Announcements

- HW 6:  
  • due Sunday night  
  • Up to 2 late days allowed  
- HW 7: (out today, much less complex than normal HW)  
  • due the Saturday after final exam  
  • NO LATE DAYS ALLOWED  
- Schedule for next week:  
  • Today: Semi-structured data  
  • Monday: SQL++ and HW 7  
  • Wednesday: Misc. topics  
  • Thursday section: exam review session  
  • Friday: Final exam

Outline

- AsterixDB as a case study of Document Store  
  • Semi-structured data model in JSON  
  • Introducing AsterixDB and SQL++

What is a "document" anyways?

- Loose terminology  
- Any "parsable" file qualifies  
  • Ex: MongoDB can handle CSV files

Semi-Structured Documents

- Some notion of tagging to mark down semantics  
- Examples:  
  • XML  
  • Protobuf  
  • JSON

Tags surround the respective data
Semi-Structured Documents

- Some notion of tagging to mark down semantics
- Examples:
  - XML
  - Protobuf
  - JSON

Able to record field number and type but not name

Tags introduce the respective data

Many applications have phased out XML in favor of JSON

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data

Less well-defined / More flexible

Less well-defined / More flexible

Basic retrieval process:
1. Retrieve table
2. Run through rows
3. Return data

Basic retrieval process:
1. Retrieve document
2. Parse document tree
3. Return data
Relational vs Semi-Structured Tradeoffs

<table>
<thead>
<tr>
<th>Relational Model</th>
<th>Semi-Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fixed schema</td>
<td>• Self-described schema</td>
</tr>
<tr>
<td>• Flat data</td>
<td>• Tree-structured data</td>
</tr>
</tbody>
</table>

Less well-defined/More flexible

Basic retrieval process:
1. Retrieve table
2. Run through rows
3. Return data

Inefficient encoding/Easy exchange of data

Note

• No database paradigm is "better" than another

• One-size does not fit all (M. Stonebraker)
  - Excellent article on data management in 21st century
  - http://cs.brown.edu/research/db/publications/fits_all.pdf

Everything is getting mixed up anyways

JSON Standard – Rules of the Game

• JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for human-readable data interchange"

```
{  
  "book":{  
    "id": "B1",  
    "language": "Java",  
    "author": "E. Sepp",  
    "year": 2015
  },  
  "author": "E. Sepp",  
  "id": "B1",  
  "language": "Java",  
  "year": 2015
}
```

Types

- Primitives include:
  - String (in quotes)
  - Numeric (unquoted number)
  - Boolean (unquoted true/false)
  - Null (literally just null)
- Structured data
  - Objects are an unordered collection of name-value pairs:
    - "name": "value"
    - Values can be primitives, objects, or arrays
  - Enclosed by {}
JSON Standard – Rules of the Game

• JavaScript Object Notation (JSON)
  • "Lightweight text-based open standard designed for human-readable data interchange"

Types
- Objects are an unordered collection of name-value pairs
  - Types: "null", "true", "false"
  - Values can be primitives, objects, or arrays
  - Enclosed by {

Arrays are an ordered list of values:
- Order is preserved in interpretation
- May contain any mix of types
- Enclosed by [

Read as "book" [{object1}, {object2}]
Can have mix of types like [{object1}, "string", 128, {object2}]

• JSON Standard too expressive
  • Implementations restrict syntax
  • Ex: Duplicate fields
Thinking About Semi-Structured Data

What does semi-structured data structure encode?

```
{"book": [
  {
    "id": "01",
    "language": "Java",
    "author": "H. Javerson",
    "year": 2015
  },
  {
    "author": "E. Sepp",
    "id": "07",
    "language": "C++",
    "sale": true
  }
]}
```

Tree semantics!

From Relational to Semi-Structured

What is a table in semi-structured land?

```
{ "person": [
  
  
  
  
] }
```

Tables are just an array of elements (rows)

Person

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

What is a table in semi-structured land?

Person

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>
From Relational to Semi-Structured

- What is a table in semi-structured land?
  - Tables are just an unnested array of elements (rows).
  - Rows are just simple connected objects.

- How can NULL be represented?
  - "person": [{ "name": "Magda", "phone": "555-5678" }]
  - "person": [{ "name": "Alvin", "phone": "555-6789" }]

- OK for field to be missing!

From Relational to Semi-Structured

- Person
  - Name: Magda
  - Phone: 555-123-4567
- Person
  - Name: Dan
  - Phone: 555-234-5678
- Person
  - Name: Alvin
  - Phone: 555-345-6789

- Name: Magda
  - Phone: 555-5678
- Name: Alvin
  - Phone: 555-6789
- Name: Dan
  - Phone: 555-7890

- Name: Magda
  - Phone: 555-5678
  - Name: Alvin
  - Phone: 555-6789
  - Name: Dan
  - Phone: 555-7890
From Relational to Semi-Structured

Are there things that the Relational Model can't represent?

- Array data
- Multi-part data

Are there things that the Relational Model can't represent?

- Flat data!
- Part data

How do we represent foreign keys?

