SQL Beyond Structure: Text, Documents, and Key-Value Pairs

Daniel Tahara, Yale University and Hadapt
Daniel Abadi, Yale University and Hadapt
Important Trends

- Dramatic increase in utilization of “NoSQL” databases such as MongoDB, CouchDB, Riak, Cassandra, Hbase, etc.
- Key value and JSON are increasingly popular ways to represent data
  - See MongoDB’s $150 million “largest database funding ever”
- Performing analysis inside NoSQL databases not a major use-case
  - Poor analytics primitives
  - Poor integration with other data sources (data silos)
  - Poor scan performance
  - Poor integration with analytical tools
A Motivating Example

Relational Datastore

<table>
<thead>
<tr>
<th>userID</th>
<th>name</th>
<th>email</th>
<th>preferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walter White</td>
<td><a href="mailto:heisenberg@bb.com">heisenberg@bb.com</a></td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>Hank Schrader</td>
<td><a href="mailto:mineral@rock.com">mineral@rock.com</a></td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>Jesse Pinkman</td>
<td><a href="mailto:pink@thenewblack.ca">pink@thenewblack.ca</a></td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>Gustavo Fring</td>
<td><a href="mailto:gus@lospollos.com">gus@lospollos.com</a></td>
<td>true</td>
</tr>
</tbody>
</table>

Text Datastore

<table>
<thead>
<tr>
<th>userID</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is your return policy?</td>
</tr>
<tr>
<td>2</td>
<td>Do you have bulk discounts?</td>
</tr>
<tr>
<td>4</td>
<td>I can’t figure out your Website. You should fire your designer!</td>
</tr>
</tbody>
</table>

JSON Datastore

What if you want to send an email to all customers who indicate a potential desire to buy in bulk, with a deal on items they looked at?
Option 1: User Code

Extract data from each relevant store, perform join and further query analysis in user code.

```sh
#!/bin/sh

if ['some condition']
  then
    'do something'
  fi

'more complicated code'
```
Option 2: ETL

Use ETL to combine data from separate backends into one unified datastore.

**Relational Datastore**

<table>
<thead>
<tr>
<th>userID</th>
<th>name</th>
<th>email</th>
<th>preferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walter White</td>
<td><a href="mailto:heisenberg@bb.com">heisenberg@bb.com</a></td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>Hank Schrader</td>
<td><a href="mailto:mineral@rock.com">mineral@rock.com</a></td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>Jesse Pinkman</td>
<td><a href="mailto:pink@thenewblack.ca">pink@thenewblack.ca</a></td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>Gustavo Fring</td>
<td><a href="mailto:gus@lospollos.com">gus@lospollos.com</a></td>
<td>true</td>
</tr>
</tbody>
</table>

**JSON Datastore**

<table>
<thead>
<tr>
<th>JSON Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&quot;userID&quot;: 1, &quot;action&quot;: &quot;view&quot;, &quot;productName&quot;: &quot;door&quot;, &quot;price&quot;: 12.50, &quot;color&quot;: &quot;green&quot;, &quot;tags&quot;: [&quot;home&quot;, &quot;green&quot;]}</td>
</tr>
<tr>
<td>{&quot;userID&quot;: 3, &quot;action&quot;: &quot;purchase&quot;, &quot;price&quot;: 57.12}</td>
</tr>
<tr>
<td>{&quot;userID&quot;: 2, &quot;action&quot;: &quot;view&quot;, &quot;productName&quot;: &quot;window&quot;, &quot;color&quot;: &quot;orange&quot;, &quot;price&quot;: 52.50, &quot;tags&quot;: [&quot;home&quot;, &quot;orange&quot;]}</td>
</tr>
</tbody>
</table>

**Text Datastore**

<table>
<thead>
<tr>
<th>userID</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is your return policy?</td>
</tr>
<tr>
<td>2</td>
<td>Do you have bulk discounts?</td>
</tr>
<tr>
<td>4</td>
<td>I can’t figure out your Website. You should fire your designer!</td>
</tr>
</tbody>
</table>
### A “Unified” Datastore

