Modeling for NoSQL and SQL Databases

2014 NoSQL Now

by Michael Bowers
2014-08-13
v. 3.1
Abstract

We are in the middle of a database revolution. NoSQL is disrupting the database world by innovating in many disruptive ways.

- How do we model in these new paradigms?
- How does the old SQL paradigm fit in this new brave world?
- What paradigm is best for your project?

- We are in a new data paradigm:
  - New database architectures (software and hardware) that handle the large and ever growing velocity and volume of data dispersed across geographically distant data centers
  - New graph and document modeling paradigms that compete with object, relational, and dimensional
  - Schema-less databases to enable extreme agility of software development and rapid changes to huge data sets
What will you learn?

• You will understand the strengths and weaknesses of relational, dimensional, document, key-value, and triple models.

• You will be able to choose the best model to meet your specific needs.

• You will be able to choose the best NoSQL database to fit your model and other concerns, such as write performance, read performance, data integration, etc.

• You will be able to choose when to use ACID or BASE consistency models.
About the Author

Michael Bowers
• Principal Architect
  LDS Church

• Author
  – Pro CSS and HTML Design Patterns
    • Published by Apress, 2007

  – Pro HTML5 and CSS3 Design Patterns
    • Published by Apress, 2011

• mike@cssDesignPatterns.com
We are in a Database Revolution

• Existing paradigms are being challenged
  – Models
  – Hardware
  – Software
  – Languages

• Will tweaking current data solutions be enough?
Agenda

• Defining NoSQL and Big Data
• Modeling Paradigms
• Optimizing for Velocity or Volume
• Optimizing for Availability or Consistency
• Summary
Agenda

• Defining NoSQL and Big Data
• Modeling Paradigms
• Optimizing for Velocity or Volume
• Optimizing for Availability or Consistency
• Summary
Databases (Ranked by Popularity)

Graph/RDF
- #22 Neo4j
- #35 MarkLogic
- #66 OrientDB
- #101 AllegroGraph
- #148 InfiniteGraph

Document
- #5 MongoDB
- #21 CouchDB
- #24 Couchbase
- #35 MarkLogic
- #65 Cloudant
- #66 OrientDB

Column
- #10 Cassandra
- #11 Redis
- #15 Hbase
- #18 MemcacheDB
- #84 Aerospike
- #30 Riak
- #32 DynamoDB
- #56 Accumulo
- #71 Oracle NoSQL
- #127 FoundationDB

newSQL
- #60 Oracle x10,
- #62 GemFire
- #81 VoltDB
- #108 Clustrix

Live Analytics
- #1 Oracle Exalytics
- #23 SAP HANA
- #99 memsql

Big Data
- #19 Hadoop
- #20 Splunk

Doc Warehouse
- XML
- #12 Solr
- #16 ElasticSearch
- #33 Sphinx
- #35 MarkLogic

Data Warehouse
- #1 Oracle DB
- #2 MySQL
- #3 SQL Server
- #4 PostgreSQL
- #6 DB2
- #9 Sybase ASE
- #17 Informix
- #25 Firebird

Less structure (schemaless)

More structure (schema)
What’s wrong with SQL DB’s?

• Relevance
• Velocity
• Volume
• Variety
• Variability
Variability

Managing Rapid Change

- **Schemas** are incompatible with rapid change
  - **Constantly evolving data structures**
    - Can we afford to keep a large application in sync with regular changes to data structures?
  - **Big data**
    - Is data so large that it takes too long to modify values, structures, and indexes?
  - **Agile development**
    - Are requirements stable enough to create long-lasting relational data structures?

- **Schemaless** data is ideal for rapid change
  - **Schemaless data and languages**
    - JSON/JavaScript, Triple/SPARQL, XML/XQuery
  - **Defensive programming** is required
    - You never know what queries will return
Variety

Handling data in all imaginable forms

• Impedance mismatch
  – Different data structures
    • Structured, unstructured, semi-structured
  – Different data paradigms
    • Relational, Dimensional, Document, Graph, Object-oriented, etc.
  – Different data types
    • JSON doesn’t have a date/time/duration type, XML schema and SQL have a variety, etc.
  – Different markup standards
    • JSON, XML, RDF, etc
Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Information Retrieval, Volume 13 / Number 6 / June, 1970

Programs should remain unaffected when the internal representation of data is changed. ...Tree-structured inadequacies are discussed. ...Relations are discussed and applied to the problems of redundancy and consistency...

KEY WORDS AND PHRASES: data base, data structure, data organization, hierarchies of data, networks of data, relations

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22

1. Relational Model and Normal Form

1.1. INTRODUCTION
This paper is concerned with the application of elementary relation theory to formatted data...The problems...are those of data independence...and...data inconsistency...
The relational view...appears to be superior in several respects to the graph or network model...
...Relational view...forms a sound basis for treating derivability, redundancy, and consistency...[and] a clearer evaluation...of

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS
...Tables...represent a major advance toward the goal of data independence...

1.2.1. Ordering Dependence. Programs which take advantage of the stored ordering of a file are likely to fail...if...it becomes necessary to replace that ordering by a different one.

1.2.2. Indexing Dependence. Can application programs...remain invariant as indices come and go?

1.2.3. Access Path Dependence. Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data...These programs fail when a change in structure becomes necessary. The...program...is required to exploit...paths to the data...Programs become dependent on the continued existence of the...paths.
Agenda

• Defining NoSQL and Big Data
• Modeling Paradigms
• Optimizing for Velocity or Volume
• Optimizing for Availability or Consistency
• Summary
Five Data Paradigms

Relational
Flexible Queries

- Hospital
  - Hospital ID
  - Hospital Name
- Surgeon
  - Surgeon ID
  - Surgeon Name
- Operation Codes
  - Operation Code ID
  - Operation Code Type

Dimensional
Data Warehousing

- Hospital Dimension
  - Hospital ID
  - Attributes...
- Operation
  - Operation ID
  - Hospital ID
  - Surgeon ID
  - Operation Name
  - Attributes...
- Surgeon Dimension
  - Surgeon ID
  - Attributes...
- Drug Dose Facts
  - Drug ID
  - Hospital ID
  - Surgeon ID
  - Operation ID
  - Drug Dose

Graph
Unlimited Relationships

- Surgeon
  - performed
  - operated on
- Operation
  - operated on
- Person
  - operated on
- Hospital
  - works at

Document
Easy Development

<table>
<thead>
<tr>
<th>Hospital Name</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Operation ID</td>
<td>13</td>
</tr>
<tr>
<td>Surgeon Name</td>
<td>Dorothy Oz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micillin</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Micillin</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Micillin</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

Column/Key-value
Fast Puts and Gets

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillin</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Relational Model

PROs
• Most flexible queries & updates
• Reuse data structures in any context
• Great DB-to-DB integration
• Mature tools
• Standard Query Language
• Easy to hire expertise

CONs
• Design-time, static relationships
• Design-time, static structures: design first then load data
• Hard to normalize model
• Requires code to integrate relational data with object-oriented code
• Cannot query for relevance
**Dimensional Model**

**PROs**
- Queries facts in context
- Self-service, ad hoc queries
- High-performance platforms
- Mature tools and integration
- Standard Query Language
- Turns data into information

**CONs**
- Expensive platforms
- Design-time, static structures: design structures first then load data
- Design-time, static relationships
- Cannot query for relevance
- Cannot query for answers that are not built into the model
Three Column / Key-Value Models

Multidimensional key plus a cell value

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>12</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
</tbody>
</table>

Column

<table>
<thead>
<tr>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

Simple key plus multidimensional value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
## Multidimensional Key / Cell Value

### PROs
- Fast puts and gets
- Massive scalability
- Easy to shard & replicate
- Data colocation
- Simple to model
- Developer in control

### CONs
- Carefully design key
- Shred JSON into flat columns
- Secondary indexes required to query outside of hierarchical key
- No standard query API or Lang
- Hand code all joins in app
- Immature tools and platform
- Hard to integrate and hire

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drug USA</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose Size</td>
<td>20140814</td>
<td>150</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>
### Simple Key / Multidimensional Value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

**PROs**
- Fast puts and gets
- Massive scalability
- Easy to shard & replicate
- Simple to model
- Inexpensive
- Data in transactional context
- Developer in control

**CONs**
- Carefully design value structures
- Shred JSON value into flat columns
- Secondary indexes required to query values
- Design queries using non-standard API or query language
- Hand code all joins & referential integrity into application
- Cannot query for relevance
- Immature tools and platform
- Hard to integrate and hire expertise
Column Model

Column model adds a table-like layer and a SQL-like language on top of an internal multi-dimensional key/value engine.
**Document Model**

**PROs**
- Fast development
- Schemaless, run-time designed, rich, JSON and/or XML data structures
- Queries everything in context
- Self-service, ad hoc queries
- Turns data into information
- Can query for relevance

**CONs**
- Defensive programming for unexpected data structures
- Expensive platforms, immature tools, and hard to integrate
- Non-standard Query Languages, and hard to hire expertise
- Not as fast as Column/Key-Value databases

