SQL & NoSQL

Phu Nguyen
Principal Software Engineer

11/11/2015

Client Logo

Agenda

Assessment Disciplines

- Class Participation: 30%
- Final Exam: 70%
- Passing Scores: 70%

Duration and Course Timetable

- Course Duration: 6 hrs

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

SQL

Client Logo

History

- Relational Databases – mainstay of business
- Web-based applications: explosion of social media sites (Facebook, Twitter) with large data needs
- Hooking RDBMS to web-based application becomes trouble

Issues with scaling up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

What is NoSQL?

- The Name
  - "SQL" = Traditional relational DBMS
  - "NoSQL" = "No SQL" = Not using traditional relational DBMS
  - Stands for Not Only SQL

What is NoSQL?

- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- Data are replicated to multiple nodes and can be partitioned.
- Focus on solving the problems of scalability and availability
- Uses CAP Theorem/Brewer's Theorem to improve consistency

Defining NoSQL

According to Wikipedia:

In computing, NoSQL (mostly interpreted as "not only SQL") is a broad class of database management systems identified by its non-adherence to the widely used relational database management system model; that is, NoSQL databases are not primarily built on tables, and as a result, generally do not use SQL for data manipulation.

Why NoSQL?

- Schemaless data representation
- Development time
- Performance
- Scalability
- High availability

History of NoSQL

- Started with rational databases
- Growing need to handle larger amounts of data with real-time performance
- First termed as NoRel, “No Relational”, by Carlo Strozzi in1998
- Reintroduced in 2009 as NoSQL, “Not Only SQL”, by Eric Evans to discuss open-source non-rational database systems

List of NoSQL Databases

Document	Key-Value	XML	Column	Graph
MongoDB	Redis	BaseX	BigTable	Neo4J
CouchDB	Membase	eXist	Hadoop / HBase	FlockDB
RavenDB	Voldemort		Cassandra	InfiniteGraph
Terrastore	MemcacheDB		SimpleDB	
			Cloudera	

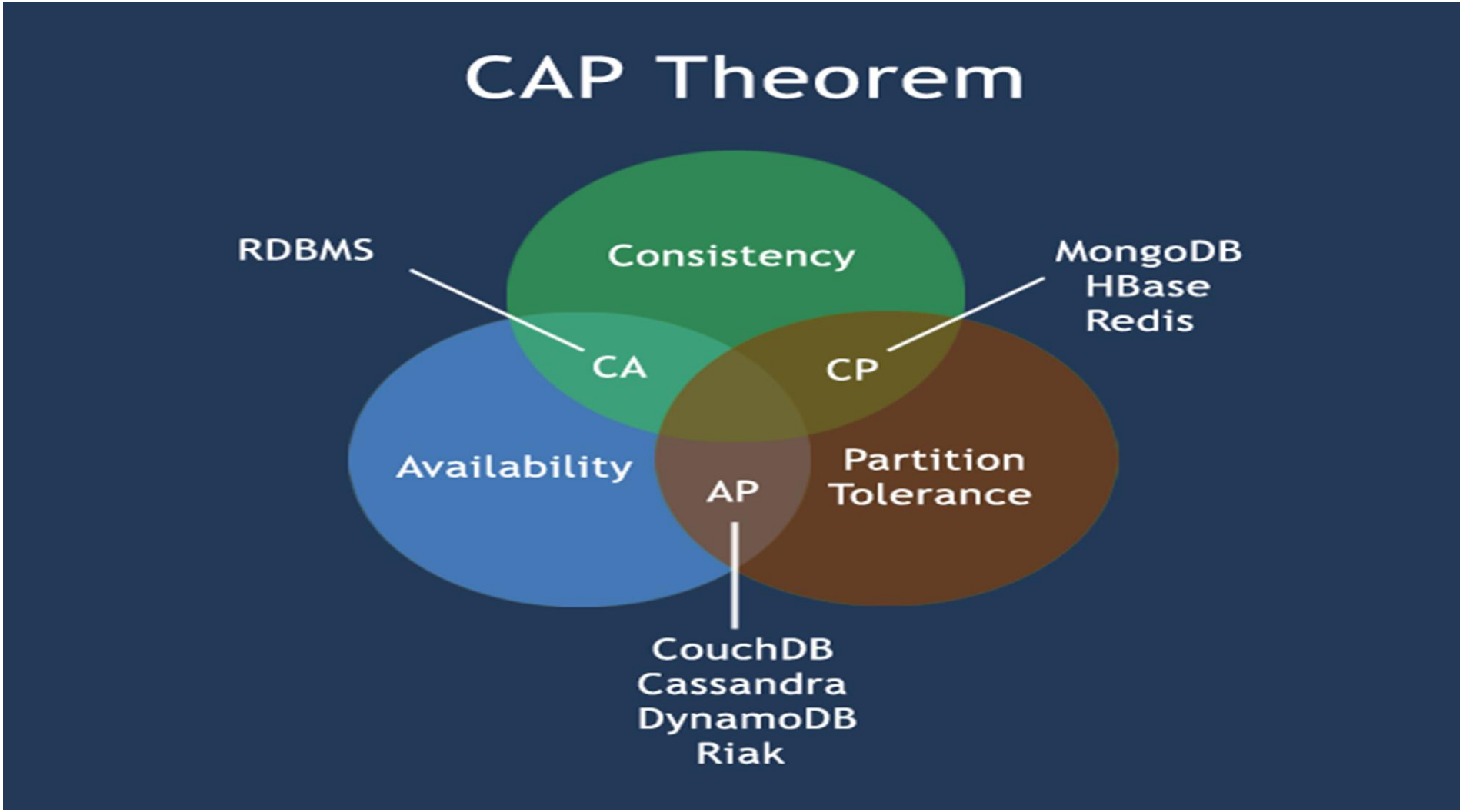
NoSQL Characteristics

- Large data volumes
- Scalable replication and distribution
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem

CAP Theorem

- **Consistency**: All the servers in the system will have the same data so anyone using the system will get the same copy regardless of which server answers their request.
- **Availability**: The system will always respond to a request (even if it's not the latest data or consistent across the system or just a message saying the system isn't working).
- **Partition Tolerance**: The system continues to operate as a whole even if individual servers fail or can't be reached.