<table>
<thead>
<tr>
<th>userID</th>
<th>name</th>
<th>emailAddr</th>
<th>preferred</th>
<th>json</th>
<th>emails</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walter White</td>
<td><a href="mailto:heisenberg@bb.com">heisenberg@bb.com</a></td>
<td>false</td>
<td>{&quot;eventName&quot;: 1, “action”:”view”, “productName”: “door”, “price”: 12.50,”color”:”green”, “tags”:”[“home”,”green”]}</td>
<td>What is your return policy?</td>
</tr>
<tr>
<td>2</td>
<td>Hank Schrader</td>
<td><a href="mailto:mineral@rock.com">mineral@rock.com</a></td>
<td>true</td>
<td>{&quot;eventName&quot;: 2, “action”:”view”, “productName”: “window”, “color”:”orange”, “price”:52.50,”tags”: ”[“home”,”orange”]}</td>
<td>Do you have bulk discounts?</td>
</tr>
<tr>
<td>3</td>
<td>Jesse Pinkman</td>
<td><a href="mailto:pink@thenewblack.ca">pink@thenewblack.ca</a></td>
<td>true</td>
<td>{&quot;eventName&quot;: 3, “action”:”purchase”,”price”: 57.12}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Gustavo Fring</td>
<td><a href="mailto:gus@lospollos.com">gus@lospollos.com</a></td>
<td>true</td>
<td></td>
<td>I can’t figure out your Website. You should fire your designer!</td>
</tr>
</tbody>
</table>

```sql
SELECT name,
    emailAddr,
    regex_value(json, "productName":\s*"([\^]*?)"", 1, ‘i’)
FROM T
WHERE regex_match(json, "action":\s*"view"”,’i’)
    AND emails like ‘%bulk%’;
```
### A Better Approach?

<table>
<thead>
<tr>
<th>userID</th>
<th>name</th>
<th>emailAddr</th>
<th>preferred</th>
<th>productName</th>
<th>price</th>
<th>tags</th>
<th>color</th>
<th>action</th>
<th>emails</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walter White</td>
<td><a href="mailto:heisenberg@bb.com">heisenberg@bb.com</a></td>
<td>no</td>
<td>door</td>
<td>12.50</td>
<td>home</td>
<td>green</td>
<td>view</td>
<td>What is your return policy?</td>
</tr>
<tr>
<td>2</td>
<td>Hank Schrader</td>
<td><a href="mailto:mineral@rock.com">mineral@rock.com</a></td>
<td>yes</td>
<td>window</td>
<td>52.50</td>
<td>home</td>
<td>orange</td>
<td>view</td>
<td>Do you have bulk discounts?</td>
</tr>
<tr>
<td>3</td>
<td>Jesse Pinkman</td>
<td><a href="mailto:pink@thenewblack.ca">pink@thenewblack.ca</a></td>
<td>yes</td>
<td></td>
<td>57.12</td>
<td></td>
<td></td>
<td>purchase</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Gustavo Fring</td>
<td><a href="mailto:gus@lospollos.com">gus@lospollos.com</a></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I can’t figure out your Website. You should fire your designer!</td>
</tr>
</tbody>
</table>

**SQL Query:**

```sql
SELECT name, emailAddr, productName
FROM T
WHERE action = 'view'
AND emails like '%bulk%';
```
Goal: Multistructured Tables

- Store data with various levels of structure side-by-side in an RDBMS
- Remove the ETL step in the processing pipeline
- Maintain a standard SQL interface to the data - allow elements to be manipulated as relational attributes irrespective of original format
- Bonus: Support for non-relational operations such as full text search
Implementation: Overview

Parser and Query Rewriter

RDBMS

Text Index

Storage Layer

SELECT name, emailAddr, productName
FROM customer_table
WHERE action = 'view' AND emails like '%bulk%';

{ "eventID": 1, "action": "view", "productName": "door", "price": 12.50, "color": "green", "tags": ["home", "green"] }

I can’t figure out your Website. You should fire your designer!

SELECT name, emailAddr, productName
FROM customer_table
WHERE action = 'view' AND emails like '%bulk%';

{ "eventID": 1, "action": "view", "productName": "door", "price": 12.50, "color": "green", "tags": ["home", "green"] }

us

er

name

emailAddr

Preferred?

json

emails

1

Walt

Heisenberg

@bb.com

false

{"eventID": 1, "action": "view", "productName": "door", "price": 12.50, "color": "green", "tags": ["home", "green"] }

What is your return policy?

