Introduction to Data Management

Lecture #17

SQL  NoSQL (©)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

• Homework info:
  – HW #7: Due tomorrow (6 PM).
  – HW #8 is the end (“NoSQL”):
    • Due a week from Friday (6 PM).
    • Late penalty: 10 pts/day (BUT JUST ONE DAY).
• NoSQL lecture plans:
  – Today/Tuesday: NoSQL & Big Data (a la AsterixDB)
    • Not in book: See paper linked to wiki syllabus!
    • Also see docs on the Apache AsterixDB site.
  – Watch the Stanford online lecture material!
    • Watch both of the JSON video lectures.
    • Be sure to take the quiz at the end!
Our Plan for NoSQL + AsterixDB

• The pre-relational era
• The relational DB era
• Beyond rows and columns?
  1. The object-oriented DB era
  2. The object-relational DB era
  3. The XML DB era
  4. The NoSQL DB era*  (*watch Stanford material too...!)

• Reflections, and then ...  AsterixDB!

The Birth of Today’s DBMS Field

• In the beginning was the Word, and the Word was with Codd, and the Word was Codd...
  – 1970 CACM paper: “A relational model of data for large shared data banks”
• Many refer to this as the first generation of (real?) database management systems
This is a SQL/NoSQL History Talk

• The pre-relational era
• The relational DB era
• Beyond rows and columns?
  1. The object-oriented DB era
  2. The object-relational DB era
  3. The XML DB era
  4. The NoSQL DB era
• Reflections & challenges

The First Decade B.C.

• The need for a data management library, or a database management system, had actually been well recognized
  – Hierarchical DB systems (e.g., IMS from IBM)
  – Network DB systems (most notably CODASYL)
• These systems provided *navigational* APIs
  – Systems provided files, records, pointers, indexes
  – Programmers had to (carefully!) scan or search for records, follow parent/child structures or pointers, and maintain code when anything physical changed
The First Decade B.C. (cont.)

Order (id, custName, custCity, total)

Product (sku, name, listPrice, size, power)

Item (ino, qty, price)

Enter the Relational DB Era

Order (id, custName, custCity, total)

Product (sku, name, listPrice, size, power)

Item (order-id, ino, product-sku, qty, price)

- Be sure to notice that
  - Everything’s now (logical) rows and columns
  - The world is flat; columns are atomic (1NF)
  - Data is now connected via keys (foreign/primary)
As the Relational Era Unfolded

• The Spartan simplicity of the relational data model made it possible to start tackling the opportunities and challenges of a logical data model
  – Declarative queries (Rel Alg/Calc, Quel, QBE, SQL, …)
  – Transparent indexing (physical data independence)
  – Query optimization and execution
  – Views, constraints, referential integrity, security, …
  – Scalable (shared-nothing) parallel processing
• Today’s multi-$B industry was slowly born
  – Commercial adoption took ~10-15 years
  – Parallel DB systems took ~5 more years

Enter the Object-Oriented DB Era

• Notice that:
  – Data model contains objects and pointers (OIDs)
  – The world is no longer flat – the Order and Product schemas now have set(Item) and Product in them, respectively
What OODBs Sought to Offer

- Motivated largely by late 1980’s CAx applications (e.g., mechanical CAD, VLSI CAD, software CAD, ...)
  - Rich schemas with inheritance, complex objects, object identity, references, ...
  - Methods (“behavior”) as well as data in the DBMS
  - Tight bindings with (OO) programming languages
  - Fast navigation, some declarative querying
- Ex: Gemstone, Ontos, Objectivity, Versant, Object Design, O2, also DASDBS (sort of)

Why OODBs “Fell Flat”

- Too soon for another (radical) DB technology
  - Also technically immature relative to RDBMSs
- Tight PL bindings were a double-edged sword
  - Data shared, outlives programming languages
  - Bindings led to significant system heterogeneity
  - Also made schema evolution a major challenge
- Systems “overfitted” in some dimensions
  - Inheritance, version management, ...
  - Focused on thick clients (e.g., CAD workstations)
Enter the Object-Relational DB Era