CAP Theorem


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

CAP Theorem

- A distributed system can support only two of these three properties.
- To scale out, you have to partition. That leaves either consistency or availability to choose from
  - In almost all cases, you would choose availability over consistency

ACID vs BASE

ACID

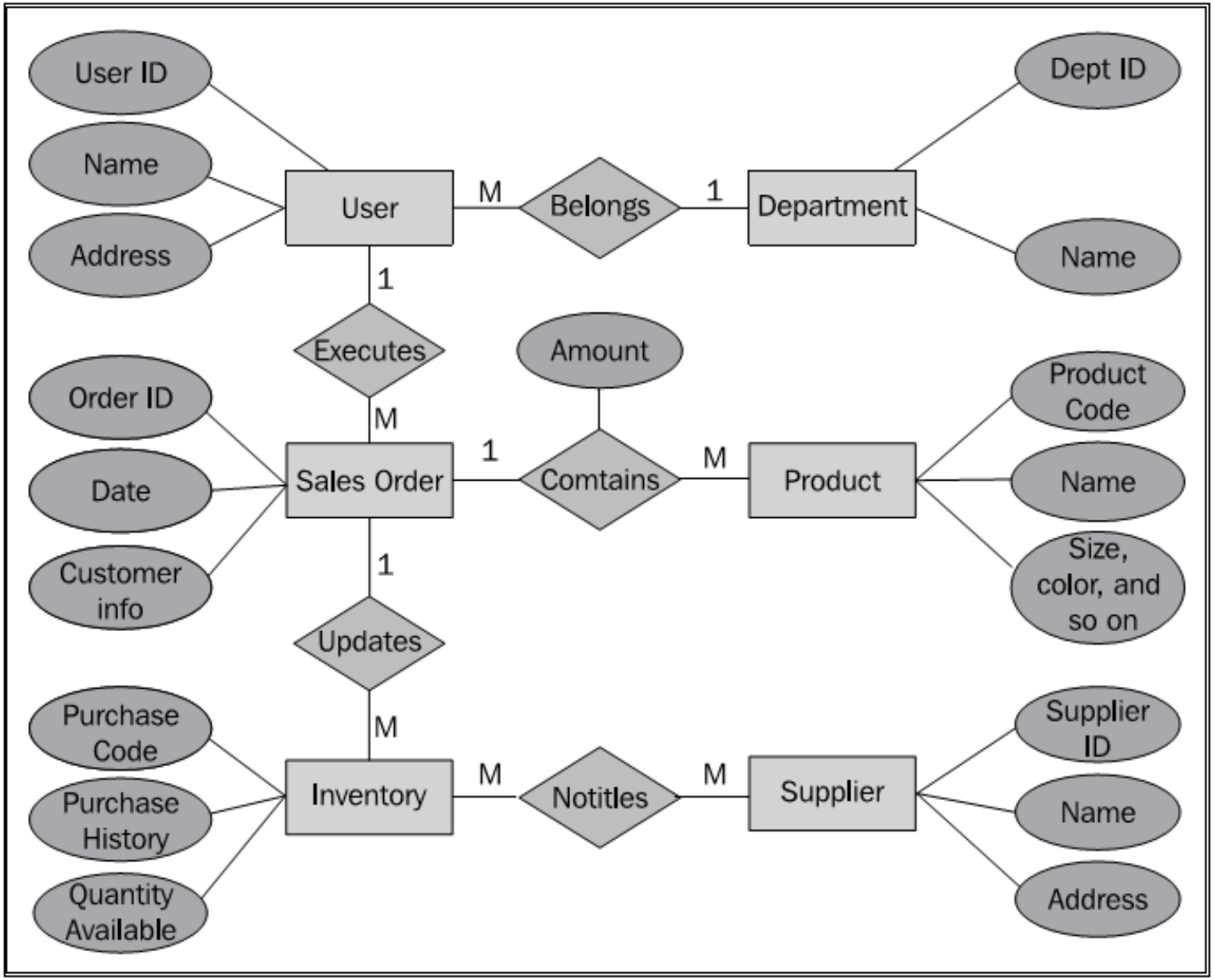
- **Atomicity**: In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are.
- **Consistency**: A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.
- **Isolation**: A transaction in process and not yet committed must remain isolated from any other transaction.
- **Durability**: Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state

BASE

- **Basic availability**: Each request is guaranteed a response—successful or failed execution.
- **Soft state**: The state of the system may change over time, at times without any input (for eventual consistency).
- **Eventual consistency**: The database may be momentarily inconsistent but will be consistent eventually.

RDBMS approach

- Identify actors
- Define models
- Define entities
- Define relationships
- Program database and application
- Iterate


User ID
Name
Address
User
M
Belongs
1
Department
Dept ID
Name
1
Executes
M
Amount
Order ID
Date
Customer info
Sales Order
1
Comtains
M
Product
Product Code
Name
Size, color, and so on
1
Updates
M
Purchase Code
Purchase History
Quantity Available
Inventory
M
Notitles
M
Supplier
Supplier ID
Name
Address

Challenges

- Schema flexibility
- Complex queries
- Data update
- Scalability

NoSQL approach

- **Schema flexibility**: Column-oriented databases or document stores

```
{
   "_id": "98ef65e7-52e4-4466-bacc-3a8dc0c15c79",
   "firstName": "Gaurav",
   "lastName": "Vaish",
   "department": "f0adcbf5-7389-4491-9c42-f39a9d3d4c75",
   "homeAddress": {
      "_id": "fa62fd39-17f8-4a16-80d6-71a5b71d758d",
      "line1": "123, 45th Main"
      "city" : "NoSQLLand",
      "country": "India",
      "zipCode": "123456"
   }
}
```

NoSQL approach

- **Complex queries**: NoSQL databases do not have support for relationships or foreign keys. There are no complex queries. There are no JOIN statements.
- **Data update**: NoSQL solutions offer great synchronization options.
  - MongoDB allows concurrent updates across nodes.
  - The complexity of managing the transaction may be moved out of the database (Multiversion concurrency control (MCC))
- **Scalability**

NoSQL Storage Types

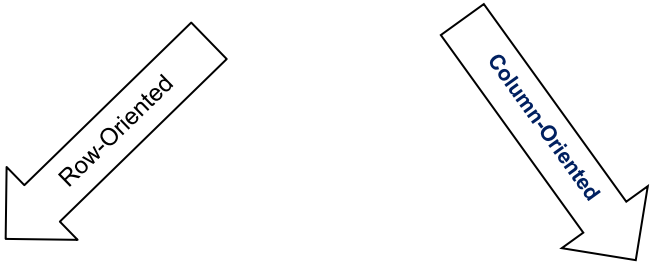
Client Logo

Storage types

- Column-oriented
- Document Store
- Key Value Store
- Graph

Column-oriented databases

EmployeeID	FirstName	LastName	Age	Salary
SM1	Anuj	Sharma	45	10000000
MM2	Anand		34	5000000
T3	Vikas	Gupta	39	7500000
E4	Dinesh	Verma	32	2000000




```
SM1,Anuj,Sharma,45,10000000
MM2,Anand,,34,5000000
T3,Vikas,Gupta,39,7500000
E4,Dinesh,Verma,32,2000000
```

```
SM1,MM2,T3,E4
Anuj,Anand,Vikas,Dinesh
Sharma,,Gupta,Verma,
45,34,39,32
10000000,5000000,7500000,2000000
```

Column-oriented databases

- Apache Hbase
- MonetDB
- Amazon Redshift
- Google Datastore

Document-oriented database

- Can model more complex objects
- Data model: collection of documents
- Document: Using XML, JSON, BSON, or YAML, other semi-structured formats.

