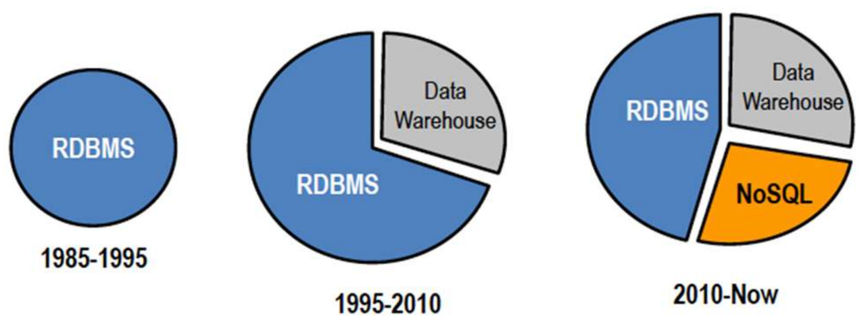
NoSQL data models

Vu Tuyet Trinh

1

Eras of Databases



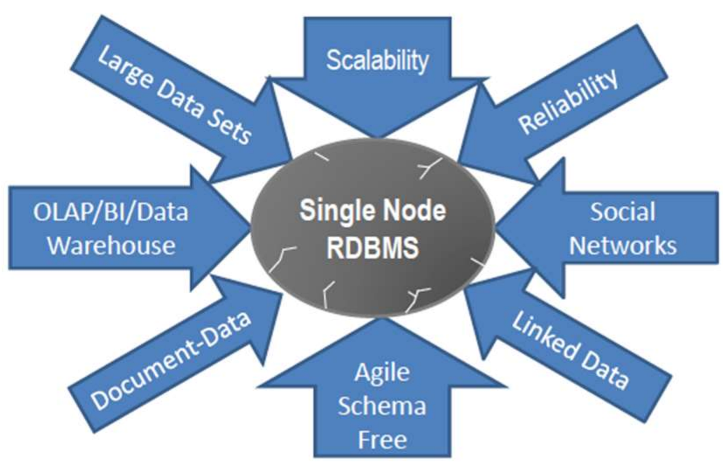
- Why NoSQL?

2

Before NoSQL

3

RDBMS one-size-fits-all-needs



4

ICDE 2005 conference

"One Size Fits All": An Idea Whose Time Has Come and Gone

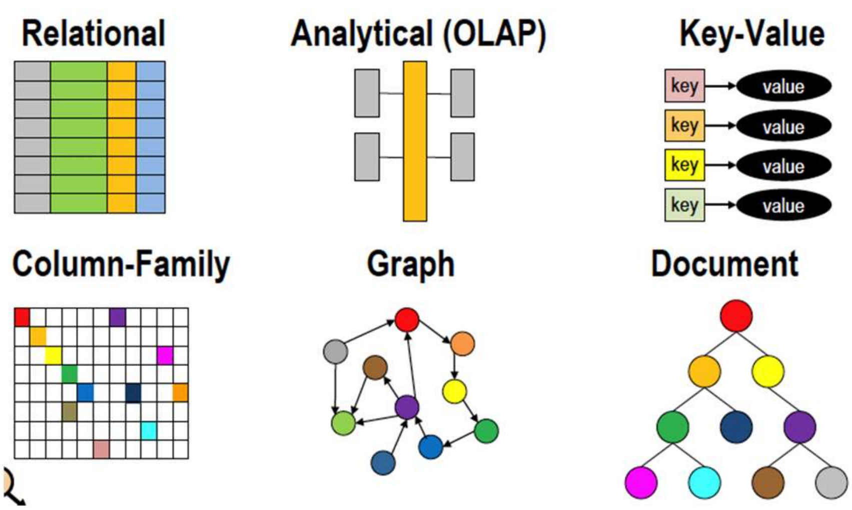
Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

The last 25 years of commercial DBMS development can be summed up in a single phrase: "one size fits all". This phrase refers to the fact that **the traditional DBMS architecture (originally designed and optimized for business data processing) has been used to support many data-centric application**s with widely varying characteristics and requirements. In this paper, we argue that this concept is no longer applicable to the database market, and that the commercial world will fracture into a collection of independent database engines, some of which may be unified by a common front-end parser. We use examples from the stream-processing market and the data-warehouse market to bolster our claims. We also briefly discuss other markets for which the traditional architecture is a poor fit and argue for a critical rethinking of the current factoring of systems services into products.