Order (id, customer, total)

<table>
<thead>
<tr>
<th>Order</th>
<th>Customer</th>
<th>City</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Fred</td>
<td>LA</td>
<td>25.97</td>
</tr>
</tbody>
</table>

Item (order-id, ino, product-sku, qty, price)

<table>
<thead>
<tr>
<th>Order-id</th>
<th>Ino</th>
<th>Product-sku</th>
<th>Qty</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>(123)</td>
<td>1</td>
<td>(401)</td>
<td>2</td>
<td>9.99</td>
</tr>
<tr>
<td>(123)</td>
<td>2</td>
<td>(544)</td>
<td>1</td>
<td>3.99</td>
</tr>
</tbody>
</table>

Product (sku, name, listPrice)

<table>
<thead>
<tr>
<th>ClothingProduct (size)</th>
<th>ElectricProduct (power)</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Garfield T-Shirt</td>
</tr>
<tr>
<td>544</td>
<td>USB Charger</td>
</tr>
</tbody>
</table>

• Be sure to notice:
  – “One size fits all!”
  – UDTs/UDFs, table hierarchies, references, ...
  – But the world got flatter again...
  (Timing lagged OODBs by just a few years)

What O-R DBs Sought to Offer

• Motivated by newly emerging application opportunities (multimedia, spatial, text, ...)
  – User-defined functions (UDTs/UDFs) & aggregates
  – Data blades (UDTs/UDFs + indexing support)
  – OO goodies for tables: row types, references, ...
  – Nested tables (well, at least Oracle added these)

• Back to a model where applications were loosely bound to the DBMS (e.g., ODBC/JDBC)
  • Ex: ADT-Ingres, Postgres, Starburst, UniSQL, Illustra, DB2, Oracle
Why O-R DBs “Fell Flat”

- Significant differences across DB vendors
  - SQL standardization lagged somewhat
  - Didn’t include details of UDT/UDF extensions
  - Tough to extend the innards (for indexing)
- Application issues (and multiple platforms)
  - Least common denominator vs. coolest features
  - Tools (e.g., DB design tools, ORM layers, ...)
- Also still probably a bit too much too soon
  - IT departments still rolling in RDBMSs and creating relational data warehouses

Then Came the XML DB Era

```xml
<Order id="123">
  <Customer>
    <custName>Fred</custName>
    <custCity>LA</custCity>
  </Customer>
  <total>25.97</total>
  <Items>
    <Item ino="1">
      <product-sku>401</product-sku>
      <qty>2</qty>
      <price>9.99</price>
    </Item>
    <Item ino="2">
      <product-sku>544</product-sku>
      <qty>1</qty>
      <price>3.99</price>
    </Item>
  </Items>
</Order>

<Product sku="401">
  <name>Garfield T-Shirt</name>
  <listPrice>9.99</listPrice>
  <size>XL</size>
</Product>

<Product sku="544">
  <name>USB Charger</name>
  <listPrice>5.99</listPrice>
  <power>115V</power>
</Product>
```

Note that
- The world’s less flat again
- We’re now in the 2000’s
What XML DBs Sought to Offer

• One <flexible/> data model fits all (XML)
  – Origins in document markup (SGML)
  – Nested data
  – Schema variety/optionality
• New declarative query language (XQuery)
  – Designed both for querying and transformation
  – Early standardization effort (W3C)
• Two different DB-related use cases, in reality
  – *Data storage*: Lore (pre-XML), Natix, Timber, Ipedo, MarkLogic, BaseX; also DB2, Oracle, SQL Server
  – *Data integration*: Nimble Technology, BEA Liquid Data (from Enosys), BEA AquaLogic Data Services Platform

Why XML DBs “Fell Flat” Too

• Document-centric origins (vs. data use cases) of XML Schema and XQuery made a mess of things
  – W3C XPATH legacy (😊)
  – Document identity, document order, ...
  – Attributes vs. elements, nulls, ...
  – Mixed content (overkill for non-document data)
• Two other external trends also played a role
  – SOA and Web services came but then went
  – JSON (and RESTful services) appeared on the scene
• *Note*: Likely still an important niche market...
Now the NoSQL DB Era?