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

**Hospital Name:** John Hopkins  
**Operation Type:** Heart Transplant  
**Operation ID:** 13  
**Surgeon Name:** Dorothy Oz  
**Hospital Name:** John Hopkins  
**Operation Type:** Heart Transplant  
**Operation ID:** 13  
**Surgeon Name:** Dorothy Oz
PROs
• Unlimited flexibility — model any structure
• Run time definition of types & relationships
• Relate anything to anything in any way
• Query relationship patterns
• Standard Query Language (SPARQL)
• Creates maximal context around data

CONs
• Hard to model at such a low level
• Hard to integrate with other systems
• Immature tools
• Hard to hire expertise
• Cannot query for relevance because original document context is not preserved
Modeling Takeaways

Each model has a specialized purpose

- **Dimensional**  
  Business Intelligence reporting and analytics

- **Relational**  
  Flexible queries, joins, updates, mature, standard

- **Column/Key-Value**  
  Simple, fast puts and gets, massively scalable

- **Document**  
  Fast development, schemaless JSON/XML, searchable

- **Graph/RDF**  
  Modeling anything at runtime including relationships
Relational Modeling

in detail
Relational Modeling

#1 Normalize

- Give each attribute its own field
- Group attributes into tables ensuring each table has one coherent context
- Assign one primary key to each table
- Eliminate duplicate attributes

---

**Document ID: 1**

<table>
<thead>
<tr>
<th>Hospital Name:</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type:</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Surgeon Name:</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Operation Number:</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
#2 Orthogonalize

- Create reference tables that stand independent of all contexts
- This maximizes data reuse by allowing tables to be combined with other tables to create any context

### Document ID: 1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospital Name:</td>
<td>John Hopkins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation Type:</td>
<td>Heart Transplant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surgeon Name:</td>
<td>Dorothy Oz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation Number:</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
#3 Generalize

- Make tables more general in purpose so they can be reused in multiple contexts
- Such as replacing Surgeon with person
- Do not over generalize because it hides the purpose of the model

### Document ID: 1

<table>
<thead>
<tr>
<th>Hospital Name:</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type:</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Surgeon Name:</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Operation Number:</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Relational Modeling

#4 Tune
- Modify tables to meet the applications performance needs for reads and writes
- Such as speeding reads by materializing views and duplicating attributes across tables to eliminate joins

### Document ID: 1

| Hospital Name: | John Hopkins |
| Operation Type: | Heart Transplant |
| Surgeon Name: | Dorothy Oz |
| Operation Number: | 13 |

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Relational Modeling Exercise

Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address: Street City, State, Postal Code</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

PROs

- 

CONs

- 

- 

-
Relational Model

Summary

Use for maximum flexibility in querying and updating operational data

Example:
traditional data-entry apps
Dimensional Modeling

in detail
#1 Model Contexts

- Determine the business questions you want to answer
- Determine which fact will answer one or more questions
- Determine the grain of the fact
- Determine the dimensions needed to join with the fact to answer the business questions
- Create one star schema (or OLAP model) per fact
Dimensional Modeling

#2 ELT

- Extract data from a source system
- Load it into a staging area in the data warehouse
- Transform it into the star schema
  - Improve data quality
#3 Semantic Layer

- Define semantic layer to enable self-service reporting
  - Rename columns to be business friendly
  - Add descriptions to columns
  - Create “join paths” for error-free reporting
Dimensional Modeling

#4 Tune
- Determine query patterns
- Optimize queries
- Optimize indexes and full-table scans
- Move to specialized data warehouse technology
Dimensional Modeling Exercise

### Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address:</td>
<td>Street City, State, Postal Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

### PROs
- 
- 
- 

### CONs
- 
- 
- 
-
Dimensional Model

Summary

Use to transform authoritative data into contextual information to enable self-service, ad hoc, flexible reporting

Examples: Business Intelligence, Data Warehouse
Column/Key-Value Modeling

in detail
Databases (Ranked by Popularity)

Key-value DBs ↔ Evolve

Column DBs Evolve⇒

High Bandwidth Analytical Volume

Low Latency Operational Velocity

Less structure (schemaless) More structure (schema)
# Column/Key-Value Databases

**Column model or Multi-dimensional Key**

<table>
<thead>
<tr>
<th>Database</th>
<th>Query</th>
<th>Unique Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>CQL</td>
<td><strong>Schema-defined</strong>, collocated, composite columns</td>
</tr>
<tr>
<td>HBase</td>
<td>API</td>
<td>Massively sparse columns on HDFS</td>
</tr>
<tr>
<td>Aerospike</td>
<td>AQL</td>
<td>Schemaless with dynamically typed columns</td>
</tr>
<tr>
<td>Accumulo</td>
<td>API</td>
<td>Like HBase with cell-level security</td>
</tr>
</tbody>
</table>

**Simple Key**

<table>
<thead>
<tr>
<th>Database</th>
<th>Query</th>
<th>Unique Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>API</td>
<td>In-memory data structures</td>
</tr>
<tr>
<td>MemcachedB</td>
<td>API</td>
<td>Simple Memcache API</td>
</tr>
<tr>
<td>Riak</td>
<td>API</td>
<td>Search, MapReduce</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>API</td>
<td>Schemaless: key plus flat JSON-like value</td>
</tr>
<tr>
<td>Oracle NoSQL</td>
<td>API</td>
<td>N-dimensional keys, <strong>JSON</strong> values, ACID</td>
</tr>
<tr>
<td>FoundationDB</td>
<td>API, SQL</td>
<td><strong>ACID</strong>, user-defined key structures</td>
</tr>
</tbody>
</table>
# 3 Column / Key-Value Models

## Multidimensional key plus Cell value

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillin</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>12</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillin</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>15</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>12</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
</tbody>
</table>

## Simple key plus Multidimensional value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs R Us</td>
<td>200 mg</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillin</td>
<td>Canada4Less Drugs</td>
<td>400 mg</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drug USA</td>
<td>150 mg</td>
<td></td>
</tr>
</tbody>
</table>
Multidimensional Key/Cell Model

### Multidimensional key plus Cell value

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Operation Type</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillin</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs &amp; Us</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200 mg</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillin</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400 mg</td>
<td></td>
</tr>
</tbody>
</table>

### Simple key plus multidimensional value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs &amp; Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Heart Transplant</td>
<td>20140814</td>
<td>Dorothy Oz</td>
<td>Maxicillin</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Heart Transplant</td>
<td>20140814</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs USA</td>
<td>550</td>
<td>mg</td>
<td></td>
</tr>
</tbody>
</table>
# Multidimensional Key / Cell Value

## #1 Model Transactions
- Create a denormalized hierarchical key structure that connects each attribute of the transaction

<table>
<thead>
<tr>
<th>Hospital Name</th>
<th>Operation Type</th>
<th>Operation ID</th>
<th>Surgeon Name</th>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Hopkins</td>
<td>Heart Transplant</td>
<td>13</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drug USA</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose Size</td>
<td>20140814</td>
<td>150</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>
Multidimensional Key / Cell Value

#2 Review Colocation Structure of Key

- The key defines how data is collocated on disk.
- **OpsDrugs** table is collocated within the Ops DB.
- **Hospital IDs** are collocated within the OpsDrugs table.
- **Op IDs** are collocated within the Hospital ID rows.
- **Drug IDs** are collocated within the Op ID rows.
- **Columns** are collocated within each row, etc.

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drug USA</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose Size</td>
<td>20140814</td>
<td>150</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>
#3 Verify Colocation Effects on Queries

- Queries are fast when they use the key to retrieve data.
- You can retrieve all values that are collocated within a portion of the key:

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>

⇒ Returns all cells in the Ops database
⇒ Returns all cells in the OpsDrugs table
⇒ Returns all cells in the John Hopkins hospital
⇒ Returns all cells in Operation 13
⇒ Returns all cells for Drug ID 1997
⇒ Returns the value for the Surgeon cell
#4 Sharding Strategy

- Determine best sharding strategy for the data based on the quantities of data in each key and geographic distribution of data by key.
- Configure how the key is sharded and replicated across servers in the cluster and across data centers.
#5 Modify Key

- Modify key to match query needs, optimize collocation, and optimize sharding
- For example, you may want to move the Drug ID before Op ID. This is because
  - Drugs are queried more often than Operations
  - The large amount of data within drugs makes it a good segment for sharding

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Drug ID</th>
<th>Op ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs John Hopkins</td>
<td>1997</td>
<td>13</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
<td></td>
</tr>
</tbody>
</table>
#6 Create Secondary Indexes

- Create secondary indexes for queries that do not follow the hierarchy of the key
- For example, you need a secondary index on surgeon if you want to quickly find all operations performed by a surgeon
- Secondary indexes slow down inserts, updates, and deletes

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drug USA</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose Size</td>
<td>20140814</td>
<td>150</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>
### #7 Write Code

- Developer writes application code against the database API or DSL to
  - create keys
  - create secondary indexes
  - put data
  - delete data
  - get data
  - join data
  - ensure data integrity

(NOTE: joins and data integrity are not part of the database)