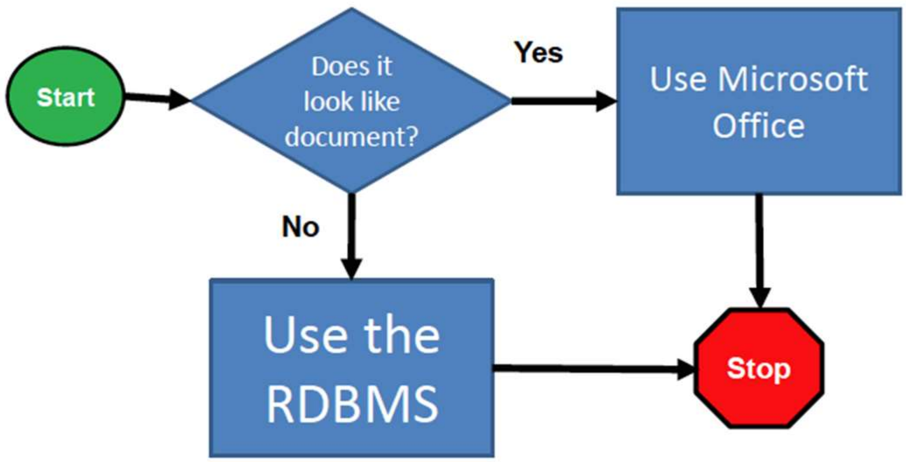
5

After NoSQL



6

RDBMS vs. others

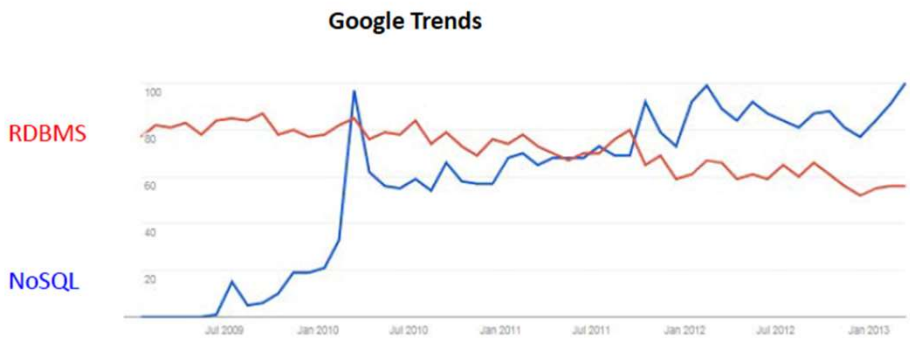


7

NoSQL landscape

8

NoSQL raising



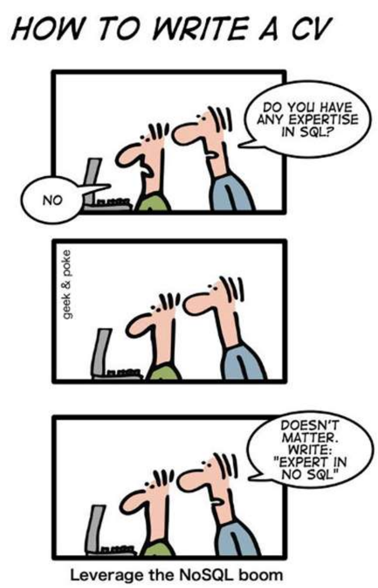
http://www.google.com/trends/explore#q=nosql%2C%20rdbms&date=1%2F2009%2051m&cmpt=q

9

Why NoSQL

- “The whole point of seeking alternatives [to RDBMS systems] is that you need to solve a problem that relational databases are a bad fit for.” Eric Evans - Rackspace

10



11

Why NoSQL [cont'd]

- ACID does not scale
- Web applications have different needs
  - Scalability
  - Elasticity
  - Flexible schema/ semi-structured data
  - Geographically distributed
- Web applications do not always need
  - Transaction
  - Strong consistency
  - Complex queries