- **Not** from the DB world
  - Distributed systems folks
  - Also various startups
- From caches → K/V use cases
  - Needed massive scale-out
  - OLTP (vs. parallel DB) apps
  - Simple, low-latency API
  - Need a key K, but want no schema for V
  - Record-level atomicity, replica consistency varies
- In the context of this talk, NoSQL does **not** mean
  - Hadoop (or SQL on Hadoop)
  - Graph databases or graph analytics platforms

---

NoSQL Data (**JSON-based**)  

**Collection(Order)**

```
{  
    "id": "123",  
    "Customer":  
    {  
        "custName": "Fred",  
        "custCity": "LA"  
    },  
    "total": 25.97,  
    "Items": [  
        {  
            "product-sku": 401,  
            "qty": 2,  
            "price": 9.99  
        },  
        {  
            "product-sku": 544,  
            "qty": 1,  
            "price": 3.99  
        }  
    ]
}
```

**Collection(Product)**

```
{  
    "sku": 401,  
    "name": "Garfield T-Shirt",  
    "listPrice": 9.99,  
    "size": "XL"  
}

{  
    "sku": 544,  
    "name": "USB Charger",  
    "listPrice": 5.99,  
    "power": "115V"  
}
```

Note that
- The world’s not flat, but it’s less `<messy/>`
- We’re now in the 2010’s, timing-wise
Current NoSQL Trends

• Popular examples: MongoDB, Couchbase
• Coveting the benefits of many DB goodies
  – Secondary indexing and non-key access
  – Declarative queries
  – Aggregates and now (initially small) joins
• Seem to be heading towards...
  – BDMS (think scalable, OLTP-aimed, parallel DBMS)
  – Declarative queries and query optimization, but applied to schema-less data
  – Return of (some, optional!) schema information

Our Example: Apache AsterixDB

ASTERIX Goal:
To ingest, digest, persist, index, manage, query, analyze, and publish massive quantities of semistructured information...

http://asterixdb.apache.org/
Big Data / Web Warehousing

So what went on – and why?

What’s going on right now?

Also: Today’s Big Data Tangle

Spark

SQL

mongoDB

(Pig)

HIVE

Cassandra

MySQL

HBASE

Hadoop
AsterixDB: “One Size Fits a Bunch”

Semistructured Data Management

BDMS Desiderata:
• Flexible data model
• Efficient runtime
• Full query capability
• Cost proportional to task at hand (1)
• Designed for continuous data ingestion
• Support today’s “Big Data data types”
  •
  •

Parallel Database Systems

Data-Intensive Computing

Project Goals

• Build a new Big Data Management System (BDMS)
  – Run on large commodity clusters
  – Handle mass quantities of semistructured data
  – Openly layered, for selective reuse by others
  – Share with the community via open source (ASF)

• Conduct scalable information systems research, e.g.,
  – Large-scale query processing and workload management
  – Highly scalable storage and index management
  – Fuzzy matching, spatial data, date/time data (all in parallel)
  – Novel support for “fast data” (both in and out)

• Train next generation of “Big Data” graduates
create dataverse TinySocial;
use dataverse TinySocial;

create type MugshotUserType as {
    id: int32,
    alias: string,
    name: string,
    userSince: datetime,
    address: {
        street: string,
        city: string,
        state: string,
        zip: string,
        country: string
    },
    friendIds: {[int32]},
    employment: [EmploymentType]
};

create type EmploymentType as open {
    organizationName: string,
    startDate: date,
    endDate: date?
};

create dataset MugshotUsers(MugshotUserType)
    primary key id;