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>20140814</td>
<td>200 mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Maxicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>20140814</td>
<td>400 mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug Name</td>
<td>20140814</td>
<td>Minicillan</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drug USA</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose Size</td>
<td>20140814</td>
<td>150 mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>Johns Hopkins</td>
<td>13</td>
<td>9448</td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>

52
### Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address:</td>
<td>Street City, State, Postal Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

**PROs**
- 
- 
- 
- 

**CONS**
- 
- 
- 
- 

Multidimensional Key /Cell Model

summary

Use for maximum speed and scalability by hand-tuning application code for queries & inserts to create Internet-scale applications

Example:
Netflix, Google, Linked-in, etc.
Column Model

Multidimensional key plus a cell value

<table>
<thead>
<tr>
<th>DB</th>
<th>Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Operation Type</td>
<td>2014-08-14</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Surgeon</td>
<td>2014-08-14</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>12</td>
<td>1997</td>
<td>Drug Name</td>
<td>2014-08-14</td>
<td>Minicillin</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Drug MFG</td>
<td>2014-08-14</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose Size</td>
<td>2014-08-14</td>
<td>250</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>Dose UOM</td>
<td>2014-08-14</td>
<td>mg</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Operation Type</td>
<td>2014-08-14</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Surgeon</td>
<td>2014-08-14</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug Name</td>
<td>2014-08-14</td>
<td>Minicillin</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Drug MFG</td>
<td>2014-08-14</td>
<td>Canada4Less Drugs</td>
</tr>
<tr>
<td>Ops</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>Dose Size</td>
<td>2014-08-14</td>
<td>400</td>
</tr>
</tbody>
</table>

Simple key plus multidimensional value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>13</td>
<td>1997</td>
<td>2014-08-14</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs R Us</td>
<td>250</td>
<td>mg</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>13</td>
<td>2110</td>
<td>2014-08-14</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillin</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>13</td>
<td>9448</td>
<td>2014-08-14</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillin</td>
<td>Drugs USA</td>
<td>150</td>
<td>mg</td>
<td></td>
</tr>
</tbody>
</table>
#1 Column model

- Easier-to-use version of the Multi-dimensional Key/Cell model
- Cells are pivoted into columns in a table
- SQL-like Query Languages make it easy to create tables and to query them
- Modeling is the same as modeling for the Multi-dimensional Key/Cell model
Column Model

summary

Use for maximum speed and scalability with SQL-like code for all operations to create Internet-scale applications

Example: Netflix, Google, Linked-in, etc.
Simple Key / Multidimensional Value Model

Multidimensional key plus a cell value

<table>
<thead>
<tr>
<th>DB Table</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Column Type</th>
<th>Time Stamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Operation Type</td>
<td>20140814</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Surgeon</td>
<td>20140814</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Drug Name</td>
<td>20140814</td>
<td>Micillin</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Drug MFG</td>
<td>20140814</td>
<td>Drugs R Us</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Dose Size</td>
<td>20140814</td>
<td>200</td>
</tr>
<tr>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td></td>
<td>Dose UOM</td>
<td>20140814</td>
<td>mg</td>
</tr>
</tbody>
</table>

Simple key plus Multidimensional value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Micillin</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Micillin</td>
<td>CanadaLess Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Micillin</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
# Simple Key / Multidimensional Value

## #1 Model Transactions
- Create a denormalized flat data structure for each attribute in the transaction
- Each record will be accessed through simple, meaningless key

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital Name</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon Name</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

**Hospital Name:** John Hopkins  
**Operation Type:** Heart Transplant  
**Operation ID:** 13  
**Surgeon Name:** Dorothy Oz

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
## Simple Key / Multidimensional Value

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1997</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

### #2 Create Secondary Indexes
- Determine which attributes need to be queried and create secondary indexes on them
#3 Write Code

- Developer writes application code against the database API or DSL to create keys, create secondary indexes, put records, delete records, get records, join records, ensure data integrity
- Joins and data integrity have to be done in application code because they are not part of the database
Databases (Ranked by Popularity)

Key-value DBs ⇔ Evolve

- Raw
  - #19 Hadoop
  - #20 Splunk

- Graph
  - #16 ElasticSearch
  - #33 Sphinx
  - #35 MarkLogic

- Document
  - #23 Couchbase
  - #35 MarkLogic

- Key-value/Column
  - #10 Cassandra
  - #11 Redis
  - #15 Hbase
  - #18 Memcached
  - #30 Riak
  - #32 DynamoDB
  - #56 Accumulo
  - #71 Oracle NoSQL
  - #127 FoundationDB

- newSQL
  - #60 Oracle x10
  - #62 GemFire
  - #81 VoltDB
  - #108 Clustrix

- Live Analytics
  - #1 Oracle Exalytics
  - #23 SAP HANA
  - #99 memsql

- Data Warehouse
  - #1 Oracle DB
  - #2 MySQL
  - #3 SQL Server
  - #4 PostgreSQL
  - #6 DB2
  - #9 Sybase ASE
  - #17 Informix
  - #25 Firebird

- Big Data
  - #19 Hadoop
  - #20 Splunk

- Raw
  - #12 Solr
  - #21 CouchDB
  - #24 Couchbase

- Column
  - key/val

- Relational
  - #1 Oracle DB
  - #2 MySQL
  - #3 SQL Server
  - #4 PostgreSQL
  - #6 DB2
  - #9 Sybase ASE
  - #17 Informix
  - #25 Firebird

- Dimensional
  - #1 Oracle Exadata
  - #13 Teradata
  - #26 Netezza
  - #22 Sybase IQ
  - #29 Vertica
  - #37 Greenplum

Less structure (schemaless) – More structure (schema)
Key/Multi-dim Value Modeling Exercise

Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address:</td>
<td>Street City, State, Postal Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

**PROs**
- 
- 
- 
- 

**CONs**
- 
- 
- 
- 

Simple Key / Multidimensional Model summary

Use when you need maximum speed to retrieve a set of flat values by key

A document DB is better than a simple key/value DB because a JSON or XML document is a richer value
Document Modeling

in detail
Databases (Ranked by Popularity)

- **Graph/RDF**
  - Neo4j
  - MarkLogic
  - Clustrix

- **Document**
  - MongoDB
  - CouchDB
  - Couchbase
  - CloudLogix
  - OrientDB

- **Column**
  - Cassandra
  - Redis
  - Memcached

- **newSQL**
  - Oracle x10
  - GemFire
  - VoltDB

- **Live Analytics**
  - Oracle Exalytics
  - SAP HANA
  - memsql

- **Big Data**
  - Hadoop
  - Redis
  - HBase

- **Doc Warehouse**
  - Solr
  - ElasticSearch
  - Sphinx

- **XML**
  - Oracle NoSQL

- **JSON**
  - MongoDB
  - CouchDB

- **Raw Big Data**
  - Cassandra
  - Redis
  - Riak
  - DynamoDB

- **Dimensional**
  - Oracle NoSQL
  - FoundationDB

---

**Hospital Name:** John Hopkins

**Operation Number:** 13

**Operation Type:** Heart Transplant

**Surgeon Name:** Dorothy Oz

**Drug Name**
- Minicillan
- Maxicillan
- Minicillan

**Drug Manufacturer**
- Drugs R Us
- Canada4Less
- Drug USA

**Dose**
- 200 mg
- 400 mg
- 150 mg

**UOM**
- g

---

Less structure (schemaless)

More structure (schema)
What is a document?

A document is a nested structure referenced by key

{ "key": "DsaDoc1",  
  "type": "DataSharingAgreement",  
  "agreement": {  
    "name": "Sharing Address Web Service 2.1 with JSB application",  
    "agreementType": "web service",  
    "approvalDate": "2014-08-16",  
    "effectiveDate": "2014-11-06",  
    "descriptions": [  
      { "name": "Business Description",  
        "value": "JSB app needs access to cleansed addresses." },  
      { "name": "Business Justification",  
        "value": "JSB app needs to cleanse addresses it gets from users." }  
    ],  
    "relations": {  
      "values": [  
        { "subject": "DsaDoc1", "predicate": "requestingSystem", "object": "rDoc95",  
          "name": "JSB application" },  
        { "subject": "DsaDoc1", "predicate": "sourceInterface", "object": "iDoc67",  
          "name": "Address WS 2.1" },  
        { "subject": "DsaDoc1", "predicate": "requestingUser", "object": "uDoc1554",  
          "name": "Maya Zelstro" }  
      ]  
    }  
  } 
}
JSON vs. XML

**JSON**
1. Best for objects (text poured into them)
2. Easy and fast to read and parse
3. Simple and compact
4. Objects, arrays, and some types (Float, String, Boolean)

**XML**
1. Best for text (objects added on top)
2. Attributes add metadata to elements
3. Namespaces embed foreign types
4. Sets and all types (date, integer, etc.)
Relational Model of Data for Large Shared Data Banks

E. F. Codding

IBM Research Laboratory, San Jose, California

Information Retrieval, Volume 13 / Number 5 / June 1970

Programs should remain unaffected when the internal representation of data is changed. ...Tree-structured inadequacies...are discussed. ...Relations...are discussed and applied to the problems of redundancy and consistency....