2

Han

Schrader

mineral@rock.com

true

{"eventID": 2, "action": "view", "productName": "window", "color": "orange", "price": 52.50, "tags": ["home", "orange"] }

Do you have bulk discounts?

3

Jesse

Pinkman

@thenewblack.ca

true

{"eventID": 3, "action": "purchase", "price": 57.12 }

4

Gus

Fring

@lospo.com

true

I can’t figure out your Website. You should fire your designer!
Implementation: Details

- System does **not** store data according to the virtual, “universal” view
- Store key-value data in a serialized, key-value format in order balance competing needs:
  - Limit table width
  - Speed up attribute extraction
  - Identify which attributes exist
- Inverted indexes over all keys and text columns
- Query rewriter transforms:
  - references to serialized, virtual columns into key extractions
  - predicates over virtual columns into queries to the text index, applied as a filter set on the attribute ids
### Flexible Schema

- **JSON Example:**

  ```json
  {"userID": 4, "action": "view", "productName": "door", "price": 12.50, "color": "green", "tags": ["home", "green"], "size": "standard"}
  ```

- **User Information Table:**

<table>
<thead>
<tr>
<th>user ID</th>
<th>name</th>
<th>emailAddr</th>
<th>Preferred?</th>
<th>productName</th>
<th>price</th>
<th>tags</th>
<th>color</th>
<th>action</th>
<th>size</th>
<th>emails</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walter White</td>
<td><a href="mailto:heisenberg@bb.com">heisenberg@bb.com</a></td>
<td>no</td>
<td>door</td>
<td>12.50</td>
<td>home green</td>
<td>green</td>
<td>view</td>
<td></td>
<td>What is your return policy?</td>
</tr>
<tr>
<td>2</td>
<td>Hank Schrader</td>
<td><a href="mailto:mineral@rock.com">mineral@rock.com</a></td>
<td>yes</td>
<td>window</td>
<td>52.50</td>
<td>home orange</td>
<td>orange</td>
<td>view</td>
<td></td>
<td>Do you have bulk discounts?</td>
</tr>
<tr>
<td>3</td>
<td>Jesse Pinkman</td>
<td><a href="mailto:pink@thenewblack.ca">pink@thenewblack.ca</a></td>
<td>yes</td>
<td></td>
<td>57.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I can’t figure out your Website. You should fire your designer!</td>
</tr>
<tr>
<td>4</td>
<td>Gustavo Fring</td>
<td><a href="mailto:gus@lospollos.com">gus@lospollos.com</a></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>standard</td>
</tr>
</tbody>
</table>
Challenges with the Current Design

- Relying on a text index can be slow if the result set is large
- Can’t utilize B-tree indexes
- RDBMS optimizer often produces suboptimal query plans
Dynamic Column Creation/Deletion

- **Specifications:**
  - Create physical columns for most commonly used or frequently appearing keys.
  - Ongoing process. Columns can be created or deleted at any point.

- **Benefits:**
  - Can create indices on physical columns
  - Optimizer has better statistics on frequently queried data
  - Dynamic physical schema allows the system to adapt to the evolving nature of datasets
“Smart” Query Rewrite and UDFs

- Specification:
  - Define an extraction function and a text index query function for use within the RDBMS
  - Selectively execute text predicates in the RDBMS or push them out to the text index

- Benefits:
  - More code running within the RDBMS
  - Take into account system knowledge of the data to produce globally optimal execution
Alternatives

- Entity-Attribute-Value
  - “flatten” key-value data types
  - stored in 3-column tables
- PostgreSQL 9.3
  - use JSON text data type
  - extraction operators
- Column Store
  - store in the form of universal relation
Conclusions

- Bevy of options for formatting data
- As long as SQL remains a standard of data analytics, there will be a need for unified solution
- Tighter RDBMS integration correlates with improved performance
Questions?

- Email: daniel.tahara@yale.edu
- Further Resources:
  - Hadapt Schemaless SQL:
    - http://hadapt.com/schemaless-sql-overview/
  - Yale Database Group:
    - http://db.cs.yale.edu/
  - My Website:
    - http://danieltahara.com/