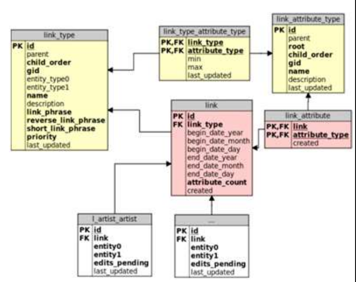
12

NoSQL use cases

- Massive data volume (Big volume)
  - Google, Amazon, Yahoo, Facebook – 10-100K servers
- Extreme query workload
- Schema evolution

Relational data model revisited

- Data is usually stored in row by row manner (row store)
- Standardized query language (SQL)
- Data model defined **before** you add data
- Joins merge data from multiple tables
  - Results are tables
- **Pros:** Mature ACID transactions with fine-grain security controls, widely used
- **Cons:** Requires up front data modeling, does not scale well



Oracle, MySQL, PostgreSQL, Microsoft SQL Server, IBM DB/2

Key/value data model

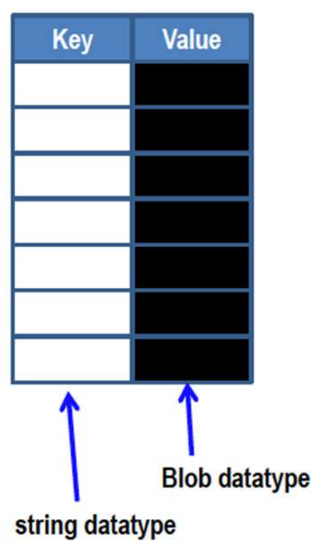
- Simple key/value interface
  - GET, PUT, DELETE
- Value can contain any kind of data
- Pros
- Cons
- Berkley DB, Memcache, DynamoDB, S3, Redis, Riak

key	value
firstName	Bugs
lastName	Bunny
location	Earth

15

Key/value vs. table

- A table with two columns and a simple interface
  - Add a key-value
  - For this key, give me the value
  - Delete a key
- Super fast and easy to scale (no joins)

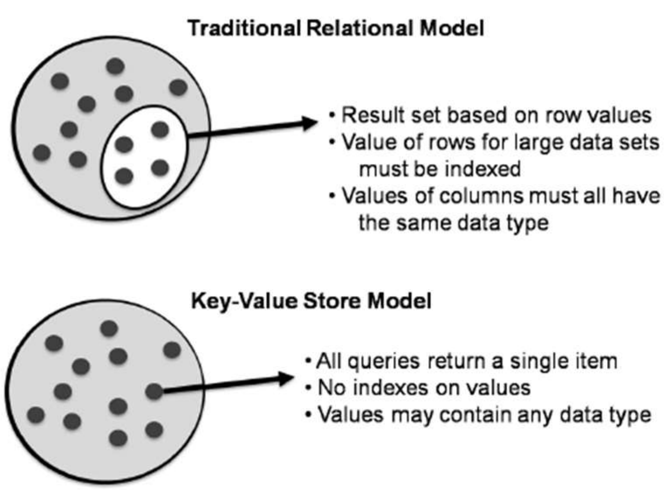


16

Key/value vs. locker

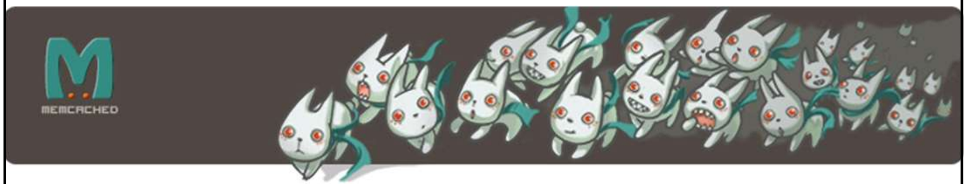
17

vs. Relational Model



18

Memcached



- Open source **in-memory key-value** **caching system**
- Make effective use of RAM on many distributed web servers
- Designed to speed up dynamic web applications by alleviating database load
  - Simple interface for highly distributed RAM caches
  - 30ms read times typical
- Designed for quick deployment, ease of development
- APIs in many languages

