Highly Scalable Blog

ARTICLES ON BIG DATA, NOSQL, AND HIGHLY SCALABLE SOFTWARE ENGINEERING

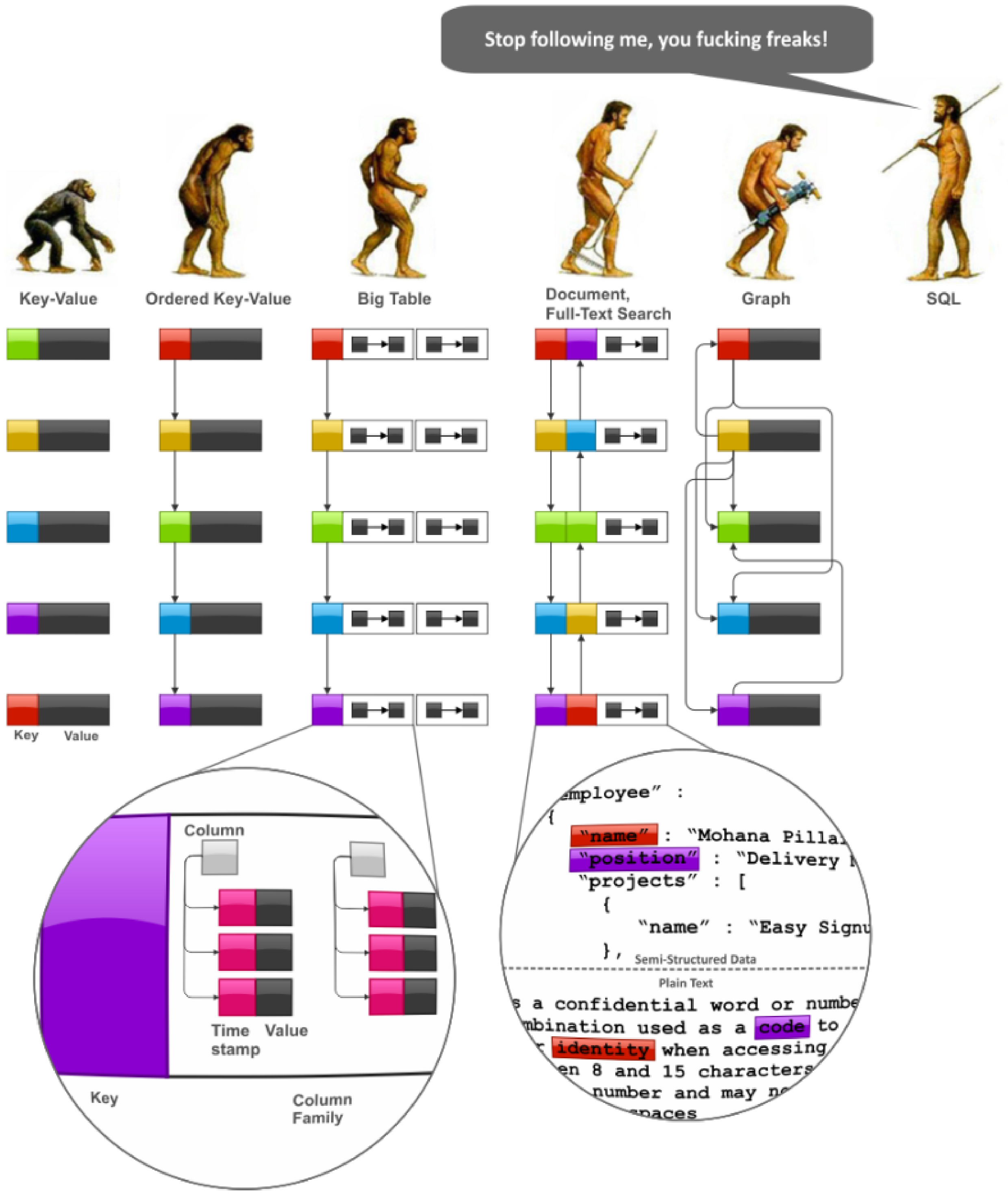
⚜ **EXTRAS**

NoSQL Data Modeling Techniques

NoSQL databases are often compared by various non-functional criteria, such as scalability, performance, and consistency. This aspect of NoSQL is well-studied both in practice and theory because specific non-functional properties are often the main justification for NoSQL usage and fundamental results on distributed systems like the CAP theorem (http://en.wikipedia.org/wiki/CAP_theorem) apply well to NoSQL systems. At the same time, NoSQL data modeling is not so well studied and lacks the systematic theory found in relational databases. In this article I provide a short comparison of NoSQL system families from the data modeling point of view and digest several common modeling techniques.

I would like to thank Daniel Kirkdorffer (http://www.kirkdorffer.com/) who reviewed the article and cleaned up the grammar.

To explore data modeling techniques, we have to start with a more or less systematic view of NoSQL data models that preferably reveals trends and interconnections. The following figure depicts imaginary "evolution" of the major NoSQL system families, namely, Key-Value stores, BigTable-style databases, Document databases, Full Text Search Engines, and Graph databases:



```
mployee" :
{
  "name" : "Mohana Pilla
  "position" : "Delivery M
  "projects" : [
    {
      "name" : "Easy Signu
    },
```

```
s a confidential word or numbe
mbination used as a code to
r identity when accessing
en 8 and 15 characters
number and may no
spaces
```

(https://highlyscalable.files.wordpress.com/2012/02/overview2.png)

NoSQL Data Models

First, we should note that SQL and relational model in general were designed long time ago to interact with the end user. This user-oriented nature had vast implications:

- The end user is often interested in aggregated reporting information, not in separate data items, and SQL pays a lot of attention to this aspect.
- No one can expect human users to explicitly control concurrency, integrity, consistency, or data type validity. That's why SQL pays a lot of attention to transactional guaranties, schemas,

and referential integrity.

On the other hand, it turned out that software applications are not so often interested in in-database aggregation and able to control, at least in many cases, integrity and validity themselves. Besides this, elimination of these features had an extremely important influence on the performance and scalability of the stores. And this was where a new evolution of data models began:

- Key-Value storage is a very simplistic, but very powerful model. Many techniques that are described below are perfectly applicable to this model.
- One of the most significant shortcomings of the Key-Value model is a poor applicability to cases that require processing of key ranges. Ordered Key-Value model overcomes this limitation and significantly improves aggregation capabilities.
- Ordered Key-Value model is very powerful, but it does not provide any framework for value modeling. In general, value modeling can be done by an application, but BigTable-style databases go further and model values as a map-of-maps-of-maps, namely, column families, columns, and timestamped versions.
- Document databases advance the BigTable model offering two significant improvements. The first one is values with schemes of arbitrary complexity, not just a map-of-maps. The second one is database-managed indexes, at least in some implementations. Full Text Search Engines can be considered a related species in the sense that they also offer flexible schema and automatic indexes. The main difference is that Document database group indexes by field names, as opposed to Search Engines that group indexes by field values. It is also worth noting that some Key-Value stores like Oracle Coherence gradually move towards Document databases via addition of indexes and in-database entry processors.
- Finally, Graph data models can be considered as a side branch of evolution that origins from the Ordered Key-Value models. Graph databases allow one model business entities very transparently (*this depends on that*), but hierarchical modeling techniques make other data models very competitive in this area too. Graph databases are related to Document databases because many implementations allow one model a value as a map or document.

