



The Role of Data Architecture in NOSQL

**DAMA - Philadelphia / Delaware Valley
Wednesday, January 11th, 2012**

Tom Haughey
President
InfoModel LLC
868 Woodfield Road
Franklin Lakes, NJ 07417
201 755 3350
thaughey@gmail.com



What Advances Occurred in DBMSs?

- The emergence of NOSQL data stores and new DBMSs
- Column oriented DBMSs
- Data warehouse appliances
- New database technologies, such as business intelligence DBMSs
- Re-emergence of the Enterprise Data Warehouse





Why NOSQL?

- Stands for **Not Only SQL** or even **Not SQL**
- RDBMSs (they say) have shown poor performance on data-intensive applications, including:
 - Indexing a large number of documents
 - Serving pages on high-traffic websites
 - Handling the volumes of social networking data
 - Delivering streaming media
- Typical RDBMS implementations (they say) are tuned for small but frequent read/write transactions or for large batch transactions with rare write access
- NOSQL (they also say) can service heavy read/write workloads (yes but for certain types of workloads, which they don't say)
- Real-world NOSQL deployments include:
 - Digg's 3 TB for green badges (markers that indicate stories upvoted by others in a social network)
 - Facebook's 50 TB for inbox search
 - Google uses BigTable in over 60 applications such as Google Earth, Orkut



NOSQL Characteristics

- "NOSQL has nothing to do with SQL"
 - (Michael Stonebraker)
- Characteristics of NOSQL data stores are:
 - Often, file management system, not DBMS
 - Large volumes of data
 - Non-relational and no support for joins
 - Improved performance for large data sets
 - Distributed databases and queries
 - Fault tolerance so that an application will continue even if some connection is lost
 - Horizontally scalable using component nodes
 - Schema-free (so they have a more flexible structure)
 - Eventually consistent (ACID-free)
 - Easy to replicate so as to improve availability
 - Access via API support or other non-SQL interfaces
 - Open-source



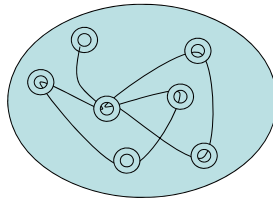


Four New NOSQL Data Store Types

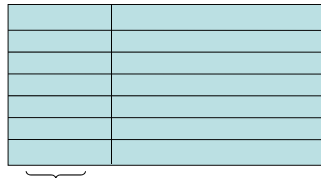
Key Value •Distributed Hash Table

Key	Value
1	Bob
2	Sue
3	Joe
4	Jo

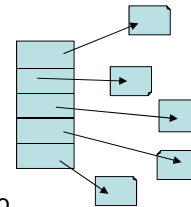
Graph •Graph Theory



Column Based •Semi-structured



Document •Semi-structured



Actually, there are now more DBMS *types*
than there used to be DBMS *instances* 10-years ago



NOSQL Assumptions

- NOSQL DBMS further utilize one or more of three assumptions:
 - The database will be big enough that it should be scaled across multiple servers.
 - The application should run well if the database is replicated across multiple geographically distributed data centers, even if the connection between them is temporarily lost.
 - The database should run well if the database is replicated across a host server and a bunch of occasionally-connected mobile devices.
- In addition, NOSQL proposes that a database should have no fixed schema, other than whatever emerges as a byproduct of the application-writing process.





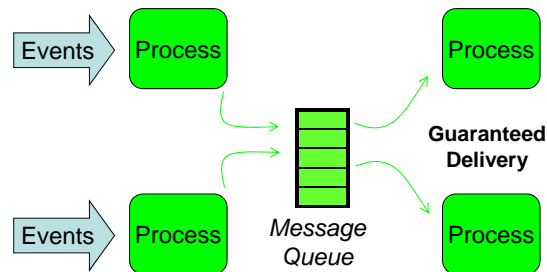
CAP Theorem

- Eric Brewer challenged the ACID properties of RDBMS
 - **Atomic** - do the whole transaction or nothing at all
 - **Consistent** – obey all integrity and business rules
 - **Isolated** – hide the results of a transaction in progress
 - **Durable** – once committed, the results are persisted
- He contends that a database cannot ensure all three of the following properties at once, called CAP:
 - **Consistency**: the client perceives that all of the operations have been performed at once (Do you agree with this definition?)
 - **Availability**: every operation must end in an intended response
 - **Partition tolerance**: operations will continue even if individual components are unavailable
- A database can ensure only any two-of-the-three (PICK TWO)
- Consequently, NOSQL proposes to replace the ACID properties of an RDBMS with CAP properties
- NOSQL emphasizes **eventual** consistency