19

redis

- Open source in-memory key-value store with optional durability
- Focus on high speed reads and writes of common data structures to RAM
- **Allows simple lists, sets and hashes to be stored within the value and manipulated**
- Many features that developers like expiration, transactions, pub/sub, partitioning

20

amazon DynamoDB

- Scalable key-value store
- Fastest growing product in Amazon's history
- Focus on throughput not storage and predictable read and write times
- **Strong integration with S3 and Elastic MapReduce**

21

riak

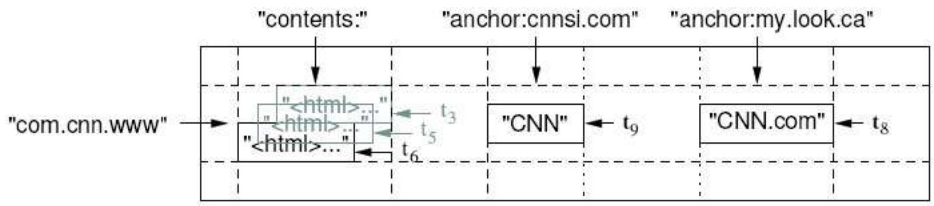
- Open source **distributed key-value store** with support and commercial versions by Basho
- A "Dynamo-inspired" database
- Focus on availability, fault-tolerance, operational simplicity and scalability
- Support for replication and auto-sharding and rebalancing on failures
- Support for MapReduce, fulltext search and secondary indexes of value tags
- Written in ERLANG

22

Column family store

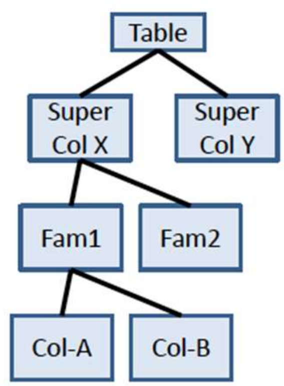
- Dynamic schema, column-oriented data model
- Sparse, distributed persistent multi-dimensional sorted map

(row, column (family), timestamp) -> cell contents



Column families

- Group columns into "Column families"
- Group column families into "Super-Columns"
- Be able to query all columns with a family or super family
- Similar data grouped together to improve speed



Column family data model vs. relational

- Sparse matrix, preserve table structure
  - One row could have millions of columns but can be very sparse
- Hybrid row/column stores
- Number of columns is extendible
  - New columns to be inserted without doing an "alter table"

Key ⟵⟶ (Row-ID, Column Family, Column Name)

Row-ID	Column Family	Column Name	Timestamp	Value

Bigtable

- ACM TOCS 2008
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

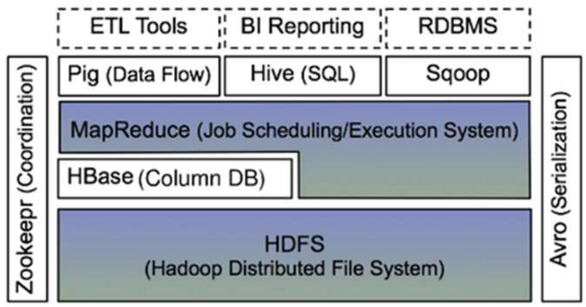
Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Sec-

APACHE
HBASE

- Open-source Bigtable, written in JAVA
- Part of Apache Hadoop project



27

Hadoop?

28

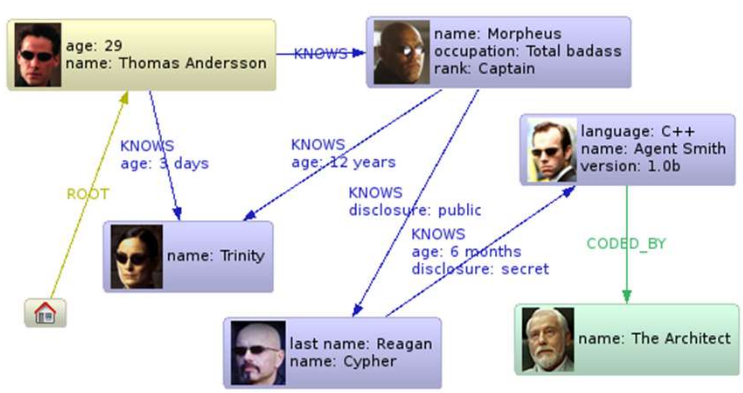
Cassandra

- Apache open source column family database
- Supported by DataStax
- Peer-to-peer distribution model
- Strong reputation for linear scale out (millions of writes/second)
- Written in Java and works well with HDFS and MapReduce