General Notes on NoSQL Data Modeling

The rest of this article describes concrete data modeling techniques and patterns. As a preface, I would like to provide a few general notes on NoSQL data modeling:

- NoSQL data modeling often starts from the application-specific queries as opposed to relational modeling:
  - Relational modeling is typically driven by the structure of available data. The main design theme is "**What answers do I have?**"
  - NoSQL data modeling is typically driven by application-specific access patterns, i.e. the types of queries to be supported. The main design theme is **"What questions do I have?"**

- NoSQL data modeling often requires a deeper understanding of data structures and algorithms than relational database modeling does. In this article I describe several well-known data structures that are not specific for NoSQL, but are very useful in practical NoSQL modeling.
- Data duplication and denormalization are first-class citizens.
- Relational databases are not very convenient for hierarchical or graph-like data modeling and processing. Graph databases are obviously a perfect solution for this area, but actually most of NoSQL solutions are surprisingly strong for such problems. That is why the current article devotes a separate section to hierarchical data modeling.

Although data modeling techniques are basically implementation agnostic, this is a list of the particular systems that I had in mind while working on this article:

- Key-Value Stores: Oracle Coherence, Redis, Kyoto Cabinet
- BigTable-style Databases: Apache HBase, Apache Cassandra
- Document Databases: MongoDB, CouchDB
- Full Text Search Engines: Apache Lucene, Apache Solr
- Graph Databases: neo4j, FlockDB

Conceptual Techniques

This section is devoted to the basic principles of NoSQL data modeling.

(1) Denormalization

Denormalization can be defined as the copying of the same data into multiple documents or tables in order to simplify/optimize query processing or to fit the user's data into a particular data model. Most techniques described in this article leverage denormalization in one or another form.

In general, denormalization is helpful for the following trade-offs:

- *Query data volume* or *IO per query* VS *total data volume*. Using denormalization one can group all data that is needed to process a query in one place. This often means that for different query flows the same data will be accessed in different combinations. Hence we need to duplicate data, which increases total data volume.
- *Processing complexity* VS *total data volume*. Modeling-time normalization and consequent query-time joins obviously increase complexity of the query processor, especially in distributed systems. Denormalization allow one to store data in a query-friendly structure to simplify query processing.

Applicability: Key-Value Stores, Document Databases, BigTable-style Databases

(2) Aggregates

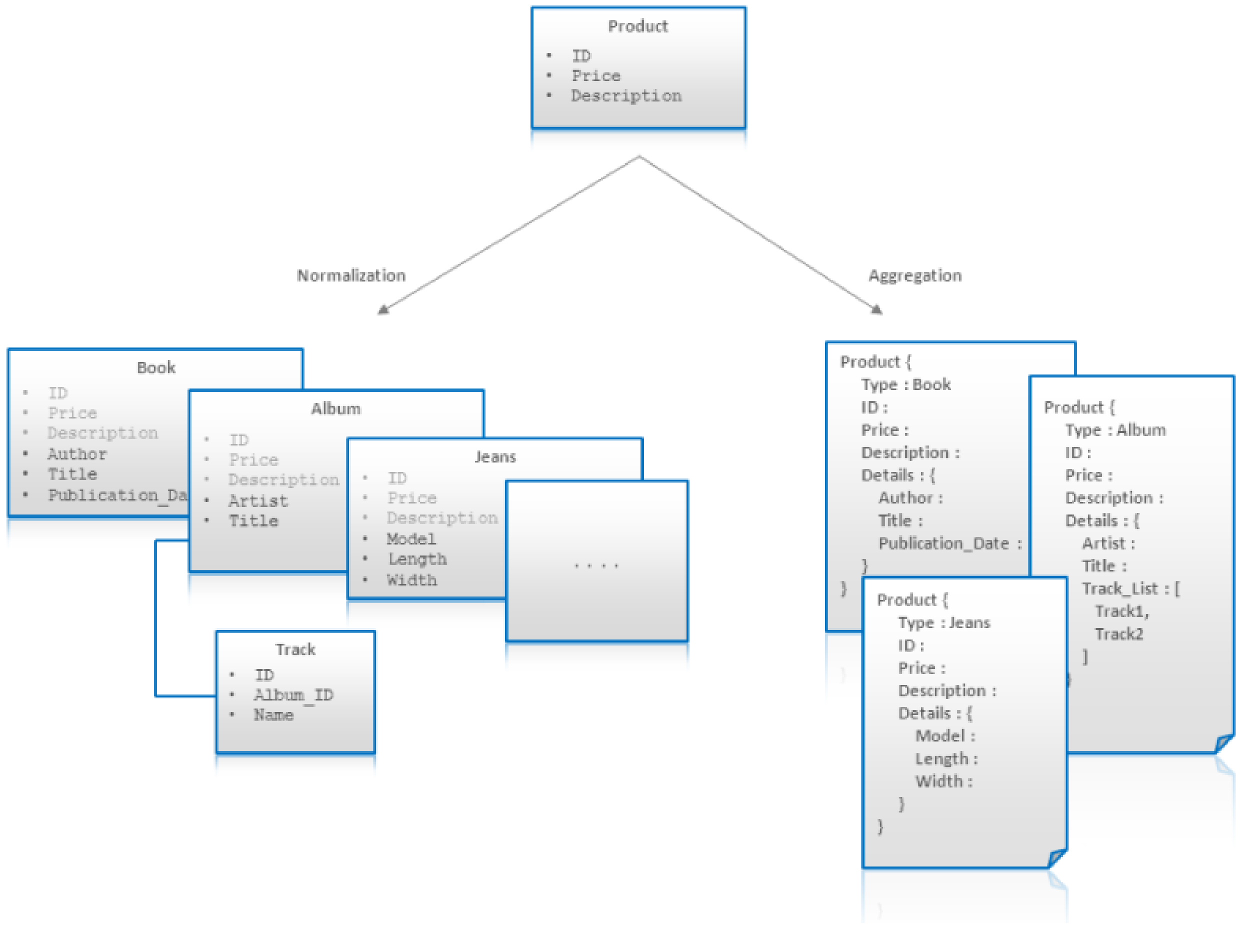
All major genres of NoSQL provide soft schema capabilities in one way or another:

- Key-Value Stores and Graph Databases typically do not place constraints on values, so values can be comprised of arbitrary format. It is also possible to vary a number of records for one business entity by using composite keys. For example, a user account can be modeled as a set of entries with composite keys like *UserID_name, UserID_email, UserID_messages* and so on. If a user has no email or messages then a corresponding entry is not recorded.
- BigTable models support soft schema via a variable set of columns within a *column family* and a variable number of *versions* for one *cell*.
- Document databases are inherently schema-less, although some of them allow one to validate incoming data using a user-defined schema.

Soft schema allows one to form classes of entities with complex internal structures (nested entities) and to vary the structure of particular entities.This feature provides two major facilities:

- Minimization of one-to-many relationships by means of nested entities and, consequently, reduction of joins.
- Masking of "technical" differences between business entities and modeling of heterogeneous business entities using one collection of documents or one table.