Document-oriented database

```
{
  "EmployeeID": "SM1",
  "FirstName" : "Anuj",
  "LastName"  : "Sharma",
  "Age"       : 45,
  "Salary"    : 10000000
}
```

```
<contact>
  <firstname>Bob</firstname>
  <lastname>Smith</lastname>
  <email type=Home>bob.smith@example.com</email>
  <phone type=Cell>(123) 555-0178</phone>
  <phone type=Work>(890) 555-0133</phone>
  <address>
    <type>Home</type>
    <street1>123 Back St.</street1>
    <city>Boys</city>
    <state>AR</state>
    <zip>32225</zip>
    <country>US</country>
  </address>
</contact>
```

Document-oriented database

- MongoDB
- CouchDB
- Apache Cassandra
- Terrastore

Key-value store databases

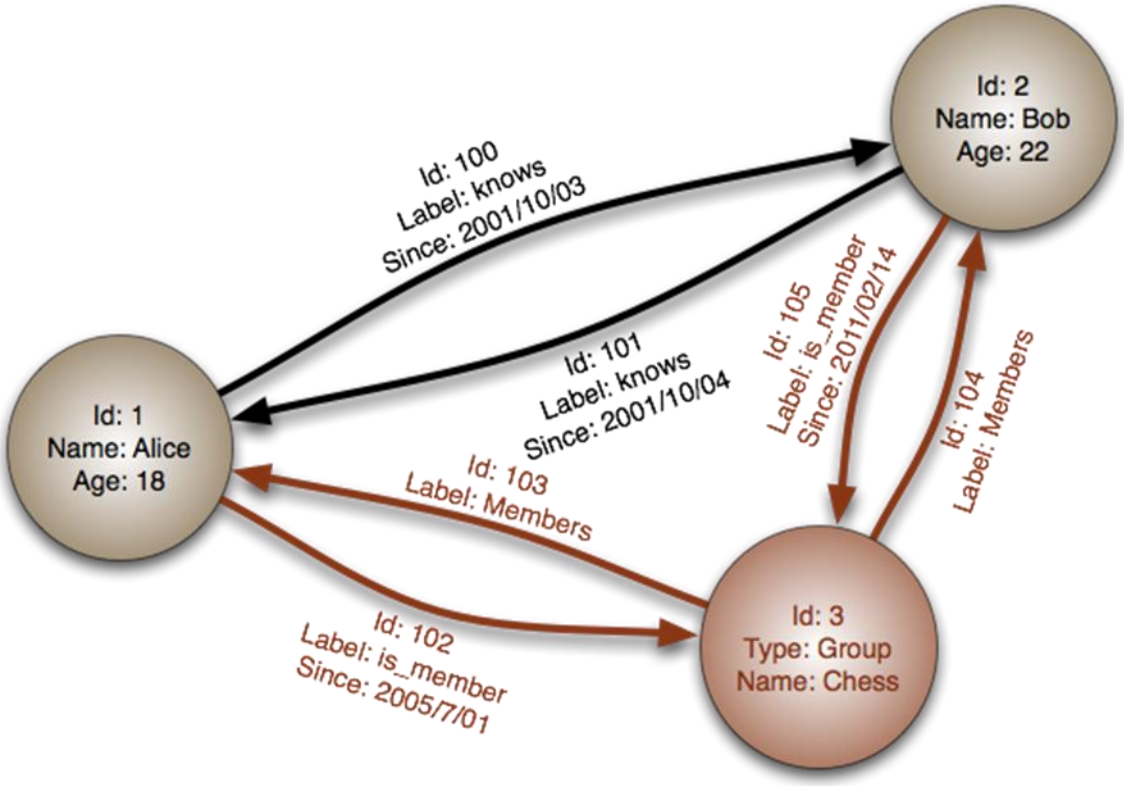
- Closely related to a document store
- Focus on scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon’s dynamo paper
- Data model: (global) collection of Key-value pairs
- Dynamo ring partitioning and replication

Key-value store databases

- Redis
- DynamoDB
- LevelDB
- BerkeleyDB
- Memcached
- Riak

Graph store databases

- A special category of NoSQL databases where relationships are represented as graphs



Graph store databases

- Neo4j
- FlockDB

Engine types

Database	Column oriented	Document store	Key value store	Graph
Amazon SimpleDB	No	No	Yes	No
BaseX	No	Yes	No	No
Cassandra	Yes	Yes	No	No
CouchDB	No	Yes	No	No
Google Datastore	Yes	No	No	No
HBase	Yes	No	No	No
MemcacheDB	No	No	Yes	No
MongoDB	No	Yes	No	No
Neo4j	No	No	No	Yes
Redis	No	Yes	Yes	No

Comparing the models

Feature	Column Oriented	Document Store	Key Value Store	Graph
Table-like schema support (columns)	Yes	No	No	Yes
Complete update/ fetch	Yes	Yes	Yes	Yes
Partial update/ fetch	Yes	Yes	Yes	No
Query/Filter on value	Yes	Yes	No	Yes
Aggregates across rows	Yes	No	No	No
Relationships between entities	No	No	No	Yes
Cross-entity view support	No	Yes	No	No
Batch fetch	Yes	Yes	Yes	Yes
Batch update	Yes	Yes	Yes	No

Benchmark Table

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–Value Store	high	high	high	none	variable (none)
Column-Oriented Store	high	high	moderate	low	minimal
Document-Oriented Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory
Relational Database	variable	variable	low	moderate	relational algebra

MongoDB

Client Logo

What is MongoDB?

- MongoDB is a scalable, high-performance, open source, schema-free, document-oriented database

MongoDB Philosophy

- Create a database that worked with documents rather than rows and that was blindingly fast, massively scalable, and easy to use.
- MongoDB is a perfect fit for resolving, such as analytics and complex data structures.
- There should always be more than one copy of the database (Horizontal scaling).

Who's using MongoDB?

- Education

- Financial Services

Who's using MongoDB?

- Government

CSC

Who's using MongoDB?

- Healthcare

Who's using MongoDB?

- Media and Entertainment

Forbes

SQUARE ENIX

dropevent.com

EUREKA KING

GAMECHANGER

NationalJournal

Who's using MongoDB?

- Retail

Who's using MongoDB?