Highlights include:
• JSON++ based data model
• Rich type support (spatial, temporal, ...)
• Records, lists, bags
• Open vs. closed types
**ASTERIX Data Model (ADM)**

```plaintext
create dataverse TinySocial;
use dataverse TinySocial;
create type MugshotUserType as {
id: int32
};
create type MugshotMessageType as closed {
    messageId: int32,
    authorId: int32,
    timestamp: datetime,
    inResponseTo: int32?,
    senderLocation: point?,
    tags: [ string ],
    message: string
};
create dataverse TinySocial;
use dataverse TinySocial;
create type MugshotUserType as {
id: int32
};
create type EmploymentType as open {
    organizationName: string,
    startDate: date,
    endDate: date?
};
create dataset MugshotUsers(MugshotUserType)
    primary key id;
create dataset
    MugshotMessages(MugshotMessageType)
    primary key messageId;
```

**Highlights include:**
- JSON++ based data model
- Rich type support (spatial, temporal, ...)
- Records, lists, bags
- *Open vs. closed types*

---

**Ex: MugshotUsers Data**

```plaintext
{ "id": 1, "alias": "Margarita", "name": "MargaritaStoddard", "address":{
    "street": "234 Thomas Ave", "city": "San Hugo", "zip": "98765",
    "state": "CA", "country": "USA" },
    "userSince": datetime("2012-08-20T10:10:00"),
    "friendIds": [2, 3, 6, 10 ], "employment": [ {
    "organizationName": "Codetechno", "startDate": date("2006-08-06") } ],

{ "id": 2, "alias": "Isbel", "name": "IsbelDull", "address":{
    "street": "345 James Ave", "city": "San Hugo", "zip": "98765",
    "state": "CA", "country": "USA" },
    "userSince": datetime("2011-01-22T10:10:00"),
    "friendIds": [1, 4 ], "employment": [ {
    "organizationName": "Hexviafind", "startDate": date("2010-04-27") } ],

{ "id": 3, "alias": "Emory", "name": "EmoryUnk", "address":{
    "street": "456 Jose Ave", "city": "San Hugo", "zip": "98765",
    "state": "CA", "country": "USA" },
    "userSince": datetime("2012-07-10T10:10:00"),
    "friendIds": [1, 5, 8, 9 ], "employment": [ {
    "organizationName": "geomedia",
    "startDate": date("2010-06-17"), "endDate": date("2010-01-26") } ],
```

...
Other DDL Features

```sql
create index msUserSinceIdx on MugshotUsers(userSince);
create index msTimestampIdx on MugshotMessages(timestamp);
create index msAuthorIdx on MugshotMessages(authorId) type btree;
create index msSenderLocIndex on MugshotMessages(senderLocation) type rtree;
create index msMessageIdx on MugshotMessages(message) type keyword;
```

// --------------------- and also ------------------------------------

```sql
create type AccessLogType as closed
{
  ip: string,
  time: string,
  user: string,
  verb: string,
  `path`: string,
  stat: int32,
  size: int32
};
create external dataset AccessLog[AccessLogType] using localfs
(redirect="hostname://{path}",
(format="delimited-text"),
("delimiter"="|"));
```

```sql
create feed mySocketFeed using socket_adaptor
(redirects="{address}:{port}",
("addressType"="IP"),
("type-name"="MugshotMessageType"),
("format"="adm"));
connect feed mySocketFeed to dataset MugshotMessages;
```

**External data highlights:**
- Equal opportunity access
- “Keep everything!”
- Data ingestion, *not* streams

ASTERIX Queries (SQL++ or AQL)

**Q1:** List the user name and messages sent by those users who joined the Mugshot social network in a certain time window:

```sql
select user.name as uname,
    (select value msg.message
    from MugshotMessages msg
    where msg.authorId = user.id) as messages
from MugshotUsers user
where user.userSince >= datetime('2010-07-22T00:00:00')
and user.userSince <= datetime('2012-07-29T23:59:59');
```

```json
{
  "uname": "IsbelDull",
  "messages": ["like samsung the plan is amazing",
               "like t-mobile its platform is mind-blowing"]
}
{
  "uname": "EmoryUnk",
  "messages": ["love sprint its shortcut-menu is awesome:")",
               ...
            ]
```


SQL++ (cont.)