Message Queuing

- Enables processes running at different times to communicate across heterogeneous networks and systems that may be temporarily offline
- Applications send messages to queues and read messages from queues



- Can be used for:
 - Mission-critical financial services
 - Embedded and hand-held applications
 - Outside sales
 - Workflow





Sharding for Horizontal Scalability

- NOSQL data stores distribute data via sharding
- Sharding is just another name for **horizontal partitioning**
 - Horizontal partitioning is a database implementation principle whereby groups of rows are kept on separate nodes
 - A database shard is a horizontal subdivision of a database
 - Each subdivision (partition) forms part of a shard, which may in turn be located on a separate database server or physical location
- Multiple shards can be placed on different machines
 - Facilitates distribution of data on separate, commodity hardware
 - Is used in most NOSQL data stores
 - Is easier for NOSQL to implement because there is no enforcement of integrity except in the application (Life is good!)



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

9



Sharding and Sharing

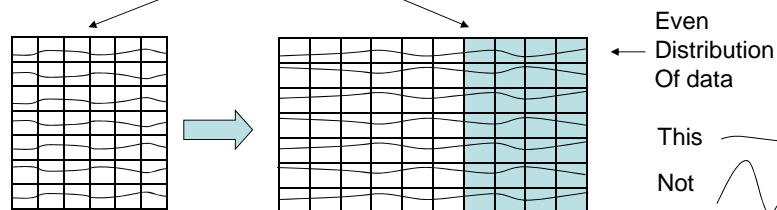
- Sharding uses the “shared nothing” model
 - Each server has its own data and processor
 - MPP (Massively Parallel Processing) used this model
- The functionality of “shared nothing” is:
 - A controlling node receives the query
 - It then sends the query to the other nodes
 - Each gets its data and sends it to the controlling node
 - The controlling node unifies or aggregates the result sets
- You can go from **this** to **this** just by adding more nodes



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

10



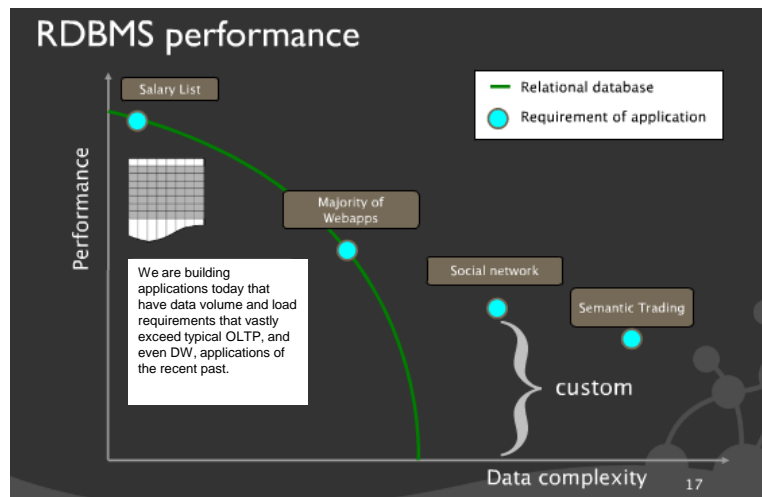


Advantages of Sharding

- The number of rows in each physical space is reduced
- This reduces index size, improving search performance
- Database activity can be spread out over multiple machines, greatly improving performance
- If a shard is based on some real-world segmentation of the data (e.g. European customers vs. American customers), only the relevant shard needs to be queried
- Two considerations in sharding can improve performance:
 - Even distribution of data to avoid peaks and valleys in instances
 - Collocation of related data (Customer, Order, Order Item all related by Customer ID)



NOSQL – RDBMS Comparison



Source: Tobias Ivarsson





Key Value Databases



Key-Value (KV) Pair

- A set of **two linked data items**:
 - A **key**, which is a unique identifier for some item of data, and
 - The **value**, which is either the data that is identified by the key or a pointer to the location of that data
- Often used in lookup tables, hash tables and configuration files
- Only way to query a key value store is by the key
- The keys could be the names of the objects, alphabetically sorted
- Data must be grouped into domains, which are sets (Sound familiar? It should. Entities/tables are sets.)
- Each object is usually replicated in several other stores for high availability