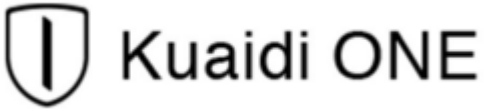
- Technology

FOURSQUARE

Who's using MongoDB?

- Telecommunications

- Travel and Hospitality



MongoDB's features

- Using Document-Oriented Storage (BSON)
  - BSON is an open standard
  - BSON is a binary form of JSON
  - BSON is much easier to traverse and index very quickly
  - BSON is easy and quick to convert BSON to a programming language's native data format
  - BSON also provides some extensions to JSON (store binary data)

MongoDB's features

- Supporting Dynamic Queries
  - Compare to CouchDB
- Indexing Your Documents
- Leveraging Geospatial Indexes
- Profiling Queries
- Updating Information In-Place
  - MongoDB can update the data wherever it happens to be
- Storing Binary Data
  - *GridFS* is MongoDB's solution to storing binary data in the database
- Replicating Data
  - MongoDB provided a safety net with a feature called "*replica sets*"

MongoDB's features

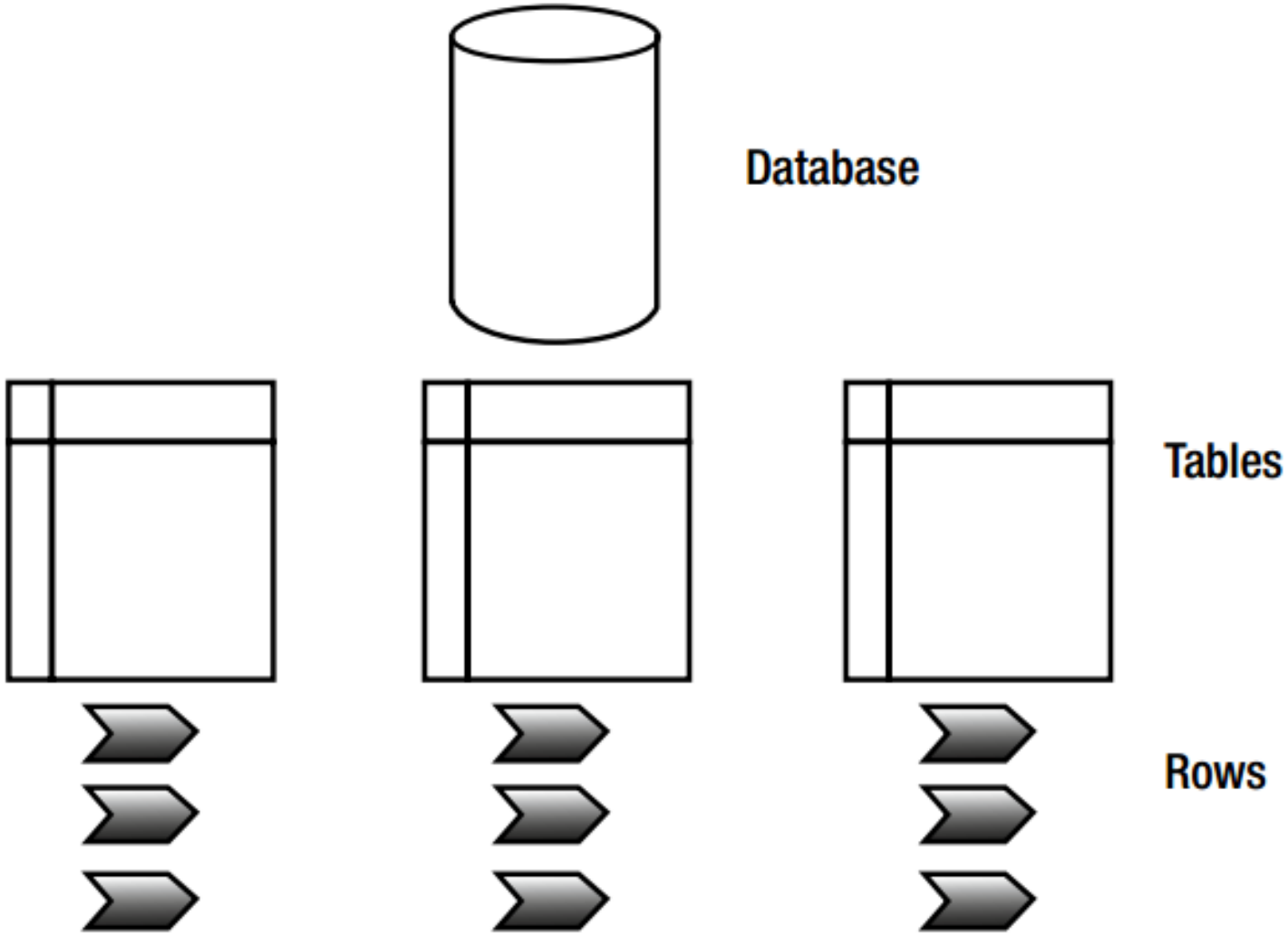
- Implementing Sharding
  - "*auto-sharding*" is one of MongoDB's most significant and oft-used features
- Using Map and Reduce Functions
- The MongoDB Aggregation Framework