- Q2: Identify active users and group/count them by country:

  ```sql
  with endTime as current_datetime(),
       startTime as endTime - duration("P30D")
  select user.address.country as country, count(users) as activeUsers
  from MugshotUsers user
  where some logrec in AccessLog satisfies
    user.alias = logrec.user
    and datetime(logrec.time) >= startTime
    and datetime(logrec.time) <= endTime
  group by user.address.country;
  ```

  **SQL++ highlights:**
  - Lots of other features (see website!)
  - Spatial predicates and aggregation
  - Set-similarity ("fuzzy") matching

Updates and Transactions

- Q3: Add a new user to Mugshot.com:

  ```sql
  insert into MugshotUsers
  ( {
    "id": 11, "alias": "John",
    "name": "JohnDoe",
    "userSince": datetime("2012-08-20T10:10:00.000Z"),
    "address": { "street": "789 Jane St",
      "city": "San Harry", "state": "CA",
      "zip": "98767", "country": "USA" },
    "friendIds": [{ 5, 9, 11 }],
    "employment": [ { "organizationName": "Kongreen", "startDate": date("2009-08-11") } ] });
  ```

- Key-value store-like transactions (w/record level atomicity)
- Insert, delete, and upsert ops; index-consistent
- 2PL concurrency
- WAL no-steal, no-force with LSM shadowing
AsterixDB Cluster Overview

Data Loads and Feeds → MD Node Controller
AQL queries and results → Node Controller
Data publishing → Node Controller

asterixDB

ASTERIX Software Stack

AQL → HiveQL → XQuery → Pregel Job → Pregelix → M/R Layer → M/R Job → Hyracks Job
Hyracks Data-Parallel Platform

Hadoop M/R Job

Apache VXQuery

Algebricks Algebra Layer

Hivesterix
A Peek at Performance

<table>
<thead>
<tr>
<th>Users</th>
<th>Messages</th>
<th>Tweets</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>120</td>
<td>330</td>
</tr>
<tr>
<td>360</td>
<td>240</td>
<td>600</td>
</tr>
<tr>
<td>290</td>
<td>100</td>
<td>495</td>
</tr>
<tr>
<td>38</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>240</td>
<td>215</td>
<td>478</td>
</tr>
</tbody>
</table>

Table 2: Dataset sizes (in GB)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Asterix (Schema)</th>
<th>Asterix (KeyOnly)</th>
<th>Syst-X</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.091</td>
<td>0.093</td>
<td>0.040</td>
<td>0.035</td>
</tr>
<tr>
<td>20</td>
<td>0.010</td>
<td>0.011</td>
<td>0.026</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Table 4: Average insert time per record (in sec)

A Peek at Performance (cont.)