KEY WORDS AND PHRASES: data base, data structure, data organization, hierarchies of data, networks of data, relations

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory...to...formatted data.... The problems...are those of data independence...and...data inconsistency....

The relational view...appears to be superior in several respects to the graph or network model....

...Relational view...forms a sound basis for treating derivability, redundancy, and consistency,... [and] a clearer evaluation...of...

1.2. Data Dependencies in Present Systems

...Tables...represent a major advance toward the goal of data independence....

1.2.1. Ordering Dependence...Programs which take advantage of the stored ordering of a file are likely to fail...if...it becomes necessary to replace that ordering by a different one.

1.2.2. Indexing Dependence...Can application programs...remain invariant as indices come and go?

1.2.3. Access Path Dependence...Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. ...These programs fail when a change in structure becomes necessary. The...program...is required to exploit...paths to the data...Programs become dependent on the continued existence of the...paths.
Document Modeling

#1 Model Transactions

- Each document is a transaction
- Document ID is the primary key
- Each document includes all data captured during the transaction
- Each document is historically accurate for its point in time
- No mismatch between JSON code and JSON data in database
- Secondary indexes flatten structure to make queries flexible
- Search indexes use the document for context

**Document ID: 1**

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

**Hospital Name:** John Hopkins

**Operation Type:** Heart Transplant

**Surgeon Name:** Dorothy Oz

**Operation Number:** 13
#2 Create Reference Docs

- Create additional document types for each type of reference data, such as Hospitals, Operation Types, Drugs, and Does Unit of Measures.

<table>
<thead>
<tr>
<th>Hospital ID</th>
<th>Hospital Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John Hopkins</td>
</tr>
<tr>
<td>20</td>
<td>Boston Children’s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation Type ID</th>
<th>Operation Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>200</td>
<td>Appendectomy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Surgeon ID</th>
<th>Surgeon Name</th>
<th>ID</th>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Dorothy Oz</td>
<td>10000</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2000</td>
<td>Van Tristic</td>
<td>20000</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>30000</td>
<td>Minicillan</td>
<td>30000</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
#3a Connect References

- **Best:** Use **triples** to *bi-directionally* connect each transaction document to its reference documents.
- This makes it easy to join across all document relationships in any way.

## Document ID: 1

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransDoc</td>
<td>SurgeryInHospital</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>SurgeryOperationType</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>SurgerySurgeon</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>SurgeryDrugsGiven</td>
<td>10000</td>
</tr>
<tr>
<td>1</td>
<td>SurgeryDrugsGiven</td>
<td>20000</td>
</tr>
<tr>
<td>1</td>
<td>SurgeryDrugsGiven</td>
<td>30000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Document Modeling

#3b Connect References

- **Good:** Use document references to uni-directionally connect each transaction document to its reference documents.
- This makes it easy to create links from a transaction document to its reference documents.
- If you index each reference ID, then you can write queries to find documents that use reference documents.

<table>
<thead>
<tr>
<th>Hospital ID</th>
<th>Hospital Name</th>
<th>Operation Type ID</th>
<th>Operation Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John Hopkins</td>
<td>100</td>
<td>Heart Transplant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug ID</th>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>20000</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>30000</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Document Modeling

#4 Sync References

- Optionally choose to synchronize changes in reference documents into transaction documents.
- To preserve history, add changes to new elements.
- To maximize integrity overwrite original.

<table>
<thead>
<tr>
<th>Hospital ID</th>
<th>Hospital Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>JH Research</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation Type ID</th>
<th>Operation Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Cardiac Transplant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Surgeon ID</th>
<th>Surgeon Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Dorothy Wiz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug ID</th>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000</td>
<td>Maxicillan</td>
<td>Best Drugs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Best Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
#5 Projections

- Optionally choose to project values from reference documents into transaction documents.
- For maximum read and search performance, project data during writes.
- For maximum write performance, project data during reads.

<table>
<thead>
<tr>
<th>Drug ID</th>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Drug Efficacy</th>
<th>Drug Recalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>20000</td>
<td>Maxicillan</td>
<td>Best Drugs</td>
<td>50%</td>
<td>3</td>
</tr>
<tr>
<td>30000</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>70%</td>
<td>1</td>
</tr>
</tbody>
</table>

### Document ID: 1

<table>
<thead>
<tr>
<th>Hospital Name</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Surgeon Name</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Operation Number</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Drug Efficacy</th>
<th>Drug Recalls</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>80%</td>
<td>1</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Best Drugs</td>
<td>50%</td>
<td>3</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>70%</td>
<td>1</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Why is the **document model** best for developer productivity?

- Supports agile development without a schema
- Handles rapidly changing requirements
- Handles deeply hierarchical, complex, and highly variable data structures
- Has little-to-no impedance mismatch between application and database
- JSON is the new lingua franca of the web
- Potential to enable better search relevance
  - Full-text search in context of document structure
  - Full-featured queries of any data anywhere in a document
## Document Modeling Exercise

### Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address:</td>
<td>Street, City, State, Postal Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

**PROs**

- 
- 
- 
- 

**CONs**

- 
- 
- 
- 


Document Model

summary

Use when you need maximum developer productivity and great speed and scalability

Example:
Enterprise applications, Websites, etc.
Document Model

*tip*

Use **JSON** for objects

Use **XML** for text

(to markup structure and semantics)
Graph Modeling in detail
Databases *(Ranked by Popularity)*

### Graph/RDF
- #22 Neo4j
- #35 MarkLogic
- #66 OrientDB
- #101 AllegroGraph
- #148 InfiniteGraph

### Document
- #5 MongoDB
- #21 CouchDB
- #24 Couchbase
- #35 MarkLogic
- #65 Cloudant
- #66 OrientDB

### Column
- #10 Cassandra
- #11 Redis
- #15 Hbase
- #18 MircacheDB
- #84 Aerospike
- #32 DynamoDB
- #56 Accumulo
- #71 Oracle NoSQL
- #127 FoundationDB

### newSQL
- #60 Oracle x10
- #62 GemFire
- #81 VoltDB
- #108 Clustrix

### Live Analytics
- #1 Oracle Exalytics
- #23 SAP HANA
- #90 memsql

### Data Warehouse
- #1 Oracle DB
- #2 MySQL
- #3 SQL Server
- #4 PostgreSQL
- #6 DB2
- #9 Sybase ASE
- #17 Informix
- #25 Firebird

### Big Data
- #19 Hadoop
- #20 Splunk

### Doc Warehouse
- #12 Solr
- #16 ElasticSearch
- #33 Sphinx
- #35 MarkLogic

### Raw
- #18 Hadoop
- #23 Redis
- #36 MongoDB
- #41 Couchbase
- #67 Cassandra

### Graph
- #19 Hadoop
- #20 Splunk
- #32 Apache Flink
- #36 MongoDB
- #41 Couchbase

### Document
- #19 Hadoop
- #20 Splunk
- #32 Apache Flink
- #36 MongoDB
- #41 Couchbase

### Key-value/Column
- #19 Hadoop
- #20 Splunk
- #32 Apache Flink
- #36 MongoDB
- #41 Couchbase

### Relational
- #19 Hadoop
- #20 Splunk
- #32 Apache Flink
- #36 MongoDB
- #41 Couchbase

### Dimensional
- #19 Hadoop
- #20 Splunk
- #32 Apache Flink
- #36 MongoDB
- #41 Couchbase

---

**Less structure (schemaless)**

**More structure (schema)**
What is a Triple?

*A triple is three IDs: subject, predicate, object*

```json
{  "triples": [
   {  "subject": "DsaDoc1",  "predicate": "requestingSystem",  "object": "requestingDoc95" },
   {  "subject": "DsaDoc1",  "predicate": "sourceInterface",  "object": "interfaceDoc67" },
   {  "subject": "DsaDoc1",  "predicate": "requestingUser",  "object": "userDoc1554" ]}
```

- **Subject**
  - It is the *focus* of the triple
    - It is a URI unique to the database

- **Predicate**
  - Specifies the *relationship* between the subject and the object
    - It is a URI typically defined by external ontologies

- **Object**
  - Specifies the *target* of the subject’s relationship
    - It is a value or a URI — typically the URI of another subject or a string, number, date, etc.
What is a Quad?

A Quad is a Triple prefixed with the collection it belongs to

{ "quads": [
  { "collection": "HospitalOps", "subject": "surgeonDoc1", "predicate": "excelsAt", "object": "operationDoc13" },
  { "collection": "HospitalOps", "subject": "surgeonDoc1", "predicate": "performed", "object": "operationDoc13" },
  { "collection": "HospitalOps", "subject": "surgeonDoc1", "predicate": "operatedOn", "object": "userDoc1554" },
  { "collection": "HospitalOps", "subject": "surgeonDoc1", "predicate": "worksAt", "object": "hospitalDoc10" },

  { "collection": "HospitalOps", "subject": "operationDoc13", "predicate": "requestingUser", "object": "userDoc1554" },
  { "collection": "HospitalOps", "subject": "operationDoc13", "predicate": "operatedAt", "object": "hospitalDoc10" },

  { "collection": "HospitalOps", "subject": "userDoc1554", "predicate": "patientAt", "object": "hospitalDoc10" } ]}
Triples break down data into singular **items** identified by IDs.