Data Model

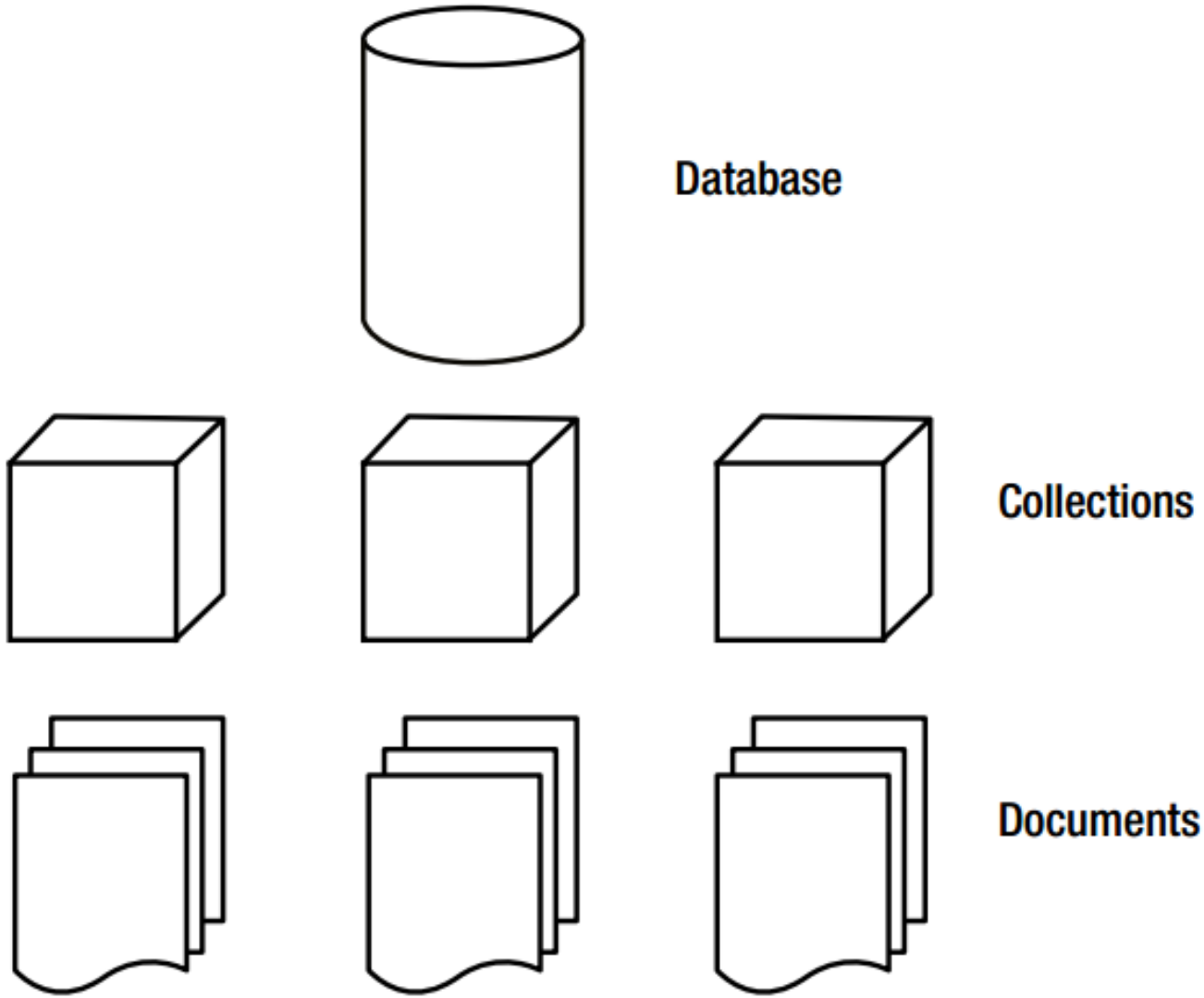
RDBMS vs MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Relational database model



MongoDB database model



Database

- Database is a physical container for collections.
- MongoDB server typically has multiple databases

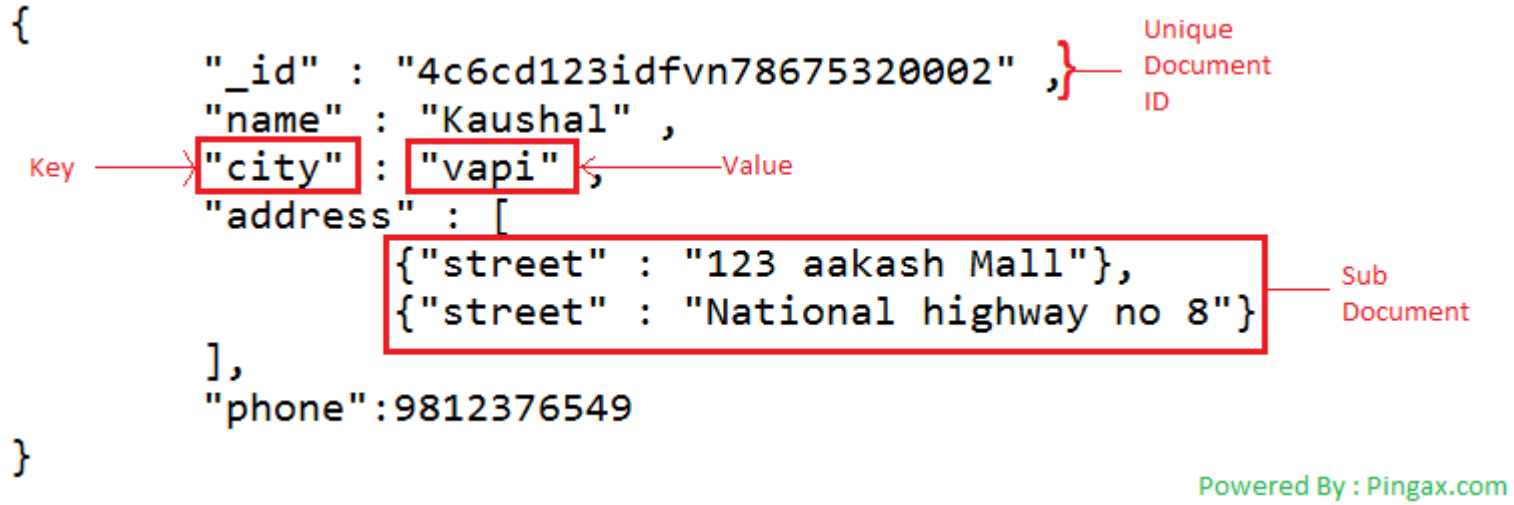
Collection

- Collection is a group of MongoDB documents
- A collection exists within a single database
- Collections do not enforce a schema
- Documents within a collection can have different fields
- All documents in a collection are of similar or related purpose

Document

- A document is a set of key-value pairs
- Documents have dynamic schema

Sample Data


{
"_id" : "4c6cd123idfvn78675320002" ,
Unique Document ID
"name" : "Kaushal" ,
Key
"city" : "vapi" ,
Value
"address" : [
{"street" : "123 aakash Mall"},
{"street" : "National highway no 8"}
Sub Document
],
"phone":9812376549
}
Powered By : Pingax.com

Design schema in MongoDB

- Combine objects into one document if you will use them together.
- Accept duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Do complex aggregation in the schema

CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Inserting a document

- SQL
  - `INSERT INTO t (fn, ln) VALUES ('John', 'Doe')`

- MongoDB
  - `db.t.insert({fn:'John', ln: 'Doe'})`

Updating a document

- SQL
  - `UPDATE t SET ln='Smith' WHERE ln='Doe'`

- MongoDB
  - `db.t.update({ln:'Doe'},{$set:{ln:'Smith'}})`

Removing documents

- SQL
  - `DELETE FROM t WHERE fn='John'`

- MongoDB
  - `db.t.remove({fn:'John'})`

Dropping a collection

- SQL
  - `DROP TABLE t`

- MongoDB
  - `db.t.drop()`

Selecting all documents

- SQL
  - `SELECT * FROM t`
  - `SELECT id, name FROM t`

- MongoDB
  - `db.t.find()`
  - `db.t.find({},{name:1})`

Specifying a “WHERE” clause

- SQL
  - `SELECT * FROM t WHERE fn='John‘`
  - `SELECT id, age FROM t WHERE fn='John' AND ln='Smith'`