These facilities are illustrated in the figure below. This figure depicts modeling of a product entity for an eCommerce business domain. Initially, we can say that all products have an ID, Price, and Description. Next, we discover that different types of products have different attributes like Author for Book or Length for Jeans. Some of these attributes have a one-to-many or many-to-many nature like Tracks in Music Albums. Next, it is possible that some entities can not be modeled using fixed types at all. For example, Jeans attributes are not consistent across brands and specific for each manufacturer. It is possible to overcome all these issues in a relational normalized data model, but solutions are far from elegant. Soft schema allows one to use a single Aggregate (product) that can model all types of products and their attributes:


Product
ID
Price
Description
Normalization
Aggregation
Book
Author
Title
Publication_Da
Album
Artist
Jeans
Model
Length
Width
Track
Album_ID
Name
Product { Type : Book ID : Price : Description : Details : { Author : Title : Publication_Date : } }
Product { Type : Album ID : Price : Description : Details : { Artist : Title : Track_List : [Track1, Track2] } }
Product { Type : Jeans ID : Price : Description : Details : { Model : Length : Width : } }

(https://highlyscalable.files.wordpress.com/2012/02/soft-schema2.png)

Entity Aggregation

Embedding with denormalization can greatly impact updates both in performance and consistency, so special attention should be paid to update flows.

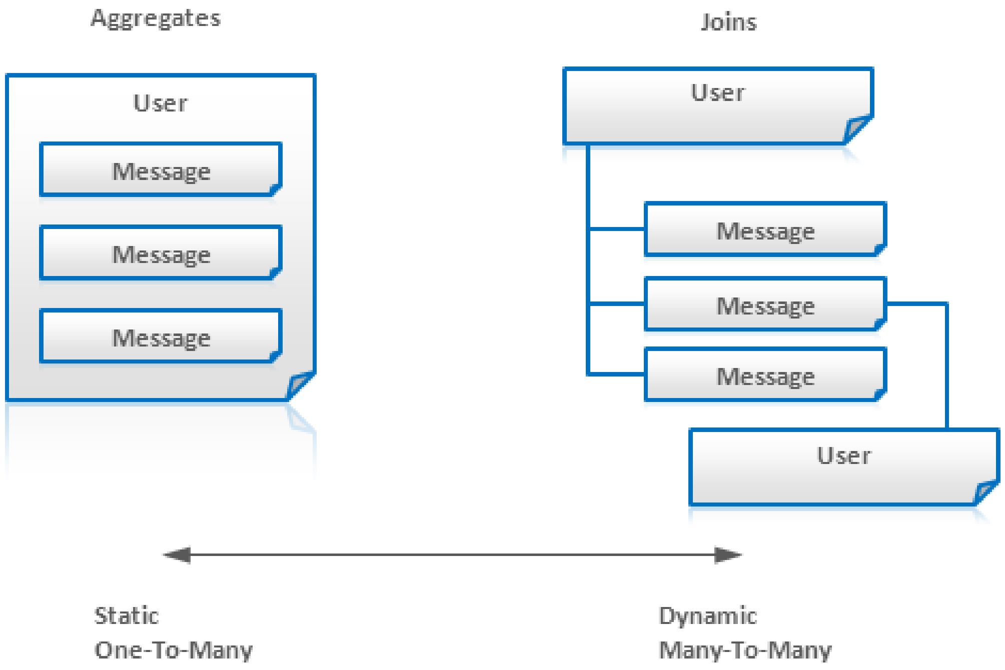
Applicability: Key-Value Stores, Document Databases, BigTable-style Databases

(3) Application Side Joins

Joins are rarely supported in NoSQL solutions. As a consequence of the "question-oriented" NoSQL nature, joins are often handled at design time as opposed to relational models where joins are handled at query execution time. Query time joins almost always mean a performance penalty, but in many cases one can avoid joins using Denormalization and Aggregates, i.e. embedding nested entities. Of course, in many cases joins are inevitable and should be handled by an application. The major use cases are:

- Many to many relationships are often modeled by links and require joins.
- Aggregates are often inapplicable when entity internals are the subject of frequent modifications. It is usually better to keep a record that something happened and join the records at query time as opposed to changing a value . For example, a messaging system can

be modeled as a User entity that contains nested Message entities. But if messages are often appended, it may be better to extract Messages as independent entities and join them to the User at query time:



(https://highlyscalable.files.wordpress.com/2012/03/aggregates-joins.png)

Applicability: Key-Value Stores, Document Databases, BigTable-style Databases, Graph Databases

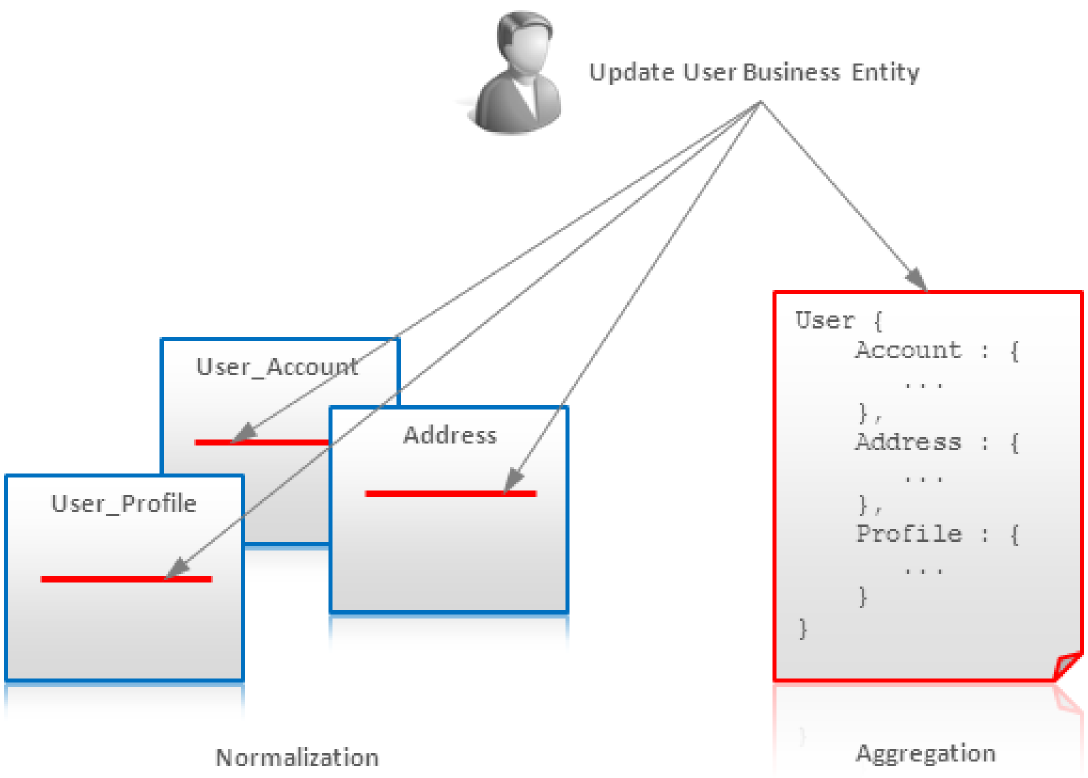
General Modeling Techniques

In this section we discuss general modeling techniques that applicable to a variety of NoSQL implementations.

(4) Atomic Aggregates

Many, although not all, NoSQL solutions have limited transaction support. In some cases one can achieve transactional behavior using distributed locks or application-managed MVCC (https://highlyscalable.wordpress.com/2012/01/07/mvcc-transactions-key-value/), but it is common to model data using an Aggregates technique to guarantee some of the ACID properties.

One of the reasons why powerful transactional machinery is an inevitable part of the relational databases is that normalized data typically require multi-place updates. On the other hand, Aggregates allow one to store a single business entity as one document, row or key-value pair and update it atomically:



(https://highlyscalable.files.wordpress.com/2012/02/atomic-aggregate1.png)

Atomic Aggregates

Of course, Atomic Aggregates as a data modeling technique is not a complete transactional solution, but if the store provides certain guaranties of atomicity, locks, or test-and-set instructions then Atomic Aggregates can be applicable.