- An **item** only has meaning when it is **related** to other items or simple data.

- This is like deconstructing data into electrons, neutrons, and protons so that you can reconstruct any type of atom and then combine atoms into molecules, and combine molecules into compounds, etc.
The primary focus of triples is on relationships between items:

- Traversing a network of relationships
- Finding items that have the same relationship patterns

To get any information about an item requires querying relationships to other items

- To make this easier, some Triple databases allow items to have properties or allow items to be documents
Connecting Triples and Documents

- **Neo4J** provides properties on subjects, predicates, and objects.

- **MarkLogic** allows triple subjects, predicates, and objects to be references to documents and it allows documents to contain triples and projections of triple data.

- **OrientDB** connects documents using triples.
Embedding Triples in a Document

Use triples to relate documents bi-directionally

```json
{ "_key": "DsaDoc1", "type": "DataSharingAgreement", "agreement": { "name": "Sharing Address Web Service 2.1 with JSB application", "agreementType": "web service", "approvalDate": "2014-08-16", "effectiveDate": "2014-11-06", "descriptions": [ { "name": "Business Description", "value": "JSB app needs access to cleansed addresses." }, { "name": "Business Justification", "value": "JSB app needs to cleanse addresses it gets from users." } ], "relations": [ { "subject": "DsaDoc1", "predicate": "requestingSystem", "object": "requestingDoc95" }, { "subject": "DsaDoc1", "predicate": "sourceInterface", "object": "interfaceDoc67" }, { "subject": "DsaDoc1", "predicate": "requestingUser", "object": "userDoc1554" } ] }}
```
Projecting Triple Values into Documents

At write or read time you can project triple data into documents

```json
{
  "_key": "DsaDoc1",
  "type": "DataSharingAgreement",
  "agreement": {
    "name": "Sharing Address Web Service 2.1 with JSB application",
    "agreementType": "web service",
    "approvalDate": "2014-08-16",
    "effectiveDate": "2014-11-06",
    "descriptions": [
      { "name": "Business Description",
        "value": "JSB app needs access to cleansed addresses." },
      { "name": "Business Justification",
        "value": "JSB app needs to cleanse addresses it gets from users." }
    ],
    "relations": {
      "values": [
        { "subject": "DsaDoc1", "predicate": "requestingSystem", "object": "rDoc95" },
        { "subject": "DsaDoc1", "predicate": "sourceInterface", "object": "iDoc67" },
        { "subject": "DsaDoc1", "predicate": "requestingUser", "object": "uDoc1554" },
        { "subject": "DsaDoc1", "predicate": "associatedSystem", "object": "uDoc1554" }
      ]
    }
  }
}
```
Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Information Retrieval, Volume 13 / Number 5 / June, 1970

Programs should remain unaffected when the internal representation of data is changed. Re-structured inadequacies are discussed and applied to the problems of redundancy and consistency.

KEY WORDS AND PHRASES: data base, data structure, data organization, hierarchies of data, networks of data, relations

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22

1. Relational Model and Normal Form

1.1. INTRODUCTION
This paper is concerned with the application of elementary relation theory to formatted data. The problems are those of data independence and data inconsistency. The relational view appears to be superior in several respects to the graph or network model. Relational view forms a sound basis for treating derivability, redundancy, and consistency.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS
Tables represent a major advance toward the goal of data independence.

1.2.1. Ordering Dependence. Programs which take advantage of the stored order of a file are likely to fail if it becomes necessary to replace that ordering by a different one.

1.2.2. Indexing Dependence. Can application programs remain invariant as indices come and go?

1.2.3. Access Path Dependence. Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. These programs fail when a change in structure becomes necessary. The program is required to exploit paths to the data. Programs become dependent on the continued existence of the paths.
#1a Define Relationships

- Define a standard set of relationships with precise meanings
- This is critical because relationships assign meaning to items and make queries possible

**Document ID: 1**

<table>
<thead>
<tr>
<th>Hospital Name:</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type:</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Surgeon Name:</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Operation Number:</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Graph Modeling

#1b Use Existing Ontologies

• Save time and make your data easier to understand by leveraging existing relationship ontologies

TIP: Search for ontologies at [Linked Open Vocabularies (LOV)]

- Dublin Core
- FOAF
- TrackBack
- MetaVocab
- Basic Geo Vocabulary
- BIO
- RSS 1.0
- VCard RDF
- Creative Commons metadata
- WOT
- SIOC
- GoodRelations
- DOAP
- Programmes Ontology
- Music Ontology
- OpenGUID
- Provenance Vocabulary
- Pedagogical diagnosis
- DILIGENT Argumentation Ontology
Graph Modeling

#2 Define Attributes

- Create an ID for each item
- The ID can be human readable, but it usually is a variation of a UUID

<table>
<thead>
<tr>
<th>Document ID: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hospital Name:</strong></td>
</tr>
<tr>
<td><strong>Operation Type:</strong></td>
</tr>
<tr>
<td><strong>Surgeon Name:</strong></td>
</tr>
<tr>
<td><strong>Operation Number:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
#3 Add Core Data to Items

- Define **common core data** you want added to all items
- Because an item is simply an ID that has no meaning, you need to add core metadata to it, such as type, name, updatedBy, updatedOn, etc.
- If your database allows an item to be a document or to have properties, you can add core data directly to it
- In a pure triple system, use relationships to connect items to core data
Graph Modeling

#3 Add Relationships

- Create a triple for every relationship between items
- Graphs are schemaless — you can add more relationships at run time

---

**Document ID: 1**

<table>
<thead>
<tr>
<th>Hospital Name:</th>
<th>John Hopkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Type:</td>
<td>Heart Transplant</td>
</tr>
<tr>
<td>Surgeon Name:</td>
<td>Dorothy Oz</td>
</tr>
<tr>
<td>Operation Number:</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug Name</th>
<th>Drug Manufacturer</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>
Query using SPARQL

SPARQL is a triple query language

```sparql
SELECT *
WHERE {
  ?Surgeon excelsAt ?Operation
  ?Operation isNamed "Heart Surgery"
}
```

Returns all surgeons who excel at heart surgery.
Document ID: 1

<table>
<thead>
<tr>
<th>Invoice Number:</th>
<th>1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2014-08-16</td>
</tr>
<tr>
<td>Total Amount:</td>
<td>$40</td>
</tr>
<tr>
<td>Customer Name:</td>
<td>Mike Bowers</td>
</tr>
<tr>
<td>Customer Phone:</td>
<td>801-555-1212</td>
</tr>
<tr>
<td>Customer Address:</td>
<td>Street City, State, Postal Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Product Description</th>
<th>Price</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS Book</td>
<td>CSS and HTML Design</td>
<td>$20</td>
<td>1</td>
</tr>
<tr>
<td>CSS Book</td>
<td>HTML5 and CSS3 Design</td>
<td>$20</td>
<td>mg</td>
</tr>
</tbody>
</table>

**PROs**

- 
- 

**CONs**

- 
- 
-
Graph Model
summary

Model networks, relate documents, and enrich data

Example: genetics, family history, social networks
Narrow your choice of NoSQL database based on how its modeling paradigm matches your project.

Example: Choose graph if you are doing genetics, family history, social networks.
Five Data Paradigms

**Relational**
Flexible Queries

**Dimensional**
Data Warehousing

**Graph**
Unlimited Relationships

**Document**
Easy Development

**Column/Key-value**
Fast Puts and Gets

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital ID</th>
<th>Op ID</th>
<th>Drug ID</th>
<th>Time Stamp</th>
<th>Operation Type</th>
<th>Surgeon</th>
<th>Drug Name</th>
<th>Drug MFG</th>
<th>Dose Size</th>
<th>Dose UOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>1957</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drugs R Us</td>
<td>200</td>
<td>mg</td>
</tr>
<tr>
<td>2</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>2110</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Maxicillan</td>
<td>Canada4Less Drugs</td>
<td>400</td>
<td>mg</td>
</tr>
<tr>
<td>3</td>
<td>OpsDrugs</td>
<td>John Hopkins</td>
<td>13</td>
<td>9448</td>
<td>20140814</td>
<td>Heart Transplant</td>
<td>Dorothy Oz</td>
<td>Minicillan</td>
<td>Drug USA</td>
<td>150</td>
<td>mg</td>
</tr>
</tbody>
</table>

Hospital
Hospital ID
Hospital Name

Surgeon
Surgeon ID
Surgeon Name

Operation
Operation ID
Operation Name
Operation Type

Hospital Dimension
Hospital ID
Attributes...

Surgeon Dimension
Surgeon ID
Attributes...

Drug Dose Facts
Hospital ID
Surgeon ID
Operation ID
Drug ID
Drug Dose

Drug Dimension
Drug ID
Attributes...