- MongoDB
  - `db.t.find({ln:'John'})`
  - `db.t.find({fn:'John',ln:'Smith'},{age:1})`

Counting

- SQL
  - `SELECT COUNT(*) FROM t WHERE country='US'`

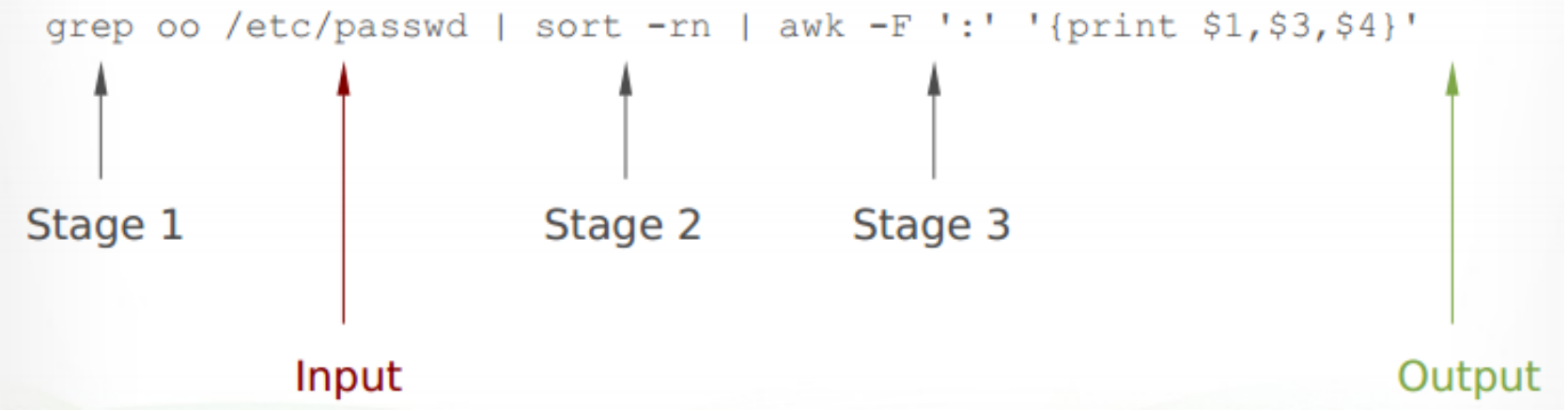
- MongoDB
  - `db.t.count({country:'US'})`

Aggregation

- “Means” GROUP BY, SUM(), …
- 2 ways to aggregate
  - Map-Reduce
  - Aggregation Framework

Aggregation Framework (AF)

- Available from MongoDB 2.2
- Easier and faster than Map-Reduce
- Some limitations
  - AF is perfect for simple cases
  - Map-Reduce can be needed for complex situations
- Documents are modified through pipelines



AF Sample

- SQL
  - `SELECT fn,count(*) AS total FROM people`
    `WHERE fn >= 'M%' GROUP BY fn`

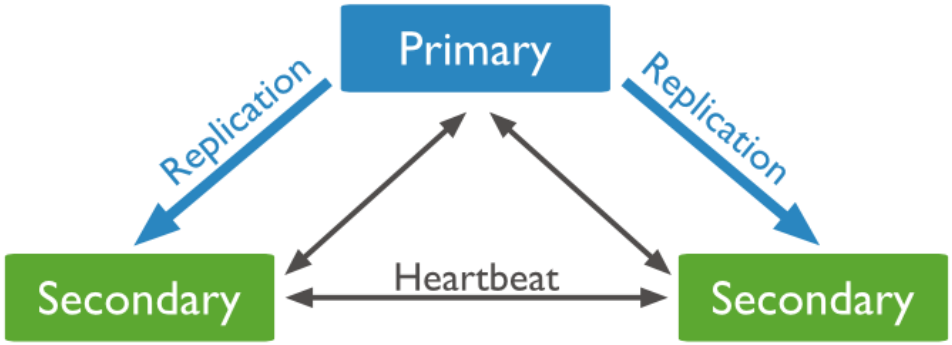
- MongoDB
  - `db.people.aggregate({$match:{fn:{$gte:'M'}}},`
    `{$group: {_id:"$fn",total:{$sum:1}}})`

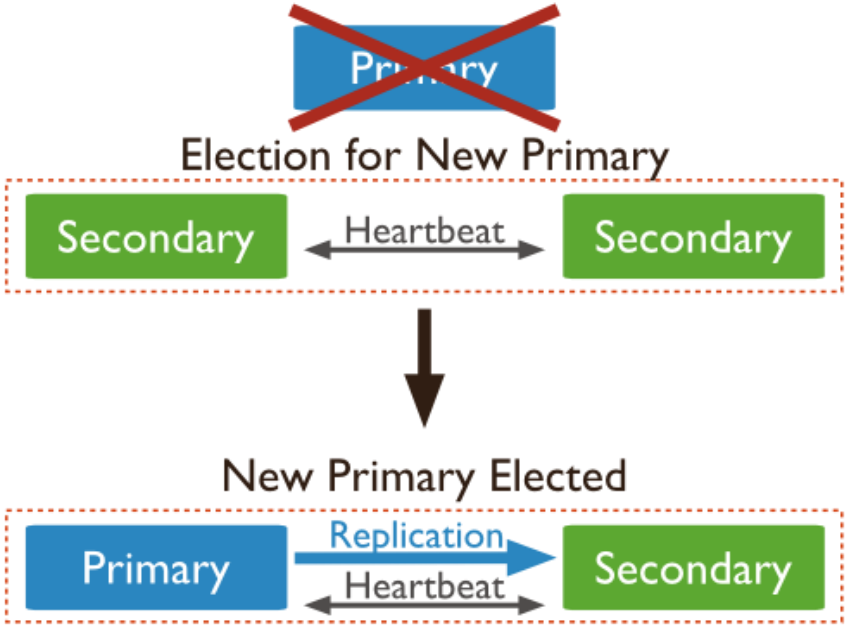
Other operators

- $sort
  - `{$sort: {total: -1}}`
- $limit
  - `{$limit: 10}`
- $skip
  - `{$skip: 100}`
- $unwind
  - Splits a document with an array into multiple documents

MongoDB Replication

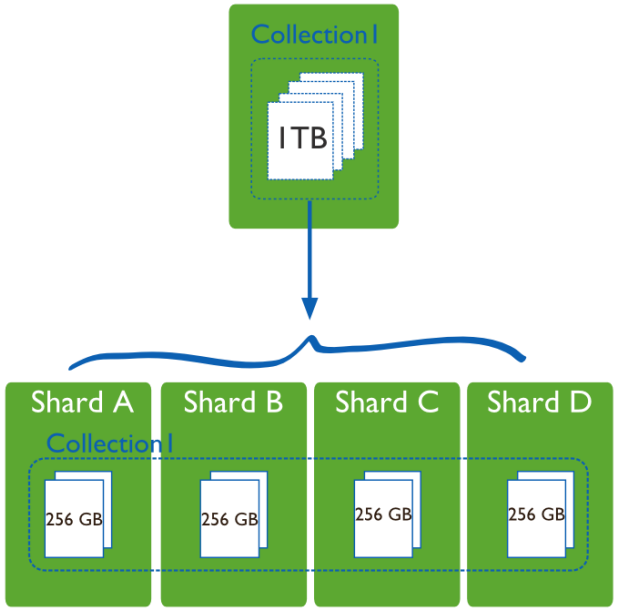
- Replication is the process of synchronizing data across multiple servers.
- Replication provides redundancy and increases data high availability.