Applicability: Key-Value Stores, Document Databases, BigTable-style Databases

(5) Enumerable Keys

Perhaps the greatest benefit of an unordered Key-Value data model is that entries can be partitioned across multiple servers by just hashing the key. Sorting makes things more complex, but sometimes an application is able to take some advantages of ordered keys even if storage doesn't offer such a feature. Let's consider the modeling of email messages as an example:

1. Some NoSQL stores provide atomic counters that allow one to generate sequential IDs. In this case one can store messages using *userID_messageID* as a composite key. If the latest message

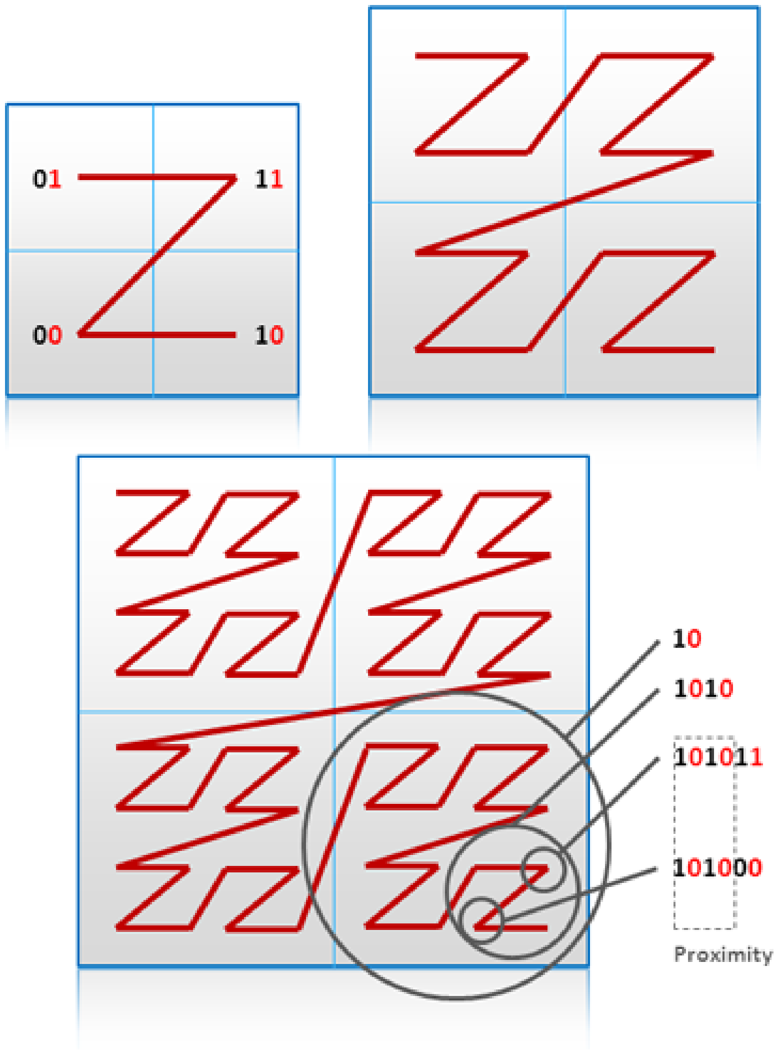
ID is known, it is possible to traverse previous messages. It is also possible to traverse preceding and succeeding messages for any given message ID.
2. Messages can be grouped into buckets, for example, daily buckets. This allows one to traverse a mail box backward or forward starting from any specified date or the current date.

Applicability: Key-Value Stores

(6) Dimensionality Reduction

Dimensionality Reduction is a technique that allows one to map multidimensional data to a Key-Value model or to other non-multidimensional models.

Traditional geographic information systems use some variation of a Quadtree or R-Tree for indexes. These structures need to be updated in-place and are expensive to manipulate when data volumes are large. An alternative approach is to traverse the 2D structure and flatten it into a plain list of entries. One well known example of this technique is a Geohash. A Geohash uses a Z-like scan to fill 2D space and each move is encoded as 0 or 1 depending on direction. Bits for longitude and latitude moves are interleaved as well as moves. The encoding process is illustrated in the figure below, where black and red bits stand for longitude and latitude, respectively:



(https://highlyscalable.files.wordpress.com/2012/02/geohash-traversal1.png)

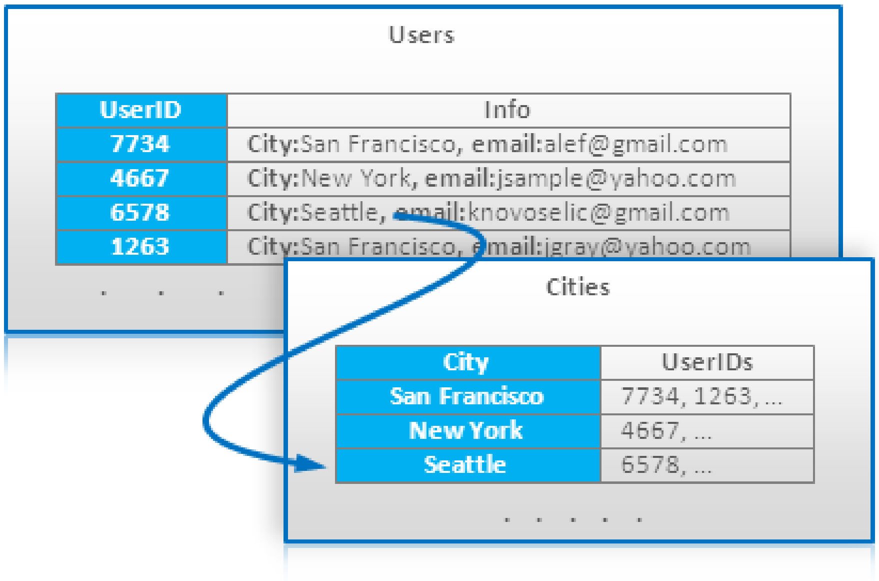
Geohash Index

An important feature of a Geohash is its ability to estimate distance between regions using bit-wise code proximity, as is shown in the figure. Geohash encoding allows one to store geographical information using plain data models, like sorted key values preserving spatial relationships. The Dimensionality Reduction technique for BigTable was described in [6.1]. More information about Geohashes and other related techniques can be found in [6.2] and [6.3].

Applicability: Key-Value Stores, Document Databases, BigTable-style Databases

(7) Index Table

Index Table is a very straightforward technique that allows one to take advantage of indexes in stores that do not support indexes internally. The most important class of such stores is the BigTable-style database. The idea is to create and maintain a special table with keys that follow the access pattern. For example, there is a master table that stores user accounts that can be accessed by user ID. A query that retrieves all users by a specified city can be supported by means of an additional table where city is a key:


Users
UserID
Info
7734
City:San Francisco, email:alef@gmail.com
4667
City:New York, email:jsample@yahoo.com
6578
City:Seattle, email:knovoselic@gmail.com
1263
City:San Francisco, email:jgray@yahoo.com
Cities
City
UserIDs
San Francisco
7734, 1263, ...
New York
4667, ...
Seattle
6578, ...

(https://highlyscalable.files.wordpress.com/2012/02/index-table.png)

Index Table Example

An Index table can be updated for each update of the master table or in batch mode. Either way, it results in an additional performance penalty and become a consistency issue.