Operation
Operation ID
Attributes...
### Modeling Takeaways

*Each model has a specialized purpose*

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dimensional</strong></td>
<td>Business Intelligence reporting and analytics</td>
</tr>
<tr>
<td><strong>Relational</strong></td>
<td>Flexible queries, joins, updates, mature, standard</td>
</tr>
<tr>
<td><strong>Column/Key-Value</strong></td>
<td>Simple, fast puts and gets, massively scalable</td>
</tr>
<tr>
<td><strong>Document</strong></td>
<td>Fast development, schemaless JSON/XML, searchable</td>
</tr>
<tr>
<td><strong>Graph/RDF</strong></td>
<td>Modeling anything at runtime including relationships</td>
</tr>
</tbody>
</table>
What model bests fits your next project?

Thoughts?

- Technical requirements
- Performance considerations
- Scalability needs
- Future growth potential

Relational
Flexible Queries

- Hospital
- Surgeon
- Operation

Dimensional
Data Warehousing

- Hospital Dimensions
- Drug Dimensions
- Operation Dimensions

Document
Easy Development

<table>
<thead>
<tr>
<th>Hospital</th>
<th>Operation Types</th>
<th>Operation ID</th>
<th>Operation Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospital</td>
<td>surgeons</td>
<td>Surgeon</td>
<td>Surgeon ID</td>
</tr>
<tr>
<td>Hospital</td>
<td>operations</td>
<td>Operation</td>
<td>Operation Type</td>
</tr>
</tbody>
</table>

Column/Key-value
Fast Puts and Gets

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Hospital</th>
<th>Drug</th>
<th>Drug ID</th>
<th>Drug Name</th>
<th>Drug Type</th>
<th>Drug Class</th>
<th>Drug Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td>Hospital</td>
<td>Drug</td>
<td>Drug ID</td>
<td>Drug Name</td>
<td>Drug Type</td>
<td>Drug Class</td>
<td>Drug Effect</td>
</tr>
<tr>
<td>2</td>
<td>Quantity</td>
<td>Hospital</td>
<td>Drug</td>
<td>Drug ID</td>
<td>Drug Name</td>
<td>Drug Type</td>
<td>Drug Class</td>
<td>Drug Effect</td>
</tr>
<tr>
<td>3</td>
<td>Unit</td>
<td>Hospital</td>
<td>Drug</td>
<td>Drug ID</td>
<td>Drug Name</td>
<td>Drug Type</td>
<td>Drug Class</td>
<td>Drug Effect</td>
</tr>
</tbody>
</table>
Agenda

- Defining NoSQL and Big Data
- Modeling Paradigms
- Optimizing for Velocity or Volume
- Optimizing for Availability or Consistency
- Summary
Once you have a list of NoSQL databases that meet your modeling needs, choose the one that best meets your need for velocity and volume.
What’s wrong with SQL DB’s?

• **Velocity**
  – SQL DBs are serialized to ensure consistency and use high latency disk, and this prevents them from scaling horizontally

• **Volume**
  – Because SQL DBs share cores, caches, and storage, they are serialized across these resources, and this prevents them from scaling horizontally
# Storage Cost/Performance

Determines Physical Architecture

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Technology</th>
<th>Persistent</th>
<th>Capacity</th>
<th>Inexpensive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume</strong></td>
<td>Hard Drives</td>
<td>Y</td>
<td>High</td>
<td>Capacity, Bandwidth</td>
</tr>
<tr>
<td><strong>Velocity</strong></td>
<td>RAM</td>
<td>N</td>
<td>Low</td>
<td>IOPs, Latency, Bandwidth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technology</th>
<th>GB / Blade</th>
<th>Latency (µS)</th>
<th>Bandwidth (MB/s)</th>
<th>IOPs (1000/s)</th>
<th>Cost / GB</th>
<th>Cost / MB/s</th>
<th>Cost / IOPs</th>
<th>Cost / Blade</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID 0 w/2 HDDs</td>
<td>1,200</td>
<td>4,850</td>
<td>288</td>
<td>2.4</td>
<td>1.25</td>
<td>5.21</td>
<td>0.63</td>
<td>1.5K</td>
</tr>
<tr>
<td>Flash SSDs</td>
<td>800</td>
<td>123</td>
<td>1,308</td>
<td>29</td>
<td>31.25</td>
<td>19.12</td>
<td>0.86</td>
<td>25.0K</td>
</tr>
<tr>
<td>Flash PCIe SLC</td>
<td>320</td>
<td>26</td>
<td>1,200</td>
<td>238</td>
<td>53.13</td>
<td>14.17</td>
<td>0.07</td>
<td>17.0K</td>
</tr>
<tr>
<td>RAM DDR3-1333 DR (12x16GB)</td>
<td>192</td>
<td>0.03</td>
<td>5,333</td>
<td>666,500</td>
<td>65.10</td>
<td>2.34</td>
<td>0.00</td>
<td>12.5K</td>
</tr>
<tr>
<td>RAM DDR3-1333 SR (10x4GB)</td>
<td>40</td>
<td>0.02</td>
<td>10,666</td>
<td>1,333,000</td>
<td>33.43</td>
<td>0.13</td>
<td>0.00</td>
<td>1.3K</td>
</tr>
</tbody>
</table>
Problem: **Serialized DB Design**

SQL DBs use serialization and synchronization for consistency

- Loading data from disk to RAM: 24%
- Latching: 24%
- Locking: 24%
- Recovery: 24%
- Locking rows of data
- Locking buffered data
- Data Processing: 4%
- Logging data

Michael Stonebraker © 2011 NoSQL Conference San Jose
Cost of Synchronization

*Single threaded operations are 746 times faster than multiple threads with locks*

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single thread</td>
<td>300</td>
</tr>
<tr>
<td>Single thread with memory barrier</td>
<td>4,700</td>
</tr>
<tr>
<td>Single thread with CAS</td>
<td>5,700</td>
</tr>
<tr>
<td>Single thread with lock</td>
<td>10,000</td>
</tr>
<tr>
<td>Two threads with CAS</td>
<td>30,000</td>
</tr>
<tr>
<td>Two threads with lock</td>
<td>224,000</td>
</tr>
</tbody>
</table>

*Slide from James Gates SORT presentation, “When Performance Really Matters”*
Future is *More* Cores — Not Faster

- **Exponential Transistors**
- **Flat Clock**
  (faster clock needs more power)
- **Flat Power**
  (power is expensive and hot)
- **Flat ILP**
  (instruction-level parallelism)
Hardware Takeaways

*Velocity and Volume*

Choose a NoSQL database that

– Scales horizontally for maximum parallel processing

– Lets you choose the right mix of *synchronous* and *asynchronous* transactions

– Leverages RAM when you need maximum *velocity* (low latency)

– Leverages disk when you need massive *volume* (high bandwidth)
## Typical Database Velocity

<table>
<thead>
<tr>
<th>Volume Per Day</th>
<th>Real-world 1K Transactions Per Day</th>
<th>Real-world 1K Transactions Per Second</th>
<th>Relational</th>
<th>Key Value</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 GB</td>
<td>8,640,000</td>
<td>100</td>
<td>Simply installed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86 GB</td>
<td>86,400,000</td>
<td>1,000</td>
<td>*Tuned</td>
<td>Simply Installed</td>
<td></td>
</tr>
<tr>
<td>432 GB</td>
<td>432,000,000</td>
<td>5,000</td>
<td>Hardware Optimized</td>
<td>Simply Installed *Tuned</td>
<td></td>
</tr>
<tr>
<td>864 GB</td>
<td>864,000,000</td>
<td>10,000</td>
<td>MPP Cluster</td>
<td>*Tuned</td>
<td>Hardware Optimized</td>
</tr>
<tr>
<td>8,640 GB</td>
<td>8,640,000,000</td>
<td>100,000</td>
<td>Horizontally Clustered</td>
<td>Horizontally Clustered</td>
<td></td>
</tr>
</tbody>
</table>

*Tuned means tuning the model and tuning queries
What velocity and volume do you need?

Thoughts?