<table>
<thead>
<tr>
<th>Rec Lookup</th>
<th>Asterix (Schema)</th>
<th>Asterix (KeyOnly)</th>
<th>Syst-X</th>
<th>Hive</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.03</td>
<td>0.03</td>
<td>0.12</td>
<td>(379.11)</td>
<td>0.02</td>
</tr>
<tr>
<td>Range Scan</td>
<td>79.47</td>
<td>148.15</td>
<td>148.33</td>
<td>11717.18</td>
<td>175.84</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.10</td>
<td>0.10</td>
<td>4.90</td>
<td>(11717.18)</td>
<td>0.05</td>
</tr>
<tr>
<td>Sel-Join (Sm)</td>
<td>78.03</td>
<td>96.76</td>
<td>55.01</td>
<td>333.56</td>
<td>66.46</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.51</td>
<td>0.55</td>
<td>2.13</td>
<td>(333.56)</td>
<td>0.62</td>
</tr>
<tr>
<td>Sel-Join (Lg)</td>
<td>79.62</td>
<td>99.73</td>
<td>56.65</td>
<td>350.92</td>
<td>273.52</td>
</tr>
<tr>
<td>— with IX</td>
<td>2.24</td>
<td>2.32</td>
<td>10.59</td>
<td>(350.92)</td>
<td>14.97</td>
</tr>
<tr>
<td>Sel2-Join (Sm)</td>
<td>79.06</td>
<td>97.82</td>
<td>55.81</td>
<td>340.02</td>
<td>66.45</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.50</td>
<td>0.52</td>
<td>2.62</td>
<td>(340.02)</td>
<td>0.61</td>
</tr>
<tr>
<td>Sel2-Join (Lg)</td>
<td>80.18</td>
<td>101.24</td>
<td>56.10</td>
<td>394.11</td>
<td>313.17</td>
</tr>
<tr>
<td>— with IX</td>
<td>2.32</td>
<td>2.32</td>
<td>10.70</td>
<td>(394.11)</td>
<td>15.28</td>
</tr>
<tr>
<td>Agg (Sm)</td>
<td>128.66</td>
<td>232.40</td>
<td>130.64</td>
<td>83.18</td>
<td>400.97</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.16</td>
<td>0.17</td>
<td>0.14</td>
<td>(83.18)</td>
<td>0.19</td>
</tr>
<tr>
<td>Agg (Lg)</td>
<td>128.71</td>
<td>232.41</td>
<td>132.19</td>
<td>94.11</td>
<td>401</td>
</tr>
<tr>
<td>— with IX</td>
<td>5.53</td>
<td>5.55</td>
<td>4.67</td>
<td>(94.11)</td>
<td>8.34</td>
</tr>
<tr>
<td>Grp-Agg (Sm)</td>
<td>130.20</td>
<td>232.77</td>
<td>131.18</td>
<td>127.85</td>
<td>398.27</td>
</tr>
<tr>
<td>— with IX</td>
<td>0.45</td>
<td>0.46</td>
<td>0.17</td>
<td>(127.85)</td>
<td>0.20</td>
</tr>
<tr>
<td>Grp-Agg (Lg)</td>
<td>130.62</td>
<td>234.10</td>
<td>133.02</td>
<td>140.21</td>
<td>400.10</td>
</tr>
<tr>
<td>— with IX</td>
<td>5.96</td>
<td>5.91</td>
<td>4.72</td>
<td>(140.21)</td>
<td>9.03</td>
</tr>
</tbody>
</table>

Table 3: Average query response time (in sec)
Example AsterixDB Use Cases

- Potential use case areas include
  - Social data analytics
  - Cell phone event analytics
  - Behavioral science
  - Education
  - Public health
  - Power usage monitoring
  - Cluster management log analytics
  - ....

Current Status

- 4 year initial NSF project (250+ KLOC), started 2009
- Now officially *Apache AsterixDB*!
  - Semistructured “NoSQL” style data model
  - Declarative parallel queries, inserts, deletes, ...
  - Data storage/indexing (primary & secondary, LSM-based)
  - Internal and external datasets both supported
  - Rich set of data types (including text, time, location)
  - Fuzzy and spatial query processing
  - NoSQL-like transactions (for inserts/deletes)
  - Data feeds and indexes for external datasets
  - ....
For More Information

- Asterix project UCI/UCR research home
  - http://asterix.ics.uci.edu/
- Apache AsterixDB home
  - http://asterixdb.apache.org/
- SQL++ Primer
  - http://asterixdb.apache.org/docs/0.9.1/index.html
- Navigate from CS122a wiki (HW) to get and install it!
  - A few other resources and hints in the HW materials.

QUESTIONS...?