29

Graph data model

- Core abstractions: Nodes, Relationships, Properties on both



30

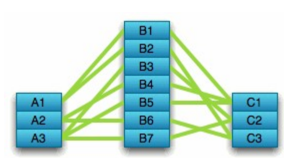
Graph database (store)

- A database stored data in an explicitly graph structure
- Each node knows its adjacent nodes
- Queries are really graph traversals

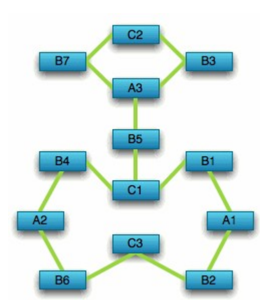
31

Compared to Relational Databases

Optimized for aggregation



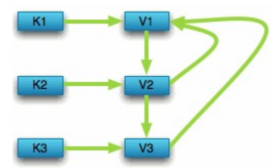
Optimized for connections



Compared to Key Value Stores

Optimized for simple look-ups

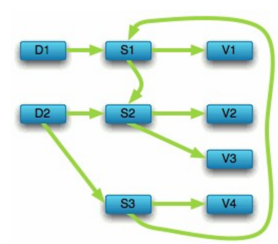
Optimized for traversing connected data



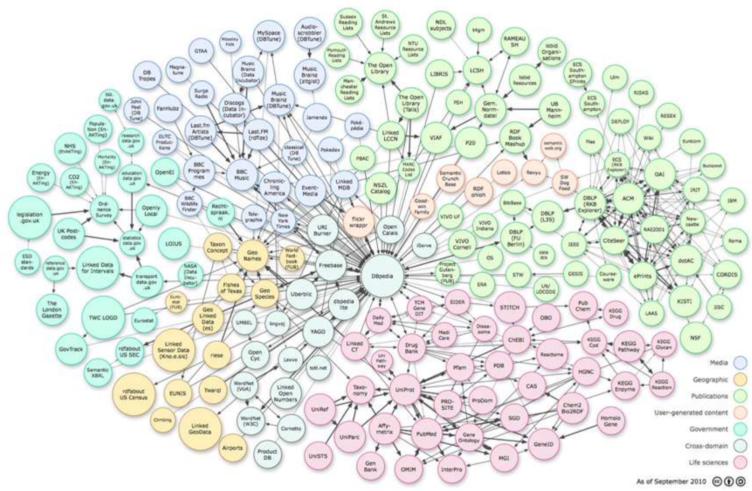
Compared to Key Value Stores

Optimized for "trees" of data

Optimized for seeing the forest and the trees, and the branches, and the trunks

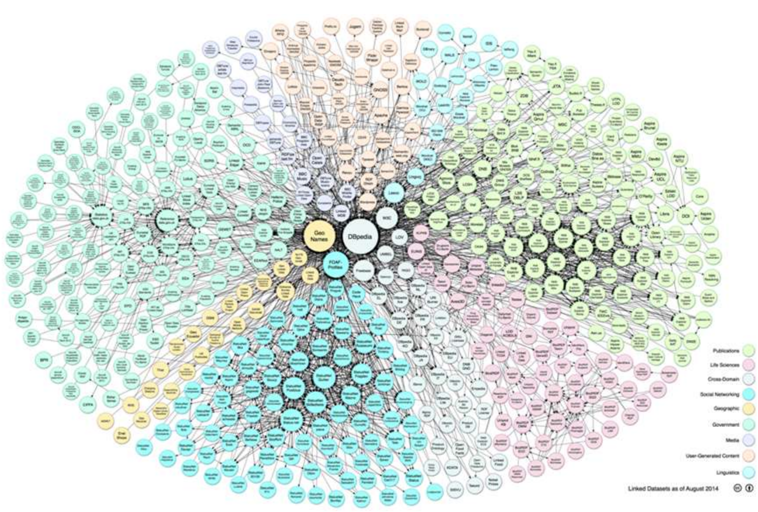


LINKINGOPENDATA
W3C SWEO Community Project



35

LINKINGOPENDATA
W3C SWEO Community Project



36

Neo4j

- Graph database designed to be easy to use by Java developers
- Disk-based (not just RAM)
- Full ACID
- High Availability (with Enterprise Edition)
- 32 Billion Nodes, 32 Billion Relationships, 64 Billion Properties
- Embedded java library
- REST API

