Alternative Data Models – Toward NoSQL

Alternative Data Models

- XML Stores
- Object-Relational databases
- NoSQL databases

Object-relational impedance mismatch

- When implementing applications we work with objects
- Databases store data in tables
- Therefore, what does the programmer need to do?

XML Stores

- The need to store and search complex document structures
- Semi-structured data
  - Incomplete or irregular
  - Some structure
  - Changes rapidly or unpredictably
  - Schema-less/self-describing

XML Document

```
<stafflist>
        <staff branchNo="B005">
                <name>
                        <fname>Fred</fname>
                        <lname>Jones</lname>
                </name>
                <dob>1988-10-4</dob>
                <salary>50000</salary>
                <telelist>
                        <phone type="cell">885-933-9393</phone>
                        <phone type="home">885-789-5761</phone>
                </telelist>
        </staff>
…
</stafflist>
```

Do you need a Schema? Where is it?

DTD (Document Type Defintions)

XML Schema

RDF (Resource Description Language)

How do you Process XML Documents?

- DOM – Document Object Model
- SAX – Simple API for XML

Separation of Physical Presentation from Logical Structure

XSL -- eXtensible Stylesheet Language

XSLT – XSL Transform

```
….
<xsl:for-each select="stafflist/staff">
<tr>
    <td bordercolor="yellow"><xsl:value-of select="staffno"/></td>
    <td bordercolor="yellow"><xsl:value-of select="name/fname"/></td
</tr>
</xsl:for-each>
…
```

How do you Search?

- Xpath (XML Path Language)

child::staff	selects the staff elements that are children of root
child::staff[3]	selects the third staff element that is a child of the context node

Why Relational Model and XML?

- Treating web resources like a structured data
- Data interchange
- XML standard formats

XML Query Languages

```
where <staff>
        <salary>$S</salary>
        <name><fname>$f</fname><lname>$L</name></name>
        </staff>
in "http://www.dreamhome.co.uk/staff.xml"
        $S > 30000
construct <lname>$L</lname>
```

XQuery

doc(“staff_list.xml”)/stafflist/staff[1]//staffno

doc(“staff_list.xml”)/stafflist/staff[1 to 2]/staffno

doc(“staff_list.xml”)/stafflist/staff[@branchNo= “B0005”]/lname

How does PostgreSQL support XML?

XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</title><chapter>...</chapter></book>')

XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')

```
SELECT      xmlelement(name foo,
            xmlattributes('xyz' as bar),
            xmlelement(name abc),
            xmlcomment('test'),
            xmlelement(name xyz));
```

xmlelement

<foo bar="xyz"><abc/><!--test--><xyz/></foo>

```
SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
WHERE table_schema = 'pg_catalog';

Xmlforest
 ----------------------------------------------------------------------------------------------
<table_name>pg_authid</table_name><column_name>rolname</column_name>
<table_name>pg_authid</table_name><column_name>rolsuper</column_name> ...
```

xpath(xpath, xml [, nsarray])

SELECT xpath('/my:a/text()', myxml);

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

JSON – Javascript Object Notation

Use In AJAX Programming (Javascript example)

```
var my_JSON_object = {};
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.onreadystatechange = function () {
        var done = 4, ok = 200;
        if (http_request.readyState == done && http_request.status == ok) {
                my_JSON_object = JSON.parse(http_request.responseText);
         }
};
http_request.send(null);
```

NoSQL Databases

- Document-style stores
  – key-value pairs PLUS document playload
  (Key, Value) → payload_blob
- Key-value stores
  (Key, Value) → [key->payload_blob, key→ paylaod_blob, …]

Observations …

- Record at a time processing of data …
- Steps
  1. Given a key go grab some (semi-structured) data
  2. Process the data
  3. Find another key and goto 1
- NO STRUCTURED QUERY LANGUAGE (a step backwards?)

Why NoSQL

- Performance argument
  …
- Flexibility argument
  …

(see Stonebraker, 2010)

AWS DynamoDB

- Managed
- Scalable
- Fast
- Durable and Highly Available
- Flexible
- Low cost
- Distributed!

Some System Assumptions

- Query model
  - Simple read and write
  - State in binary objects identified by a key
- ACID (Atomicity, Consistency, Isolation, Durability)
  - Full ACID properties are given up
- Efficiency

Data Model

- No Schema!

```
ProductCatalog ( Id, ... )
```

```
{
    Id = 101
    ProductName = "Book 101 Title"
    ISBN = "111-1111111111"
    Authors = [ "Author 1", "Author 2" ]
    Price = -2
    Dimensions = "8.5 x 11.0 x 0.5"
    PageCount = 500
    InPublication = 1
    ProductCategory = "Book"
}
```

```
{
    Id = 201
    ProductName = "18-Bicycle 201"
    Description = "201 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 100
    Gender = "M"
    Color = [ "Red", "Black" ]
    ProductCategory = "Bike"
}
```

Primary Key

1. Hash Type Primary Key

2. Hash and Range Type Primary Key

For the power data what would be the primary keys?