Array with 2 objects

Object with 2 key-value pairs
<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td></td>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td></td>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th>Product</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Furby</td>
<td>1997</td>
</tr>
<tr>
<td>Alvin</td>
<td>Furby</td>
<td>2000</td>
</tr>
<tr>
<td>Alvin</td>
<td>Magic8</td>
<td>2012</td>
</tr>
</tbody>
</table>

- Precomputed equijoin!

<table>
<thead>
<tr>
<th>Orders</th>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>9.99</td>
<td></td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
<td></td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>

Is this many-to-many relationship easily convertible to JSON?

- People data will be duplicated!
- We might miss some people!
- Product data will be duplicated!
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Date</td>
</tr>
<tr>
<td>Product</td>
</tr>
<tr>
<td>Price</td>
</tr>
<tr>
<td>Furby</td>
</tr>
<tr>
<td>Magic8</td>
</tr>
<tr>
<td>Tomagachi</td>
</tr>
</tbody>
</table>

Big ideas:

- Semi-structured data is parsed
- Data model flexibility
- Potentially lots of redundancy
- Semi-structured data expresses unique patterns
  - Collection/multi-part data
  - Precompute joins
- Semi-structured data has limits
  - Relies on relational-like patterns in some situations

A Semi-structured DBMS

- AsterixDB as a case study of Document Store
  - Semi-structured data model in JSON
  - Introducing AsterixDB and SQL++

The 5 W’s of AsterixDB

- Who
  - M. J. Carey & co.
- What
  - “A Scalable, Open Source BDMS”
  - It is now also an Apache project
- Where
  - UC Irvine, Cloudera Inc, Google, IBM, ...
- When
  - 2014
- Why
  - To develop a next-gen system for managing semi-structured data

The 5 W’s of SQL++

- Who
  - K. W. Ong & Y. Papakonstantinou
- What
  - A query language that is applicable to JSON native stores and SQL databases
- Where
  - UC San Diego
- When
  - 2015
- Why
  - Stand in for other semi-structured query languages that lack formal semantics.
Why We are Choosing SQL++

- Strong formal semantics
  - Nested relational algebra: https://dl.acm.org/citation.cfm?id=588133
- Many systems adopting or converging to SQL++
  - Apache AsterixDB
  - CouchBase (N1QL)
  - Apache Drill
  - Snowflake

Asterix Data Model (ADM)

- Nearly identical to the JSON standard
- Some additions
  - New primitive: universally unique identifier (uuid)
    - Ex: 123e4567-e89b-12d3-a456-426654400000
  - New derived type: multiset
    - Like an array but unordered
    - Encapsulated by double curly braces `{{ }}`
- Queried data must be a multiset or array

Introducing the New and Improved SQL++

SQL++ Mini Demo

General Installation (Details in HW7 spec)

Download from: https://asterixdb.apache.org/download.html

Start local cluster from:
<asterix root>/opt/local/bin/start-sample-cluster

Use web browser for interaction, default address: 127.0.0.1:19002

Don’t forget to stop cluster when you’re done:
<asterix root>/opt/local/bin/stop-sample-cluster

SQL++ Hello World

General Usage:

Everything is running locally so make sure your computer doesn’t die (advise against SELECT *)

Don’t use attu, previous quarters people accidentally used other people’s instance

Learn something! I dare say that SQL++ is a model for many future query languages.

```
SELECT x.phone
FROM [ ]
  | "name": "Dan", "phone": [300, 150] |
  | "name": "Alvin", "phone": 420 |
AS x;

-- output, same for-loop semantics like in SQL
-- array data

/*
"phone": [300, 150]
*/
/*
"phone": 420
*/
```
SELECT x.phone
FROM [{ "name": "Dan", "phone": [300, 150]
  }
  AS x;

-- same output as array data
-- multiset data

-- error
SELECT x.phone
FROM (["name": "Dan", "phone": [300, 150]]
  AS x;

-- output
-- trying to query an object
/*
Type mismatch: function scan.collection expects its 1st input parameter to be type multiset or array, but the actual input type is object
TypeMismatchException*/

SELECT x.phone
FROM [{ "name": "Dan", "phone": [300, 150]
  },
  { "name": "Alvin", "phone": null}
  AS x;

-- output, null works like in SQL
-- null values
/*
{ "phone": [300, 150]
{ "phone": null }
*/

SELECT x.phone
FROM [{ "name": "Dan", "phone": [300, 150]
  },
  { "name": "Alvin", "phone": 420}
  AS x;

-- output, missing data is simply passed over (beware of typos!)
-- missing values
/*
{ "phone": [300, 150] }
*/

SELECT x.phone -- intentional typo
FROM [{ "name": "Dan", "phone": [300, 150]
  },
  { "name": "Alvin", "phone": 420}
  AS x;

-- output, beware of typos! No errors are thrown
/*
{ }
*/

SELECT x.phone
FROM [{ "name": "Dan", "phone": [300, 150]
  },
  { "name": "Alvin", "phone": 420}
  AS x;

WHERE is_array(x.phone) OR x.phone > 100
GROUP BY x.name, x.phone
HAVING x.name = "Dan" OR x.name = "Alvin"
SELECT x.phone
ORDER BY x.name DESC;

-- output, finally the keyword order matches FWGHOS!
/*
{ "phone": [300, 150]
{ "phone": 420 }
*/