<table>
<thead>
<tr>
<th>Volume Per Day</th>
<th>Real-world 1K Transactions Per Day</th>
<th>Real-world 1K Transactions Per Second</th>
<th>Relational</th>
<th>Key Value</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 GB</td>
<td>8,640,000</td>
<td>100</td>
<td>Simply installed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86 GB</td>
<td>86,400,000</td>
<td>1,000</td>
<td>Tuned</td>
<td>Simply Installed</td>
<td></td>
</tr>
<tr>
<td>432 GB</td>
<td>432,000,000</td>
<td>5,000</td>
<td>Hardware Optimized</td>
<td>Simply Installed</td>
<td>Tuned</td>
</tr>
<tr>
<td>864 GB</td>
<td>864,000,000</td>
<td>10,000</td>
<td>MPP Cluster</td>
<td>Tuned</td>
<td>Hardware Optimized</td>
</tr>
<tr>
<td>8,640 GB</td>
<td>8,640,000,000</td>
<td>100,000</td>
<td>Horizontally Clustered</td>
<td>Horizontally Clustered</td>
<td></td>
</tr>
</tbody>
</table>
### Databases (Ranked by Popularity)

#### Graph/RDF
- #22 Neo4j
- #35 MarkLogic
- #66 OrientDB
- #101 AllegroGraph
- #148 InfiniteGraph

#### Document
- #5 MongoDB
- #21 CouchDB
- #24 Couchbase
- #35 MarkLogic
- #65 Cloudant
- #66 OrientDB

#### Column
- #10 Cassandra
- #11 Redis
- #15 Hbase
- #18 Memcached
- #84 Aerospike
- #30 Riak
- #32 DynamoDB
- #56 Accumulo
- #71 Oracle NoSQL
- #127 FoundationDB

#### newSQL
- #60 Oracle x10,
- #62 GemFire
- #81 VoltDB
- #108 Clustrix

#### Live Analytics
- #1 Oracle Exalytics
- #23 SAP HANA
- #99 memsql

#### Doc Warehouse
- #12 Solr
- #16 ElasticSearch
- #33 Sphinx
- #35 MarkLogic

#### Data Warehouse
- #1 Oracle DB
- #2 MySQL
- #3 SQL Server
- #4 PostgreSQL
- #6 DB2
- #9 Sybase ASE
- #17 Informix
- #25 Firebird

#### Raw
- #19 Hadoop
- #20 Splunk

#### Raw Big Data
- #19 Hadoop
- #20 Splunk

#### Less structure (schemaless)
- Raw
- Graph
- Document
- Key-value/Column

#### More structure (schema)
- Relational
- Dimensional
Agenda

• Defining NoSQL and Big Data
• Modeling Paradigms
• Optimizing for Velocity or Volume
• Optimizing for Availability or Consistency
• Summary
Do I want multiple data centers to have consistent data immediately or eventually?

Do I want reads and/or writes to be available when data centers are down?

noSQL lets you choose per cluster and/or per transaction
Brewer’s CAP Theorem

If we want to maintain performance, the more we partition communications (i.e. scale horizontally), the more we have to choose between consistency and availability.

Consistency
- Consistent data requires synchronous transactions across boundaries

Availability
- High availability requires asynchronous transactions across boundaries

Partitioning, parallelization, performance
- Partitioning occurs as we scale horizontally (i.e. run in parallel)
- This occurs when a transaction exceeds the boundaries of one CPU core and extends to other cores, CPUs, servers, and data centers
- Partitioning increases the latency of communications, which slows the performance of synchronous communications
- To maintain performance as we increase partitioning, we have to run transactions asynchronously across data that is sharded
SQL scales vertically to be available and consistent. NoSQL scales horizontally. Availability scales better horizontally. Consistency scales better vertically. Scale Vertically: One CPU Core, Multiple Cores, Multiple CPUs, Clustered Servers, Many Data Centers. Scale Horizontally: New SQL, No SQL.
As communications become less performant and less reliable, we have to compromise and innovate on consistency and availability.
## CAP Tradeoffs

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ensure consistency</strong></td>
<td><strong>Enable parallelism</strong></td>
</tr>
<tr>
<td>by serializing changes</td>
<td>by eliminating serialization</td>
</tr>
<tr>
<td><strong>Prevent conflicts</strong></td>
<td><strong>Embrace conflicts</strong></td>
</tr>
<tr>
<td>by using ACID trans, two-</td>
<td>by supporting eventual</td>
</tr>
<tr>
<td>phase commit, nested</td>
<td>consistency &amp; conflict resolution</td>
</tr>
<tr>
<td>transactions</td>
<td></td>
</tr>
<tr>
<td><strong>Scale vertically</strong></td>
<td><strong>Scale horizontally</strong></td>
</tr>
<tr>
<td>for lowest latency &amp; fastest</td>
<td>for maximum parallelism &amp;</td>
</tr>
<tr>
<td>consistent changes</td>
<td>fastest concurrent changes</td>
</tr>
<tr>
<td><strong>Multi-threaded</strong></td>
<td><strong>Single-threaded</strong></td>
</tr>
<tr>
<td>to enable high concurrency</td>
<td>so locks cannot occur and</td>
</tr>
<tr>
<td>against consistent data</td>
<td>stop parallelism</td>
</tr>
<tr>
<td><strong>Latches and locks</strong></td>
<td>** Queues and MVCC**</td>
</tr>
<tr>
<td></td>
<td>(Multiversion Concurrency Control)</td>
</tr>
<tr>
<td><strong>Many cores per DB engine</strong></td>
<td><strong>One DB engine per core</strong></td>
</tr>
<tr>
<td><strong>Share data</strong></td>
<td><strong>Replicate data</strong></td>
</tr>
<tr>
<td>across cluster – not</td>
<td>across cluster – not shared</td>
</tr>
<tr>
<td>replicated</td>
<td></td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td><strong>Asynchronous</strong></td>
</tr>
<tr>
<td>multi-master replication</td>
<td>multi-master replication across</td>
</tr>
<tr>
<td>across data centers</td>
<td>data centers</td>
</tr>
</tbody>
</table>

---

CAP Tradeoffs: Consistency vs. Availability

- Ensure consistency: by serializing changes
- Prevent conflicts: by using ACID trans, two-phase commit, nested transactions
- Scale vertically: for lowest latency & fastest consistent changes
- Multi-threaded: to enable high concurrency against consistent data
- Latches and locks: Queue and MVCC (Multiversion Concurrency Control)
- Many cores per DB engine: One DB engine per core
- Share data: across cluster – not replicated
- Synchronous: multi-master replication across data centers
- Enable parallelism: by eliminating serialization
- Embrace conflicts: by supporting eventual consistency & conflict resolution
- Scale horizontally: for maximum parallelism & fastest concurrent changes
- Single-threaded: so locks cannot occur and stop parallelism
- Replicate data: across cluster – not shared
- Asynchronous: multi-master replication across data centers
ACID vs. BASE

Atomic
Consistent
Isolated
Durable

Basically
Available
Soft state
Eventual consistency

H₂SO₄
Sulfuric Acid

NaOH
Sodium Hydroxide

H₂SO₄
Sulfuric Acid

NaOH
Sodium Hydroxide
ACID vs. BASE

• **ACID transactions** between two nodes (within or between clusters) require a **two-phase commit**, which tightly couples them and, thus, reduces availability, but ensures **immediate consistency**
  1. Transaction coordinator has each node pre-commit the transaction and indicate if the commit is possible
  2. If both nodes agree that a commit is possible, the transaction coordinator asks both nodes to apply the commit
  3. If either or both nodes veto the commit, the transaction coordinator asks both nodes to roll back the commit

• **BASE transactions** between two nodes (within or between clusters) are **asynchronously decoupled, eventually consistent**, reliable, and have **higher availability**
  1. Use an asynchronous, guaranteed delivery, ordered message queue
  2. Add a log table to target database to track successful execution of queue messages
  3. Entries into the log table occur when messages are successfully executed in the target database
  4. Messages in the queue are dequeued only after the log confirms they have been executed
What’s wrong with NoSQL?

• In most NoSQL solutions, the developer is responsible for ensuring consistency
  – Imagine programming an app to coordinate thousands of concurrent threads across gigabytes of data structures
  – Imagine writing code to handle all threading issues
    • Locks
    • Contention
    • Serialization
    • Dead locks
    • Race conditions
    • Threading bugs
    • Etc
What are ACID transactions?

Relational databases use the ACID model to make it easy, reliable, and fast for concurrent processes to query and modify the same data consistently

Some NoSQL databases use the ACID model

- **Atomic**
  - All parts of a transaction succeed, or all fail and roll back

- **Consistent**
  - All committed data must be consistent with all data rules including constraints, triggers, cascades, atomicity, isolation, and durability

- **Isolated**
  - No transaction can interfere with other concurrent transactions

- **Durable**
  - Once a transaction is committed, data will survive system failures, and can be recovered after an unwanted deletion
What is Durability?

Durable data survives system failures
- Once a transaction has committed, its data is guaranteed to survive system failures
  - Failures may occur in the server, operating system, disk, and database.
  - Failures may be caused by server crash, full disk, corrupted disk, power outages, etc.
- Durability requires storage to be redundant.
- Durability requires logs to replay asynchronously written data.
- Durability requires logs to be archived to another location so they can be recovered.
- Durability works with atomicity to ensure that partially written data is not durable.
- Without durability you can have faster inserts, updates, and deletes because you have no logs to write and you can store data in volatile memory while lazily writing it to disk

Durable data can be recovered after unwanted deletion
- Durability allows data to be recovered to a point before the system failed or before applications or users inappropriately destroyed or modified data
- Durability requires backups of data
- Backups need to be able to recover data to a point in time
How much durability do you need?

- Can you live with writing **advanced code** to compensate?
  - Are you willing to trust all developers to properly check for partial transaction failures, properly detect current physical layout of the database cluster, and write extra code to properly propagate data durably across the cluster?