Data Types

- Scalar data types
  - Number
  - String
  - Binary
- Multi-valued types
  - String set
  - Number set
  - Binary set

In Summary

1. Tables
   - Create / update / delete tables
2. Items
   - Add / update / delete items
3. Attributes

Query Read and Consistency

- Multiple items of each item are kept
- Two kinds of reads:
  1. Eventually consistent reads
  2. Strongly consistent reads

Updates and Concurrency Control

- "Lost update" problem
- Two possible solutions:
  - Conditional write
  - Atomic counter

Alternative Data Models – Toward NoSQL (continued …)

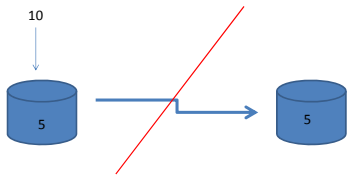
CAP Theorem*

Consistency – Availability – Partition Tolerance
Distributed systems requiring always-on, highly available operation cannot guarantee the illusion of coherent, consistent single-system operation in the presence of network partitions, which cut communication between active servers.

*Bailis, P. and Ghodsi, A. (2013). Eventual consistency today. Comm. of the ACM, 58(5), 55-63.

Eventual Consistency

If no additional updates are made to a given item, all reads to that item will return the same value



Two Properties of Distributed Systems

- Safety property – A guarantee that "nothing bad happens."
- Liveness property – A guarantee that "something good eventually happens."

Key Questions

- How eventual is eventual consistency? (How long will it take for writes to become visible by readers?)
- How should one program under eventual consistency?
- Is it possible to provide strong guarantees than eventual consistency without losing its benefits?

How eventual is eventual consistency?

- Given a workload measure the time to propagate writes
- Research shows that eventual consistency is often strongly consistent …

How to program eventual consistency?

- You assume that the value for a given data item is correct
  - BUT, if you find out later that it is not you compensate to correct things

ATM Example

Transaction – T1 in USA	**Transaction – T2 – China**
Time1 – Balance $500.00	Balance $500.00
Time2 – Withdraw $400.00	Withdraw $400.00
Time3 – Balance $100.00	Balance $100.00

ATM Example

Transaction – T1 in USA	**Transaction – T2 – China**
Time1 – Balance $500.00	Balance $500.00
Time2 – Withdraw $400.00	Withdraw $400.00
Time3 – Balance $100.00	Balance $100.00

Oh no! But who wins in the end?

How to Handle this?

- A socio-technical system which

  Maximze B – RC

  B – Benefit of weak consistency (availability and low latency)
  R – Rate of anomalies
  C – Cost of each anomaly

Application-Specific

- Too many overdrafts might cause customers to leave the system
- Propagating status updates slowly might cause users to leave a social network site
- Application logic for handling the inconsistencies needs to be written and may be difficult

Is it possible to provide strong guarantees than eventual consistency without losing its benefits?

- Probably coming soon …

Some System Assumptions

- Query model
  - Simple read and write
  - State in binary objects identified by a key
- ACID (Atomicity, Consistency, Isolation, Durability)
  - Full ACID properties are given up
- Efficiency

Data Model

- No Schema!

```
ProductCatalog ( Id, ... )
```

```
{
    Id = 101
    ProductName = "Book 101 Title"
    ISBN = "111-1111111111"
    Authors = [ "Author 1", "Author 2" ]
    Price = -2
    Dimensions = "8.5 x 11.0 x 0.5"
    PageCount = 500
    InPublication = 1
    ProductCategory = "Book"
}
```

```
{
    Id = 201
    ProductName = "18-Bicycle 201"
    Description = "201 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 100
    Gender = "M"
    Color = [ "Red", "Black" ]
    ProductCategory = "Bike"
}
```

Primary Key

1. Hash Type Primary Key

2. Hash and Range Type Primary Key

For the power data what would be the primary keys?

Data Types

- Scalar data types
  - Number
  - String
  - Binary
- Multi-valued types
  - String set
  - Number set
  - Binary set

In Summary

1. Tables
   - Create / update / delete tables
2. Items
   - Add / update / delete items
3. Attributes

Query Read and Consistency

- Multiple items of each item are kept
- Two kinds of reads:
  1. Eventually consistent reads
  2. Strongly consistent reads

Updates and Concurrency Control

- "Lost update" problem
- Two possible solutions:
  - Conditional write
  - Atomic counter