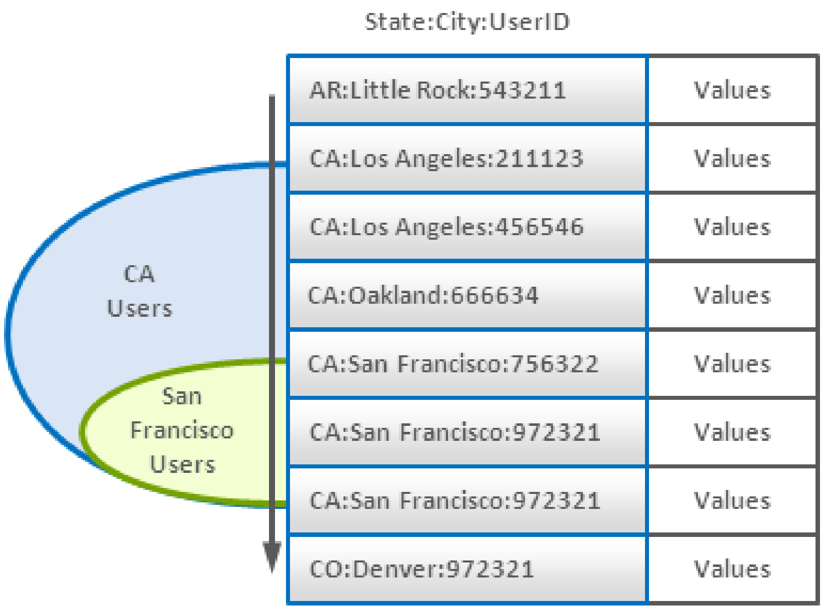
Index Table can be considered as an analog of materialized views in relational databases.

Applicability: BigTable-style Databases

(8) Composite Key Index

Composite key is a very generic technique, but it is extremely beneficial when a store with ordered keys is used. Composite keys in conjunction with secondary sorting allows one to build a kind of multidimensional index which is fundamentally similar to the previously described Dimensionality Reduction technique. For example, let's take a set of records where each record is a user statistic. If we are going to aggregate these statistics by a region the user came from, we can use keys in a format *(State:City:UserID)* that allow us to iterate over records for a particular state or city if that store supports the selection of key ranges by a partial key match (as BigTable-style systems do):

```
SELECT Values WHERE state="CA:*"
SELECT Values WHERE city="CA:San Francisco*"
```



(https://highlyscalable.files.wordpress.com/2012/03/composite-key-index.png)

Composite Key Index

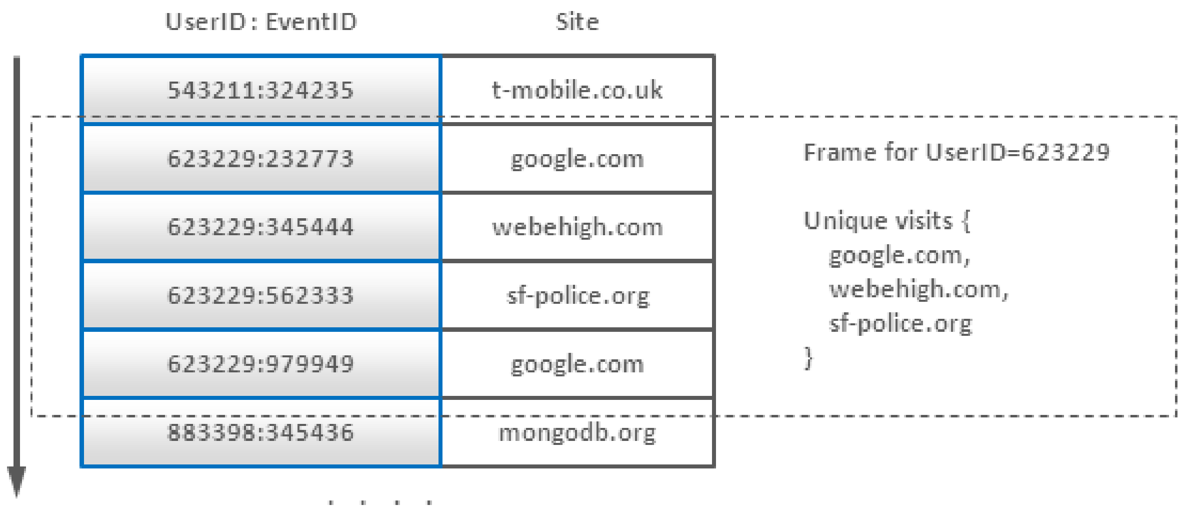
Applicability: BigTable-style Databases

(9) Aggregation with Composite Keys

Composite keys may be used not only for indexing, but for different types of grouping. Let's consider an example. There is a huge array of log records with information about internet users and their visits from different sites (*click stream*). The goal is to count the number of unique users for each site. This is similar to the following SQL query:

```
SELECT count(distinct(user_id)) FROM clicks GROUP BY site
```

We can model this situation using composite keys with a UserID prefix:



(https://highlyscalable.files.wordpress.com/2012/02/composite-key-collating1.png)

Counting Unique Users using Composite Keys

The idea is to keep all records for one user collocated, so it is possible to fetch such a frame into memory (one user can not produce too many events) and to eliminate site duplicates using hash table or whatever. An alternative technique is to have one entry for one user and append sites to this entry as events arrive. Nevertheless, entry modification is generally less efficient than entry insertion in the majority of implementations.

Applicability: Ordered Key-Value Stores, BigTable-style Databases

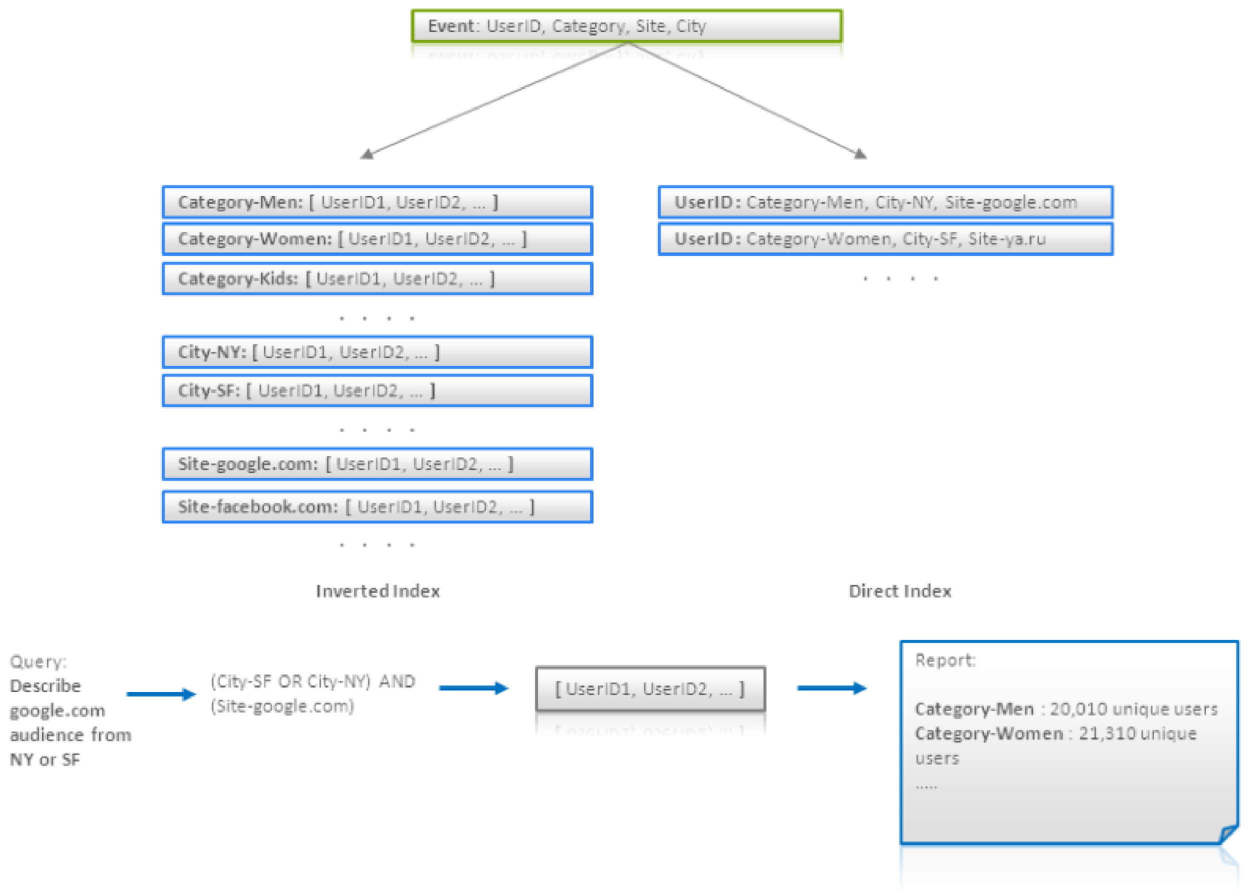
(10) Inverted Search – Direct Aggregation