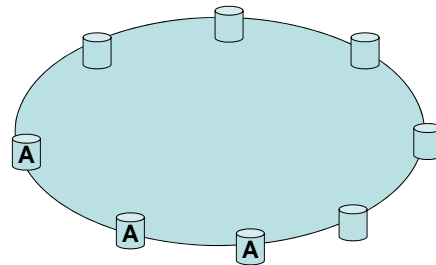




Examples

• Key value data store

Automobile	
Key	Attributes
1	Make: Toyota Model: Highlander Color: Maroon Year: 2004
2	Make: Toyota Model: Highlander Color: Blue Year: 2005 Transmission: Automatic



• Ring partitioning and replication



Characteristics of Key Value Stores

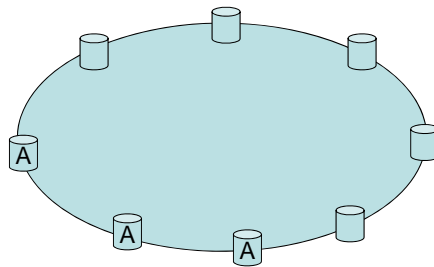
- Key can have dynamic set of attributes
- The attributes and their values are contained as a blob, usually in the format of a string
- No relationships are explicitly declared
- The activities of CRUD (create, read, update, delete) happen to the data via API's
- Integrity is maintained in the application code





Key Value Store Uses

- Though KV data store has been around for years, it would probably have a low profile now except for the Web and Cloud Computing
 - Scales to huge volumes of data
 - Handle massive load
 - Simple data model: key-value
 - Ring partitioning and replication



© InfoModel LLC, 2012

Advances in Data Modeling for DW

17



Amazon Requirements

- Amazon had these characteristics:
 - **Query**: simple read/write
 - **ACID**: requires high availability but weaker consistency
 - **Efficiency**: use commodity hardware
 - **Other**: non-hostile environment with no security issues
- To solve this they came up with Dynamo
 - Very simple to build a key value store, and very easy to scale
 - Usually good performance (for certain applications)
 - Schema-less



© InfoModel LLC, 2012

Advances in Data Modeling for DW

18



Amazon's Dynamo

- The most famous Key : Value data store
 - Available, scalable and distributed
 - Sample services are best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog
- Failure will happen
 - Millions of components
 - At any given moment, always a small but significant number of server and network components are failing
 - Must treat failure handling as the norm
- Data is partitioned & replicated for availability by hashing the key
 - Consistency facilitated by object versioning
 - Consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization
 - Nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution



Columnar Databases

A new idea or an age-old implementation?





Column Oriented Data Stores

- There really are two types of column oriented data stores:
 - (1) **Wide-column data stores** (they resist being called databases) used for web processing, streaming data, cloud computing and documents. These are NOSQL, such as Google BigTable clones.
 - (2) **Column-oriented databases** are inverted structures (which ARE databases) used for statistical analysis, querying and reporting (such as Vertica and Sybase IQ). These are NOT classified as NOSQL
- Back in the 1960's random access was supported by "fully inverted files"
 - Here's a quote from Joe Celko
 - "In an inverted file, every column in a table has an index on it. It is called an inverted file structure because columns become files."



Simple Comparison – Column vs. Row

Simple Example of Business Data

Empld	Lastname	Firstname	Salary
1	Smith	Joe	40000
2	Jones	Mary	50000
3	Johnson	Cathy	44000

A Row-oriented Database: puts rows together

1,Smith,Joe,40000; 2,Jones,Mary,50000; 3,Johnson,Cathy,44000;_

A Column-oriented Database puts values of a column together.

1,2,3; Smith,Jones,Johnson; Joe,Mary,Cathy; 40000,50000,44000;_





Row vs. Column Oriented

	Col 1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
Row 1								
Row 2								
Row 3								
Row 4								
Row 5								
Row 6								
Row 7								
Row8								

• In most RDBMSs, the entire row is read into a buffer and then sliced down to the attributes

• In columnar DBMSs, only the needed columns are read

	Col 1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
Row 1								
Row 2								
Row 3								
Row 4								
Row 5								
Row 6								
Row 7								
Row 8								



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

23



Characteristics of Columnar Databases

- The following characteristics describe Columnar DBMSs (Type 2 above)
- A storage unit (page or file) is assigned to a each column
 - Reading a few columns from every row is very fast
 - Some columnar databases claim they are significantly faster than a row-based database at querying
 - Columns are exhaustively indexed
 - Low cardinality columns may be supported by bitmap indexes
- Ideal for OLAP, statistical analysis and analytical reporting
- Unsuitable for OLTP
- The difference between row-based and columnar databases is only in the storage engine layer
- Queries still use straight SQL
- The optimizer passes the re-written query to the storage engine, which writes to / reads from either row-based or columnar based tables



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

24



Columnar Databases

- Michael Stonebraker (CTO of Vertica) says:
 - Column databases will take over the warehouse market
 - Many warehouses can't load in the available load window, can't support ad-hoc queries, can't get better performance without a "fork-lift" upgrade
 - Vertica beats all row stores typically by a factor of 50
- Columnar database systems are not new, witness Sybase IQ
- Columnar databases take advantage of exhaustive indexing of columns, bitmapping and data compression to improve performance
- They used to be considered a niche offering but no more
- Google's BigTable Data Store keeps data by columns and is great for what it does
 - But is it suitable for data warehousing?
 - Is it suitable for critical financial and medical applications?



Choosing Bitmapped Indexes

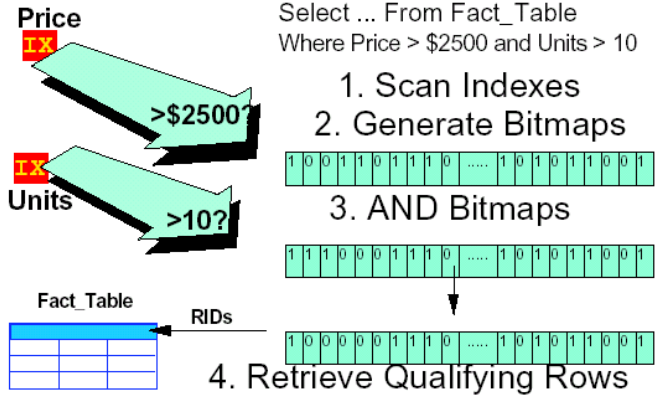
- Columnar (and relational) DBMSs often use bitmapped indexes
- These are valuable when the data and queries have the following characteristics:
 - Low cardinality - small number of distinct values
 - Low selectivity - # of distinct values / total # of values
 - ANDing or ORing of predicates is often done
 - An index is available
 - Queries with =, <, >, [NOT] EXISTS, UNIQUE, GROUP BY
 - Sometimes an optimizer determines it is cost effective (UDB)
- In some columns, such as amounts, that may not have low cardinality, consider value banding to reduce the number of distinct values
 - Value banding reduces infinite values to specific ranges that are meaningful to the business
 - Exemplified by income ranges, years of education ranges, net worth ranges, etc.





Bitmap Example (UDB)

Bitmap technology



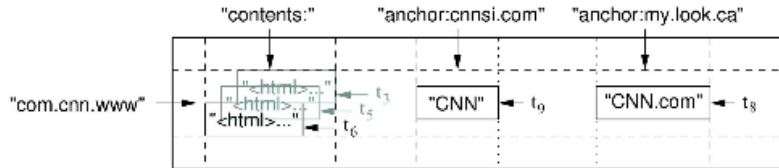
Wide Column Data Stores

Built by and for Google
But used in many BigTable clones



BigTable (Google)

- A **BigTable** instance contains a column or group of elements that are:
 - **Sparse** (rows only where there are values; not fixed length)
 - **Distributed** (horizontally partitioned over many servers)
 - **Persistent** (stored)
 - **Multidimensional** (consisting of a multi-part key)
 - **Sorted map** (in lexicographic order)
- A special variation of Key : Value data stores
 - Indexed by row key, column key, and a timestamp
 - Each value in the map is an uninterpreted array of bytes.
 - (row: string, column: string, time:int64) ->string



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

29

Wide-Column Data Store Column Structure *

Uniqueness

RowID	Column Family	Column Qualifier	Timestamp	Attribute Value
-------	---------------	------------------	-----------	-----------------

Column Key

Attribute	Meaning	Example
Row ID	Key of the instance. A string. Can be name or number. May be composite.	9019435035
Column Family	Groups data into sets (akin to entities/tables). All data stored in a family is related.	Company
Column Qualifier	Denominates or describes the stored data item (akin to attribute)	Gross Revenue Amount
Timestamp	Complete time of the instance	9/2/2011 9:05 AM
Data Value	The instance of data stored	\$2BB



* In my opinion, this structure has a strong resemblance to a denormalized Name-Value Pair

© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

30



BigTable Keys and Values

- Because these systems tend to be so huge and distributed, this sorted feature is actually very important. The spatial proximity of rows with like keys ensures that when you must scan the table, the items of greatest interest to you are near each other.
- This is important when choosing a row key and its structure. For example, consider a table whose keys are domain names. It makes the most sense to list them in reverse notation (so "com.jimbojw.www" rather than "www.jimbojw.com") so that rows about a subdomain will be near the parent domain row.
- Continuing the domain example, the row for the domain "mail.jimbojw.com" would be right next to the row for "www.jimbojw.com" rather than say "mail.xyz.com" which would happen if the keys were regular domain notation.
- This does not mean that "values" are sorted. There is no automatic indexing of anything other than the keys



So How Do You Actually Get Data?

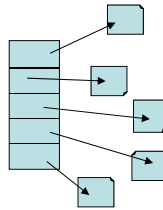
- In BigTable clones, you get data via Map:Reduce.
- MapReduce is a framework for processing huge datasets distributed over a large group of computers (nodes)
 - A cluster (if all nodes use the same hardware) or
 - A grid (if the nodes use different hardware).
- Achieved in two steps:
 - "Map" step: The master node takes the input, partitions it up into smaller sub-problems, and distributes those to worker nodes. The worker node processes that smaller problem, and passes the answer back to its master node.
 - "Reduce" step: The master node takes the answers to all the sub-problems and combines them to get the desired result
- The advantage of MapReduce is that it allows for distributed processing of the map and reduction operations
- Doesn't this sound similar to how Massively Parallel Processing (MPP) handles data retrieval? (It should because it is.)





Document Stores

- Have databases, collections, and indexes but not joins (~)
- Collections contain documents
 - They store unstructured (e.g., text) or semi-structured (e.g., XML) documents – basically hierarchical (1 : M)
 - Essentially, they support the embedding of documents and arrays within other documents and arrays
 - Document structures do not have to be predefined, so are schema-free (think about XML)
- These can be very valuable for storing data in data marts, such as salesperson sales or insurance agent sales for querying



© InfoModel LLC, 2012

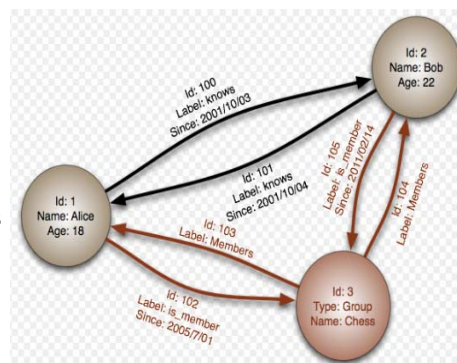
Role of Data Architecture in NOSQL

33



Graph Databases

- Employ nodes (like entities), properties (attributes), and edges (relationships)
- They say, faster for associative data sets – no joins
- Map more directly to the structure of OO apps
- Can scale to large data sets without joins
- Have a less a rigid schema than RDBMS so deal with structural change more easily
- RDBMSs are faster at set processing



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

34



Should You or Shouldn't You?

- You can use Key Value and Wide Column Data Stores as long as your company name contains one of the following words:
 - Amazon
 - Twitter
 - Flickr
 - Google
 - Yahoo
 - Facebook
 - Digg
 - Zynga
 - Best Buy
 - And some others



Are RDBMSs Doomed?

- RDBMSs are well suited to transaction processing, transaction analysis, set processing, flexible querying and have proven integrity enforcement
- Mission-critical, high performance applications such as brokerage trading, insurance and claims processing, consumer goods transaction processing, banking transactions, pharmaceuticals, will continue to be handled using RDBMSs





Should You or Shouldn't You?

- **Pure column data stores** (Type 1) such as BigTable and Dynamo clones are not suitable for:
 - The majority of DW/BI applications
 - Traditional mission-critical OLTP applications
 - Or even many traditional web-based applications
- **Pure column data stores** (Type 1), **Key/Value** data stores and other NOSQL data store are pertinent to:
 - Indexing a large number of documents
 - Serving pages on high-traffic websites
 - Delivering streaming media
 - Data as typically occurs in social networking applications
- **Columnar DBMSs** (Type 2) with their inverted structures, bitmapped indexes and compression are very applicable to DW/BI environments (but not OLTP) and must be seriously evaluated by every organization investing in BI