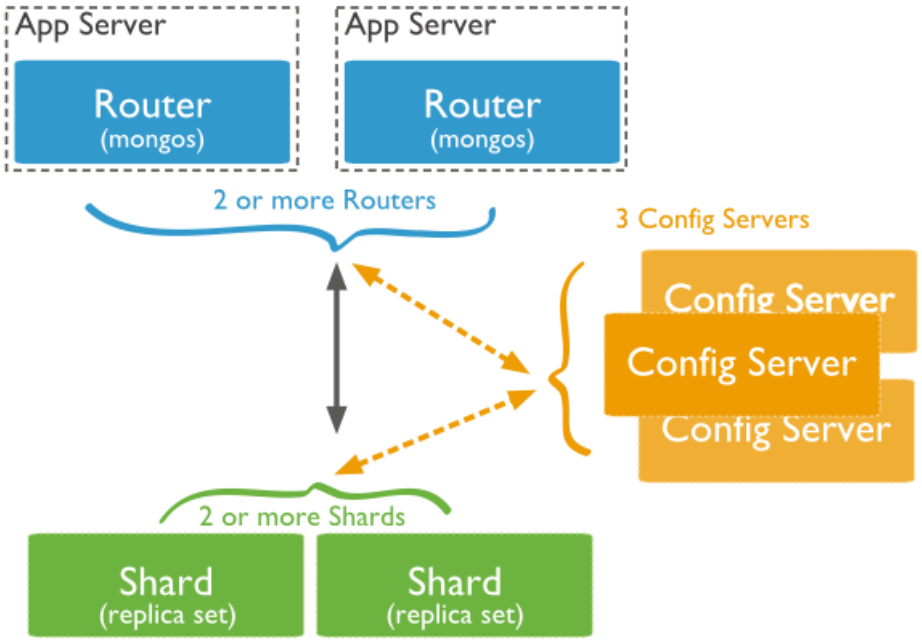




MongoDB Sharding (Horizontal scaling)

- Sharding is a method for storing data across multiple machines
- When HDD, CPU or RAM limits are reached.





Introduction to XML

Client Logo

What is XML?

- XML stands for eXtensible Markup Language.
- A markup language is used to provide information about a document.
- Tags are added to the document to provide the extra information.
- HTML tags tell a browser how to display the document.
- XML tags give a reader some idea what some of the data means.

What is XML Used For?

- XML documents are used to transfer data from one place to another often over the Internet.
- XML subsets are designed for particular applications.
- One is RSS (Rich Site Summary or Really Simple Syndication). It is used to send breaking news bulletins from one web site to another.
- A number of fields have their own subsets. These include chemistry, mathematics, and books publishing.
- Most of these subsets are registered with the W3Consortium and are available for anyone's use.

Advantages of XML

- XML is text (Unicode) based.
  - Takes up less space.
  - Can be transmitted efficiently.
- One XML document can be displayed differently in different media.
  - Html, video, CD, DVD,
  - You only have to change the XML document in order to change all the rest.
- XML documents can be modularized. Parts can be reused.

Example of an HTML Document

```
<html>
  <head><title>Example</title></head.
<body>
  <h1>This is an example of a page.</h1>
  <h2>Some information goes here.</h2>
</body>
</html>
```

Example of an XML Document

```
<?xml version="1.0"/>
<address>
  <name>Alice Lee</name>
  <email>alee@aol.com</email>
  <phone>212-346-1234</phone>
  <birthday>1985-03-22</birthday>
</address>
```

Difference Between HTML and XML

- HTML tags have a fixed meaning and browsers know what it is.
- XML tags are different for different applications, and users know what they mean.
- HTML tags are used for display.
- XML tags are used to describe documents and data.

XML Rules

- Tags are enclosed in angle brackets.
- Tags come in pairs with start-tags and end-tags.
- Tags must be properly nested.
  - **<name><email>…</name></email> is not allowed.**
  - **<name><email>…</email><name> is.**
- Tags that do not have end-tags must be terminated by a ‘/’.
  -
 is an html example.

More XML Rules

- Tags are case sensitive.
  - **<address> is not the same as <Address>**
- XML in any combination of cases is not allowed as part of a tag.
- Tags may not contain ‘<‘ or ‘&’.
- Tags follow Java naming conventions, except that a single colon and other characters are allowed. They must begin with a letter and may not contain white space.
- Documents must have a single *root* tag that begins the document.

Encoding

- XML (like Java) uses Unicode to encode characters.
- Unicode comes in many flavors. The most common one used in the West is UTF-8.
- UTF-8 is a variable length code. Characters are encoded in 1 byte, 2 bytes, or 4 bytes.
- The first 128 characters in Unicode are ASCII.
- In UTF-8, the numbers between 128 and 255 code for some of the more common characters used in western Europe, such as ã, á, å, or ç.
- Two byte codes are used for some characters not listed in the first 256 and some Asian ideographs.
- Four byte codes can handle any ideographs that are left.
- Those using non-western languages should investigate other versions of Unicode.

Well-Formed Documents

- An XML document is said to be well-formed if it follows all the rules.
- An XML parser is used to check that all the rules have been obeyed.
- Recent browsers such as Internet Explorer 5 and Netscape 7 come with XML parsers.
- Parsers are also available for free download over the Internet. One is Xerces, from the Apache open-source project.
- Java 1.4 also supports an open-source parser.

XML Example Revisited

```
<?xml version=“1.0”/>
<address>
  <name>Tran A</name>
  <email>atran@csc.com</email>
  <phone>0989000009</phone>
  <birthday>1981-01-22</birthday>
</address>
```

- Markup for the data aids understanding of its purpose.
- A flat text file is not nearly so clear.