This technique is more a data processing pattern, rather than data modeling. Nevertheless, data models are also impacted by usage of this pattern. The main idea of this technique is to use an index to find data that meets a criteria, but aggregate data using original representation or full scans. Let's consider an example. There are a number of log records with information about internet users and their visits from different sites (*click stream*). Let assume that each record contains user ID, categories this user belongs to (Men, Women, Bloggers, etc), city this user came from, and visited site. The goal is to describe the audience that meet some criteria (site, city, etc) in terms of unique users for each category that occurs in this audience (i.e. in the set of users that meet the criteria).

It is quite clear that a search of users that meet the criteria can be efficiently done using inverted indexes like *{Category -> [user IDs]}* or *{Site -> [user IDs]}*. Using such indexes, one can intersect or unify corresponding user IDs (this can be done very efficiently if user IDs are stored as sorted lists or bit sets) and obtain an audience. But describing an audience which is similar to an aggregation query like

```
SELECT count(distinct(user_id)) ... GROUP BY category
```

cannot be handled efficiently using an inverted index if the number of categories is big. To cope with this, one can build a direct index of the form *{UserID -> [Categories]}* and iterate over it in order to build a final report. This schema is depicted below:


Event: UserID, Category, Site, City
Category-Men: [UserID1, UserID2, ...]
Category-Women: [UserID1, UserID2, ...]
Category-Kids: [UserID1, UserID2, ...]
City-NY: [UserID1, UserID2, ...]
City-SF: [UserID1, UserID2, ...]
Site-google.com: [UserID1, UserID2, ...]
Site-facebook.com: [UserID1, UserID2, ...]
UserID: Category-Men, City-NY, Site-google.com
UserID: Category-Women, City-SF, Site-ya.ru
Inverted Index
Direct Index
Query: Describe google.com audience from NY or SF
(City-SF OR City-NY) AND (Site-google.com)
[UserID1, UserID2, ...]
Report:
Category-Men : 20,010 unique users
Category-Women : 21,310 unique users
.....

(https://highlyscalable.files.wordpress.com/2012/02/invert-direct1.png)

Counting Unique Users using Inverse and Direct Indexes

And as a final note, we should take into account that random retrieval of records for each user ID in the audience can be inefficient. One can grapple with this problem by leveraging batch query processing. This means that some number of user sets can be precomputed (for different criteria) and then all reports for this batch of audiences can be computed in one full scan of direct or inverse index.

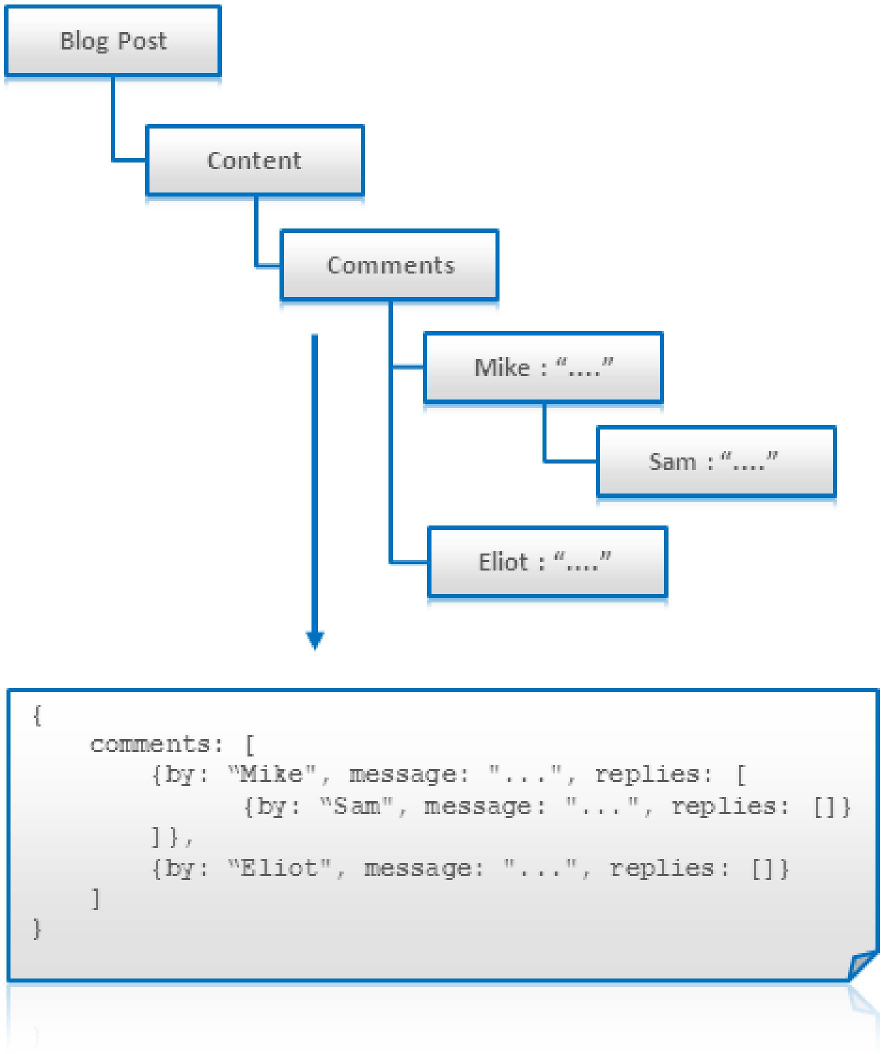
Applicability: Key-Value Stores, BigTable-style Databases, Document Databases

Hierarchy Modeling Techniques

(11) Tree Aggregation

Trees or even arbitrary graphs (with the aid of denormalization) can be modeled as a single record or document.