© InfoModel LLC, 2012

Role of Data Architecture in NOSQL

37



Database Comparison

	Performance	Scalability Of Volume	Flexibility Of Structure	Complexity Of DB	Base Model	Main Players
Key-Value Stores	high	high	high	none	variable but is an associative array	AmazonS3 Redis Memcached Voldemort
Wide Column Stores	high	high	moderate	low	key and column	Cassandra HBase Big Table & clones
Columnar DBMSs	high	high	high	high	Inverted file	Sybase IQ Vertica ParAccel InfoBright Aster Data
Document Stores	high	variable (high)	high	low	hierarchical	CouchDB MongoDB Cassandra
Graph Databases	variable	variable	high	high	graph theory	Neo4j FlockDB InfiniteGraph
Relational Databases	variable	variable	low in structure but high in joining	moderate	relational algebra	The Big Four For DW: Teradata & all DW appliances



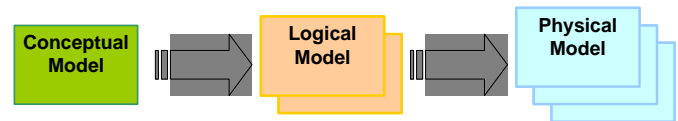


Remember the Purpose of Data Modeling

- Data modeling is the activity of **defining the information needs of an organization** by classifying the objects of interest, characterizing those objects, and interrelating them independent of technology and implementation
- The data model can have many purposes:
 - The classic use is to develop a database
 - Its primary purpose is to express data requirements
- Right now:
 - NOSQL has little or no use for Data Modeling
 - NOSQL developers generally do not create a logical model
 - If a Logical Data Model is created prior to a NOSQL implementation, the Logical Model and the Physical Model may differ – even substantially, especially due to the extreme denormalization and flattening that occurs within NOSQL
 - Nonetheless, the best way to understand the data requirement independent of implementation is to use data modeling



Levels of Models



Contains major business entities, major attributes and business relationships
Used as planning tool at a business level or initial project data model

Contains all the data entities, all attributes and all data relationships
A complete model is accurate, detailed and identifies all the required data. A good model offers a number of options for the physical implementation.

Contains table names, columns, indexes, constraints as well as partitioning
This is the implementation model, targeting a specific physical DBMS, e.g., DB2, Oracle, etc.

- These two levels should be unaffected by NOSQL and columnar databases
- Data modeling should be done as normal
- The difference in NOSQL et al. is at the physical level





Data Architecture (DA) and NOSQL

- NOSQL understands programming, sharding and building efficient stores, etc., whose characteristics we described earlier
- Many NOSQL users have a poor understanding of (and struggle with) some important and elementary DA concepts, e.g.:
 - Naming standards (the data is not just in your program)
 - Different types of data such as atomic attributes vs. structured attributes (e.g., NOSQL Columns vs. Supercolumns)
 - Principles of data management:
 - Functional dependency, uniqueness, granularity
 - Non-redundancy and integrity of data
 - Keys: candidate, natural, surrogate
 - The real characteristics of relational databases
- It's not enough to say "We're NOSQL. These are not important."
- Abandoning them isn't a trade-off. It's a brush-off.



Trust in Ourselves

- As professional data modelers, we must have trust in our science
- We treat data with respect and should ensure others do also
- We should also learn to be a bit quicker about it. Data modeling shouldn't take forever. Improved productivity is about tradeoffs.
- Is it more productive to iterate and iterate through data modeling until we get it right, or is it better to determine, this is good enough, build it and then refactor it?
- Think twice about putting into a data model something that is extraordinarily difficult to implement (or give guidelines)
- In implementing, use denormalization sensibly, guided by quantifiable factors
- Divide deliverables into increments
- Deliver in shorter increments
- Provide for early delivery of a prototype
- Assemble functional teams of motivated, skilled, practical people
- Timebox the work





Conclusions

- RDBMSs are here to stay because they have proven their value
- Remember, the paychecks you get, the bank accounts you have, the stocks you buy and sell, the pills you take, the auto insurance you have -- were all enabled by an RDBMS
- NOSQL data stores are suitable for certain web-oriented applications
- Most NOSQL DBMSs are unsuitable for the DW and BI querying
- Assess the value of column-based and other BI DBMSs
- Believe in the value of logical data modeling, regardless of the physical storage implementation, but just be quicker about it
- Streamline DBA functions so that refactoring is less painful
- Use a modified Agile approach for data modeling, especially in a DW and data mart environment
 - You'd have to be crazy not to be Agile
 - But just as crazy to do it in 2-week increments



That's My Story and I'm Sticking to It