- Can you live with **lost data**?
  - Can you live with data lost because the database doesn’t use logs, doesn’t archive it logs, doesn’t use mirrored disks, and doesn’t require data to be written to multiple servers?

- Can you live with **accidental deletion of data**?
  - Can you live without a point-in-time recovery feature that can restore data immediately before an accidental deletion?

- Can you live with **scripting your own** backup & recovery solution?
  - Are you willing to develop custom scripts to back up and restore data files and database configurations?
What is Atomicity?

An atomic transaction is *all or nothing*

- Atomicity means all parts of a transaction succeed or nothing does
- This requires partially written data to be automatically removed from disk.
- A single command or a set of commands is called a transaction
- A single command to a programmer often represents multiple commands to the database because the database needs to replicate data to multiple disks, to update indexes, to execute triggers, to verify constraints, and to cascade deletes and updates, etc.
- Without atomicity, you can have faster transactions because they don’t need two-phase commit

Sets of data

- An operation may need to process multiple data items
- All data in the set needs to be changed or none; otherwise it becomes arbitrarily inconsistent
- For example, you want to delete part of a set of data and part way through the transaction fails. If all the changes are not automatically rolled back, the data is left in an inconsistent state that can affect the results of other transactions.

Sets of commands

- A series of commands often needs to work as a unit to produce accurate results
- All commands need to succeed or all need to fail; otherwise data becomes arbitrarily inconsistent
- Inconsistent data is hard to fix: data may contradict other data, and there may be extra data or missing data. Without atomicity to roll back failures, there may be no way to fix the data.
- The classic example is where you need to debit one account and credit another as a single transaction. If one command succeeds and the other fails, account totals are inaccurate.
How much Atomicity do you need?

- Can you live with **modifying single documents at a time**?
  - Without atomicity, you can’t guarantee results when working with multiple documents at a time

- Can you live with **partially successful transactions**?
  - Without atomicity, you can have higher availability because transactions can partially succeed
  - For example, rather than losing a sale when an app failed, you can save what you captured and later contact the customer to finish the transaction

- Can you live with **inconsistent and incomplete data**?
  - Without atomicity, sets of data or sets of commands may fail before being processed completely leaving data inconsistent or incomplete
  - Is it OK not to know when data anomalies are caused by bugs in your code or are temporarily inconsistent because they haven’t yet been synchronized?

- Can you live with writing **advanced code** to compensate?
  - Are you willing to develop custom solutions to provide atomic rollback, such as transaction logs, version numbers, and two-phase commits?
  - Are you willing to program defensively to handle transactions that fail?
  - Are you willing to develop processes to find and fix inconsistent data?
What is Isolation?

Isolation prevents concurrent transactions from affecting each other

- **Read isolation** ensures queries return accurate results as of a point in time
- **Write isolation** locks data to prevent race conditions during updates, deletes, and inserts
- Without isolation, queries and transactions run faster because the database doesn’t have to provide a consistent view using locks, snapshots, or system version numbers

- **Sets of data**
  - An operation can only produce accurate results as of a **point in time**
    - It takes time for a command to process a set of data
    - During this time, concurrent transactions may insert, update, and delete data in the set
    - Without isolation, a single command executes against data while it is being changed by other concurrent transactions.
    - Records may be added after the command started running
    - Records may be deleted or changed after the command has processed them.
    - This creates inconsistent results: aggregate functions produce wrong answers

- **Sets of commands**
  - A series of commands need to work on a consistent view of data to produce accurate results
    - Without isolation, each command in a series will execute against **arbitrarily different** data
How much Isolation do you need?

- Can you live with **processing single documents at a time**?
  - This is a very limited way to guarantee consistent, isolated results

- Can you live with **inaccurate queries**?
  - Without isolation, **query results are inaccurate** because concurrent transactions change data while processing it

- Can you live with **race conditions** and **dead locks**?
  - Without isolation, transactions can create **race conditions** and **deadlocks**

- Can you live with writing **advanced code** to compensate?
  - Are you willing to implement your own versioning system (like MVCC) that provides a reliable system change number spanning multiple data centers?
  - Are you willing to write code to hide concurrent updates, inserts and deletes from queries?
  - Are you willing to write code to handle update conflicts, race conditions and deadlocks?
What is Consistency?

- All committed data must be consistent with all data rules
  - Constraints, triggers, cascades, atomicity, isolation, and durability

- Data must always be in a consistent state at any point in time

- Consistency is the product of atomicity, isolation, and durability
  - Atomicity ensures that if data rules are violated, such as constraints and triggers, the transaction fails and all changes are rolled back.
  - Isolation ensures a query sees a consistent set of data while other concurrent commands are modifying the underlying data
  - Isolation ensures bulk updates lock sets of data so they can be processed as a consistent unit without other concurrent commands modifying their data.
  - Durability ensures that data is consistently replicated to other nodes in a cluster so a loss of a node won’t cause a loss of data

Consistency is the last refuge of the unimaginative.
— Oscar Wilde
Do you need complete consistency?

Not necessarily — instead, you may prefer

- Absolute fastest performance at lowest hardware cost
- Highest global data availability at lowest hardware cost
- Working with one document at a time
- Writing advanced code to create your own consistency model
- Eventually consistent data
- Some inconsistent data that can’t be reconciled
- Some missing data that can’t be recovered
- Some inconsistent query results

Consistency is the last refuge of the unimaginative. — Oscar Wilde
What do you need most?

- **Highest performance** for queries and transactions
- **Highest data availability** across multiple data centers
- **Less data loss** (i.e. durability)
- **More query accuracy & less deadlocks** (i.e. isolation)
- **More data integrity** (i.e. atomicity)
- **Less code** to compensate for lack of ACID compliance
Agenda

• Defining NoSQL and Big Data
• Modeling Paradigms
• Optimizing for Velocity or Volume
• Optimizing for Availability or Consistency
• Summary
Five Data Paradigms

Relational
- Flexible Queries

Dimensional
- Data Warehousing

Column/Key-value
- Fast Puts and Gets

Document
- Easy Development

Graph
- Unlimited Relationships
When will NoSQL be Enterprise Ready?

- In-memory DB
- NoSQL
- MapReduce
- DB as a Service
- Open Source DBs
- Columnar DW
- DB Appliances
- Oracle SQL Server DB2

Technology Trigger | Inflated Expectations | Disillusionment | Enlightenment | Productivity

- Enterprise Ready
- 1 to 5 years
- 5 to 10 years

Derived from Gartner Hype Cycle for Data Management
Modeling for NoSQL and SQL Databases

2014 NoSQL Now

by Michael Bowers
2014-08-13
v. 3.1
Databases (Ranked by Popularity)

- Raw
  - #5 MongoDB
  - #21 CouchDB
  - #24 Couchbase
  - #35 MarkLogic
  - #85 Cloudant
  - #86 OrientDB

- Graph
  - #12 Solr
  - #16 ElasticSearch
  - #33 Sphinx
  - #35 MarkLogic

- Document
  - JSON
  - #5 MongoDB
  - #21 CouchDB
  - #24 Couchbase
  - #35 MarkLogic
  - #85 Cloudant
  - #86 OrientDB

- Column
  - key/val
  - #10 Cassandra
  - #11 Redis
  - #15 Hbase
  - #18 MemcacheDB
  - #84 Aerospike
  - #36 Riak
  - #32 DynamoDB
  - #56 Accumulo
  - #71 Oracle NoSQL
  - #127 FoundationDB

- newSQL
  - #60 Oracle x10
  - #62 GemFire
  - #81 VoltDB
  - #108 Clustrix

- Live Analytics
  - #1 Oracle Exalytics
  - #23 SAP HANA
  - #99 memsql

- Data Warehouse
  - #1 Oracle DB
  - #2 MySQL
  - #3 SQL Server
  - #4 PostgreSQL
  - #6 DB2
  - #9 Sybase ASE
  - #17 Informix
  - #25 Firebird

- Doc Warehouse
  - XML
  - #12 Solr
  - #16 ElasticSearch
  - #33 Sphinx
  - #35 MarkLogic

- Raw Big Data
  - #10 Cassandra
  - #11 Redis
  - #15 Hbase
  - #18 MemcacheDB
  - #84 Aerospike
  - #36 Riak
  - #32 DynamoDB
  - #56 Accumulo
  - #71 Oracle NoSQL
  - #127 FoundationDB

- More structure (schema)
- Less structure (schemaless)

Hospital Name: John Hopkins
Operation Number: 13
Operation Type: Heart Transplant
Surgeon Name: Dorothy Oz
Drug Name
Drug Manufacturer
Dose
Size
Dose UOM
Minicillan Drug
Rs Us 200 mg
Maxicillan Canada 4Less Drugs 400 mg
Minicillan Drug USA 150 mg

- JSON Document
  - #5 MongoDB
  - #21 CouchDB
  - #24 Couchbase
  - #35 MarkLogic
  - #85 Cloudant
  - #86 OrientDB

- XML Document
  - #12 Solr
  - #16 ElasticSearch
  - #33 Sphinx
  - #35 MarkLogic

- Less structure (schemaless)
- More structure (schema)