- This techniques is efficient when the tree is accessed at once (for example, an entire tree of blog comments is fetched to show a page with a post).
- Search and arbitrary access to the entries may be problematic.
- Updates are inefficient in most NoSQL implementations (as compared to independent nodes).



```
{
    comments: [
        {by: "Mike", message: "...", replies: [
               {by: "Sam", message: "...", replies: []}
        ]},
        {by: "Eliot", message: "...", replies: []}
    ]
}
```

(https://highlyscalable.files.wordpress.com/2012/02/tree-aggregation.png)

Tree Aggregation

Applicability: Key-Value Stores, Document Databases

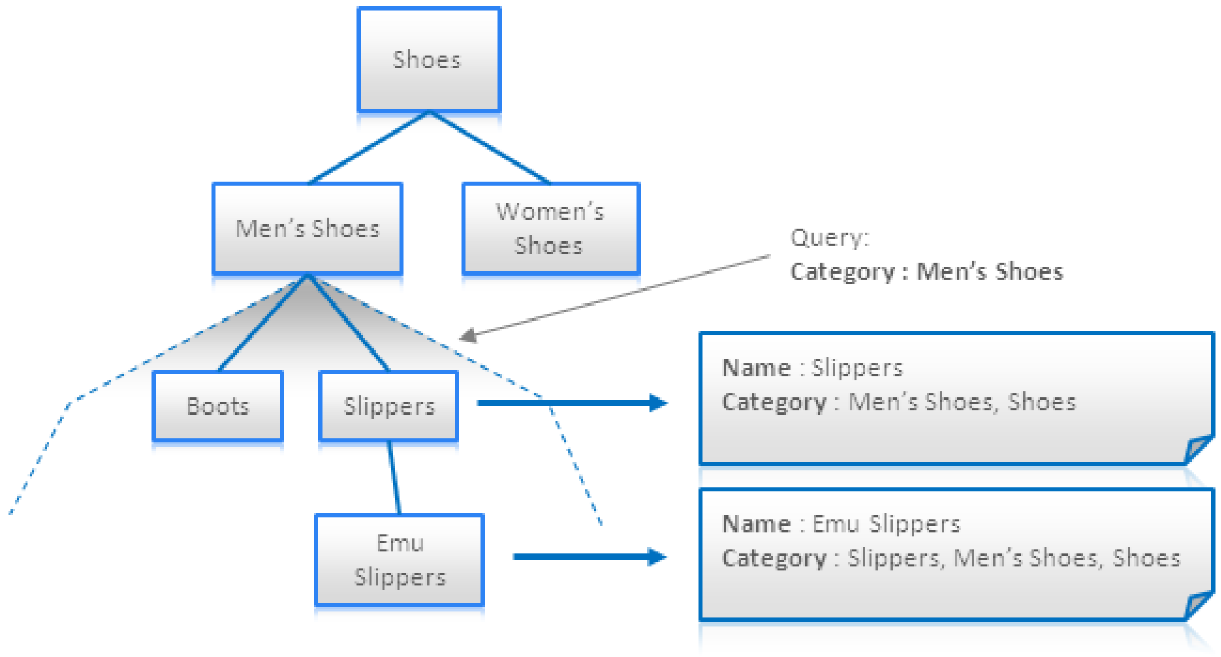
(12) Adjacency Lists

Adjacency Lists are a straightforward way of graph modeling – each node is modeled as an independent record that contains arrays of direct ancestors or descendants. It allows one to search for nodes by identifiers of their parents or children and, of course, to traverse a graph by doing one hop per query. This approach is usually inefficient for getting an entire subtree for a given node, for deep or wide traversals.

Applicability: Key-Value Stores, Document Databases

(13) Materialized Paths

Materialized Paths is a technique that helps to avoid recursive traversals of tree-like structures. This technique can be considered as a kind of denormalization. The idea is to attribute each node by identifiers of all its parents or children, so that it is possible to determine all descendants or predecessors of the node without traversal:

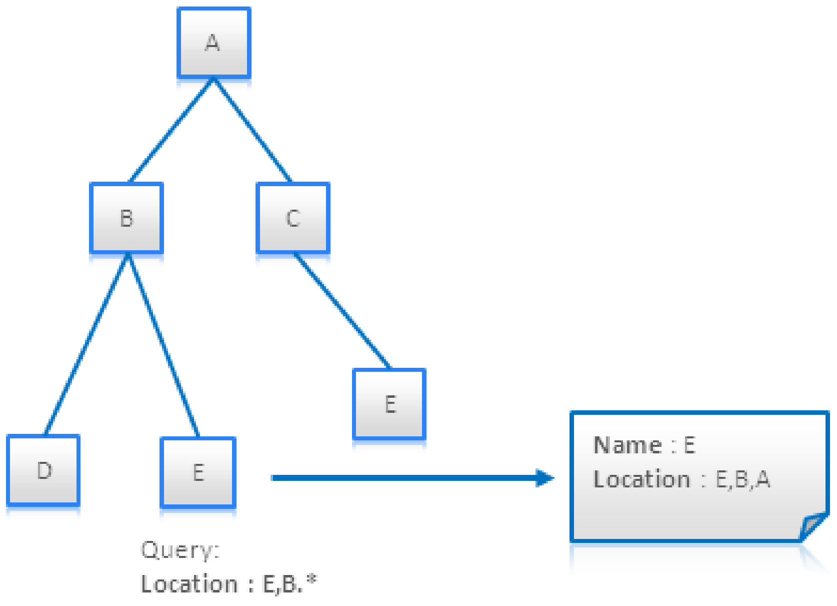


(https://highlyscalable.files.wordpress.com/2012/02/materialized-paths2.png)

Materialized Paths for eShop Category Hierarchy

This technique is especially helpful for Full Text Search Engines because it allows one to convert hierarchical structures into flat documents. One can see in the figure above that all products or subcategories within the *Men's Shoes* category can be retrieved using a short query which is simply a category name.

Materialized Paths can be stored as a set of IDs or as a single string of concatenated IDs. The latter option allows one to search for nodes that meet a certain partial path criteria using regular expressions. This option is illustrated in the figure below (path includes node itself):