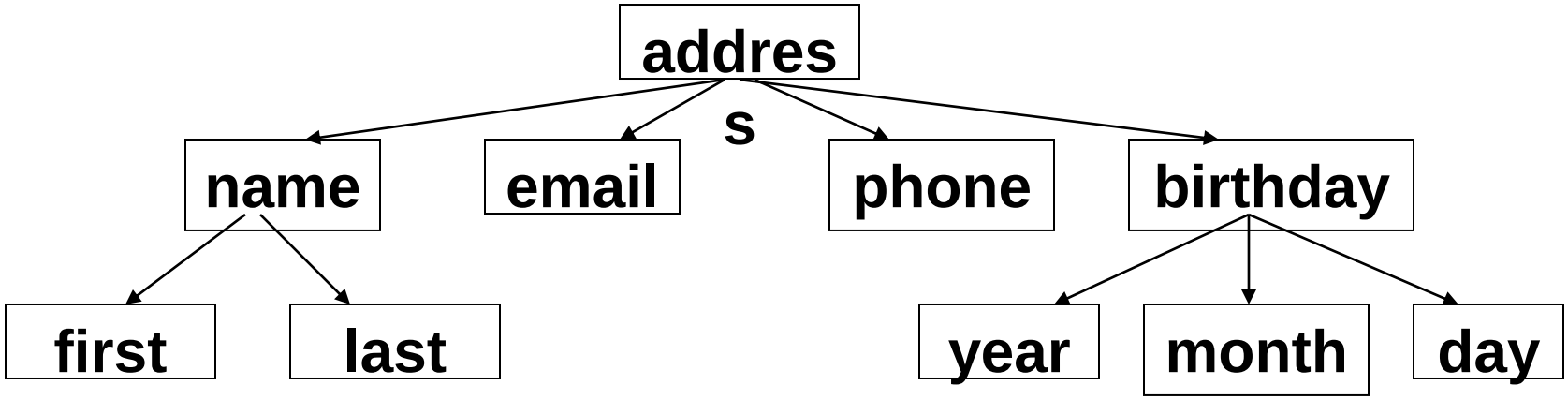
```
Tran A
atran@csc.com
0989000009
1981-01-22
```

- The last line looks like a date, but what is it for?

Expanded Example

```
<?xml version = “1.0” ?>
<address>
  <name>
      <first>A</first>
      <last>Tran</last>
    </name>
    <email>atran@csc.com</email>
    <phone>0989000009</phone>
    <birthday>
      <year>1981</year>
      <month>01</month>
      <day>22</day>
  </birthday>
</address>
```

XML Files are Trees



XML Trees

- An XML document has a single root node.
- The tree is a general ordered tree.
  - A parent node may have any number of children.
  - Child nodes are ordered, and may have siblings.
- Preorder traversals are usually used for getting information out of the tree.

Validity

- A well-formed document has a tree structure and obeys all the XML rules.
- A particular application may add more rules in either a DTD (document type definition) or in a schema.
- Many specialized DTDs and schemas have been created to describe particular areas.
- These range from disseminating news bulletins (RSS) to chemical formulas.
- DTDs were developed first, so they are not as comprehensive as schema.

Document Type Definitions

- A DTD describes the tree structure of a document and something about its data.
- There are two data types, PCDATA and CDATA.
  - PCDATA is parsed character data.
  - CDATA is character data, not usually parsed.
- A DTD determines how many times a node may appear, and how child nodes are ordered.

A sample DTD

<!ELEMENT address (name, email, phone, birthday)>

<!ELEMENT name (first, last)>

<!ELEMENT first (#PCDATA)>

<!ELEMENT last (#PCDATA)>

<!ELEMENT email (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

<!ELEMENT birthday (year, month, day)>

<!ELEMENT year (#PCDATA)>

<!ELEMENT month (#PCDATA)>

<!ELEMENT day (#PCDATA)>

XML Schemas

- XML schemas use XML syntax.
- XML schemas support datatypes.
- XML schemas are extensible.
- XML schemas have more expressive power.

A sample XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="address">
  <xs:complexType>
          <xs:sequence>
                  <xs:element name="name" type="xs:string"/>
                  <xs:element name="email" type="xs:string"/>
                  <xs:element name="phone" type="xs:string"/>
                  <xs:element name="birthday" type="xs:date"/>
          </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Explanation of Example Schema

<?xml version="1.0" encoding="ISO-8859-1" ?>

- **ISO-8859-1, Latin-1, is the same as UTF-8 in the first 128 characters.**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

- **www.w3.org/2001/XMLSchema contains the schema standards.**

<xs:element name="address">

<xs:complexType>

- **This states that address is a complex type element.**

<xs:sequence>

- **This states that the following elements form a sequence and must come in the order shown.**

<xs:element name="name" type="xs:string"/>

- **This says that the element, name, must be a string.**

<xs:element name="birthday" type="xs:date"/>

- **This states that the element, birthday, is a date. Dates are always of the form yyyy-mm-dd.**

XSLT - Extensible Stylesheet Language Transformations

- XSLT is used to transform one xml document into another, often an html document.
- The Transform classes are now part of Java 1.4.
- A program is used that takes as input one xml document and produces as output another.
- If the resulting document is in html, it can be viewed by a web browser.
- This is a good way to display xml data.

A Style Sheet to Transform address.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
          <xsl:template match="address">
                  <html><head><title>Address Book</title></head>
                  <body>
                          <xsl:value-of select="name"/>
                          <br/><xsl:value-of select="email"/>
                          <br/><xsl:value-of select="phone"/>
                          <br/><xsl:value-of select="birthday"/>
                  </body>
                  </html>
          </xsl:template>
  </xsl:stylesheet>
```

The Result of the Transformation

Tran A
atran@csc.com
0989000009
1981-01-22

Parsers

- There are two principal models for parsers.
- SAX – Simple API for XML
  - Uses a call-back method
  - Similar to javax listeners
- DOM – Document Object Model
  - Creates a parse tree
  - Requires a tree traversal
- JDOM
- StAX
- XPATH
- DOM4J

Introduction to JSON

Client Logo

What is JSON?

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is language independent *****

Why JSON?

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

JSON Syntax

- It includes the following
  - Data is represented in name/value pairs.
  - Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
  - Square brackets hold arrays and values are separated by ,(comma).
- JSON supports the following two data structures
  - **Collection of name/value pairs** − This Data Structure is supported by different programming languages.
  - **Ordered list of values** − It includes array, list, vector or sequence etc.

JSON Data Types

- Number
- String
- Boolean
- Array
- Object
- null

JSON Schema

- JSON Schema is a specification for JSON based format for defining the structure of JSON data

  - Describes your existing data format.
  - Clear, human- and machine-readable documentation.
  - Complete structural validation, useful for automated testing.
  - Complete structural validation, validating client-submitted data.

Thank you

Client Logo

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
11/11/2015	1.0	Initial document	Kien Tran	
12/1/2015	1.1	Adding Sections : • Introduction to XML • Introduction to JSON	Phu Nguyen	


CSC
BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING