Path Expressions

- Materialize Common Paths
  - Improved property table performance by 18-38%
  - Improved one-table-per-property performance by 75-84%

- Use automatic database designer (e.g., C-Store /Vertica) to decide what to materialize

<table>
<thead>
<tr>
<th>Subject</th>
<th>authorOf:title</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonID1</td>
<td>The Design of the Postgres</td>
</tr>
<tr>
<td>PersonID1</td>
<td>Implementation Techniques for Main Memory Database Systems</td>
</tr>
<tr>
<td>PersonID2</td>
<td>Implementation Techniques for Main Memory Database Systems</td>
</tr>
<tr>
<td>PersonID2</td>
<td>GAMMA – A High Performance Dataflow Database Machine</td>
</tr>
</tbody>
</table>
One-table-per-property → Column-Store

- Can think of one-table-per-property as vertical partitioning super-wide property table
- Column-store is a natural storage layer to use for vertical partitioning

- Advantages:
  - Tuple Headers Stored Separately
  - Column-oriented data compression
  - Do not necessarily have to store the subject column
  - Carefully optimized merge-join code
Library Benchmark

- Data
  - Real Library Data (50 million RDF triples)
  - Data acquired from a variety of diverse sources (some quite unstructured)

- Queries
  - Automatically generated from the Longwell RDF browser

- Details in paper …
Results

<table>
<thead>
<tr>
<th>Query</th>
<th>Triple Store</th>
<th>Prop. Table</th>
<th>Vert. Part.</th>
<th>C-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>24.63</td>
<td>12.66</td>
<td>12.66</td>
<td>0.66</td>
</tr>
<tr>
<td>Q2</td>
<td>157</td>
<td>18.37</td>
<td>41.7</td>
<td>1.64</td>
</tr>
<tr>
<td>Q3</td>
<td>224.3</td>
<td>579.8</td>
<td>71.3</td>
<td>9.28</td>
</tr>
<tr>
<td>Q4</td>
<td>27.67</td>
<td>28.54</td>
<td>35.49</td>
<td>2.24</td>
</tr>
<tr>
<td>Q5</td>
<td>408.7</td>
<td>47.85</td>
<td>52.34</td>
<td>15.88</td>
</tr>
<tr>
<td>Q6</td>
<td>212.7</td>
<td>101</td>
<td>84.6</td>
<td>10.81</td>
</tr>
<tr>
<td>Q7</td>
<td>38.37</td>
<td>6.1</td>
<td>13.25</td>
<td>1.44</td>
</tr>
<tr>
<td>Geo. Mean</td>
<td>97</td>
<td>38</td>
<td>36</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Experimented with storing RDF data using different schemas in RDMS (both row and column-oriented)
- Future work: build a fully-functional RDF database
  - Extracts and loads RDF data from structured, semi-structured, and unstructured data sources
  - Translates SPARQL to queries over vertical schema
  - Performs reasoning inside the DB
  - Use with biology research
- Excited about this work? Then …
Come To Yale!