37

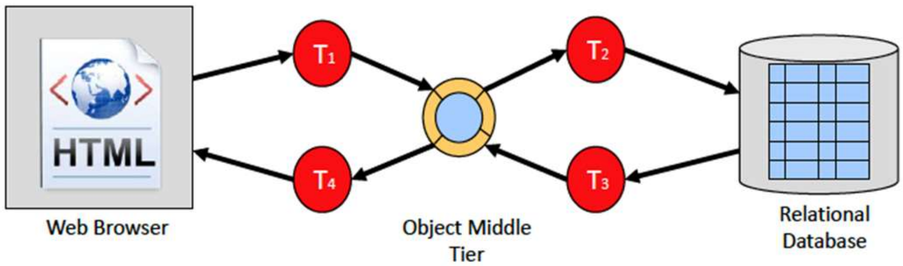
Document store

- Documents, not value, not tables
- JSON or XML formats
- Document is identified by ID
- Allow indexing on properties

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

38

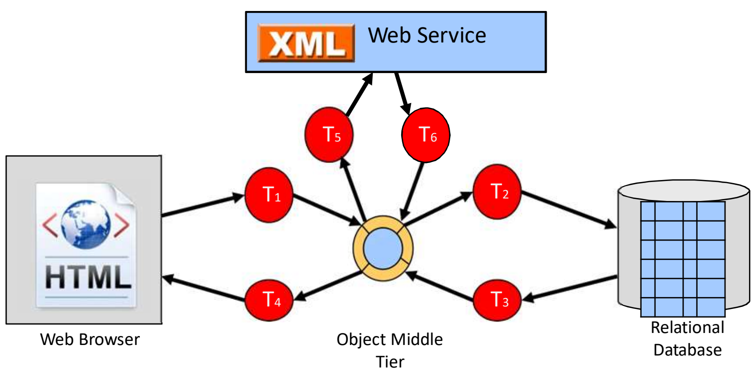
Relational data mapping



- T1–HTML into Objects
- T2–Objects into SQL Tables
- T3–Tables into Objects
- T4–Objects into HTML

39

Web Service in the middle



- T1 – HTML into Java Objects
- T2 – Java Objects into SQL Tables
- T3 – Tables into Objects
- T4 – Objects into HTML
- T5 – Objects to XML
- T6 – XML to Objects

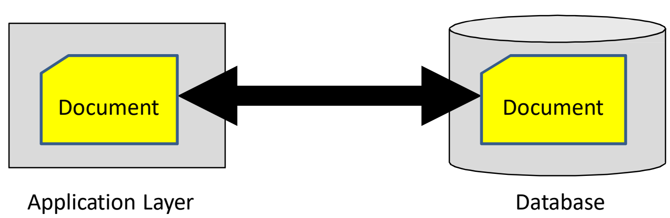
40

Discussion

- Object-relational mapping has become one of the most complex components of building applications today
  - Java Hibernate Framework
  - JPA
- To avoid complexity is to keep your architecture very simple

41

Document mapping



- Documents in the database
- Documents in the application
- No object middle tier
- No "shredding"
- No reassembly
- Simple!

42

mongoDB

- Open Source JSON data store created by 10gen
- Master-slave scale out model
- Strong developer community
- Sharding built-in, automatic
- Implemented in C++ with many APIs (C++, JavaScript, Java, Perl, Python etc.)

43

- Apache project
- Open source JSON data store
- Written in ERLANG
- RESTful JSON API
- B-Tree based indexing, shadowing b-tree versioning
- ACID fully supported
- View model
- Data compaction
- Security

Apache CouchDB™ is a database that uses **JSON** for documents, **JavaScript** for **MapReduce** indexes, and regular **HTTP** for its **API**

44