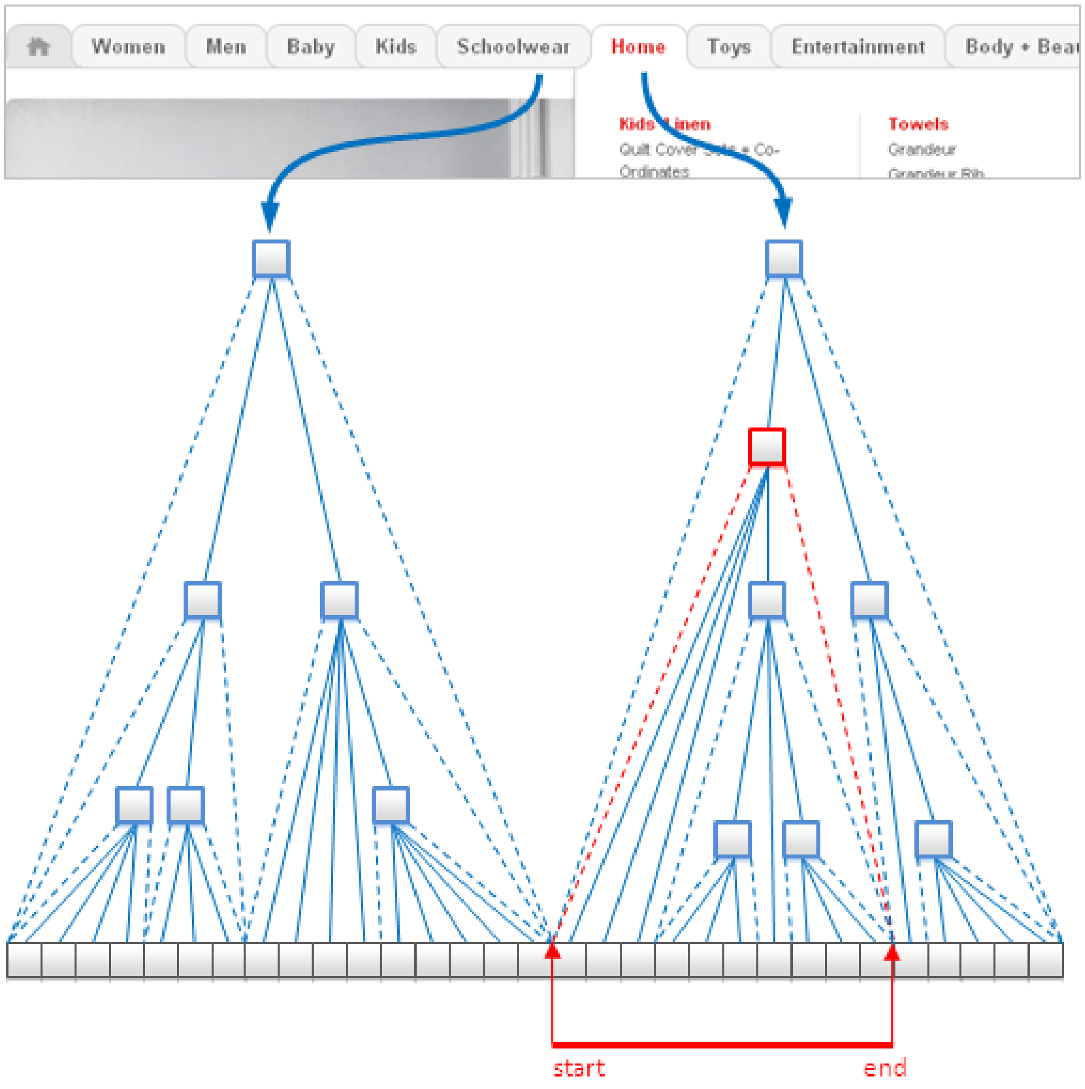
(https://highlyscalable.files.wordpress.com/2012/02/materialized-paths-2.png)

Query Materialized Paths using RegExp

Applicability: Key-Value Stores, Document Databases, Search Engines

(14) Nested Sets

Nested sets (http://en.wikipedia.org/wiki/Nested_set_model) is a standard technique for modeling tree-like structures. It is widely used in relational databases, but it is perfectly applicable to Key-Value Stores and Document Databases. The idea is to store the leafs of the tree in an array and to map each non-leaf node to a range of leafs using start and end indexes, as is shown in the figure below:



[(https://highlyscalable.files.wordpress.com/2012/02/nested-sets.png)](https://highlyscalable.files.wordpress.com/2012/02/nested-sets.png)

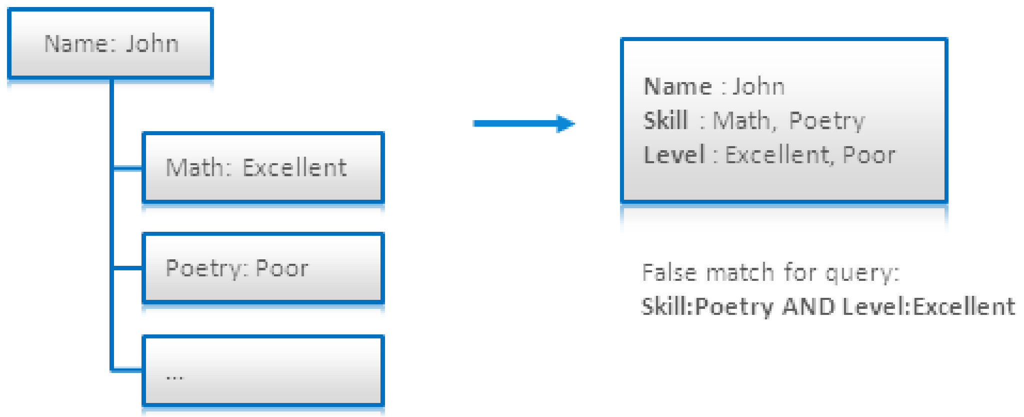
Modeling of eCommerce Catalog using Nested Sets

This structure is pretty efficient for immutable data because it has a small memory footprint and allows one to fetch all leafs for a given node without traversals. Nevertheless, inserts and updates are quite costly because the addition of one leaf causes an extensive update of indexes.

Applicability: Key-Value Stores, Document Databases

(15) Nested Documents Flattening: Numbered Field Names

Search Engines typically work with flat documents, i.e. each document is a flat list of fields and values. The goal of data modeling is to map business entities to plain documents and this can be challenging if the entities have a complex internal structure. One typical challenge mapping documents with a hierarchical structure, i.e. documents with nested documents inside. Let's consider the following example:

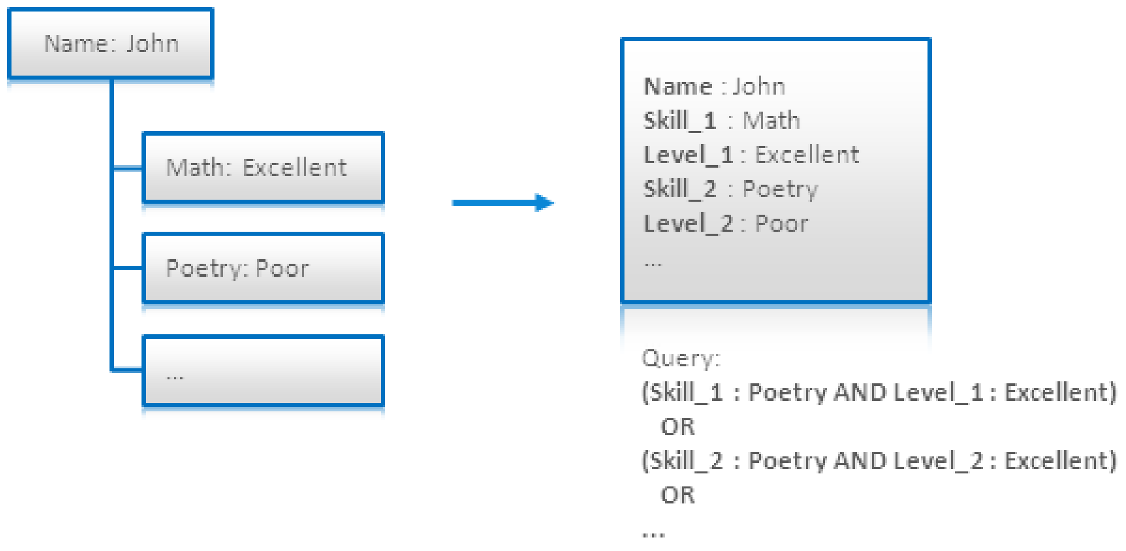


(https://highlyscalable.files.wordpress.com/2012/02/nested-documents-1.png)

Nested Documents Problem

Each business entity is some kind of resume. It contains a person's name and a list of his or her skills with a skill level. An obvious way to model such an entity is to create a plain document with *Skill* and *Level* fields. This model allows one to search for a person by skill or by level, but queries that combine both fields are liable to result in false matches, as depicted in the figure above.

One way to overcome this issue was suggested in [4.6]. The main idea of this technique is to index each skill and corresponding level as a dedicated pair of fields *Skill_i* and *Level_i,* and to search for all these pairs simultaneously (where the number of OR-ed terms in a query is as high as the maximum number of skills for one person):



(https://highlyscalable.files.wordpress.com/2012/02/nested-documents-3.png)

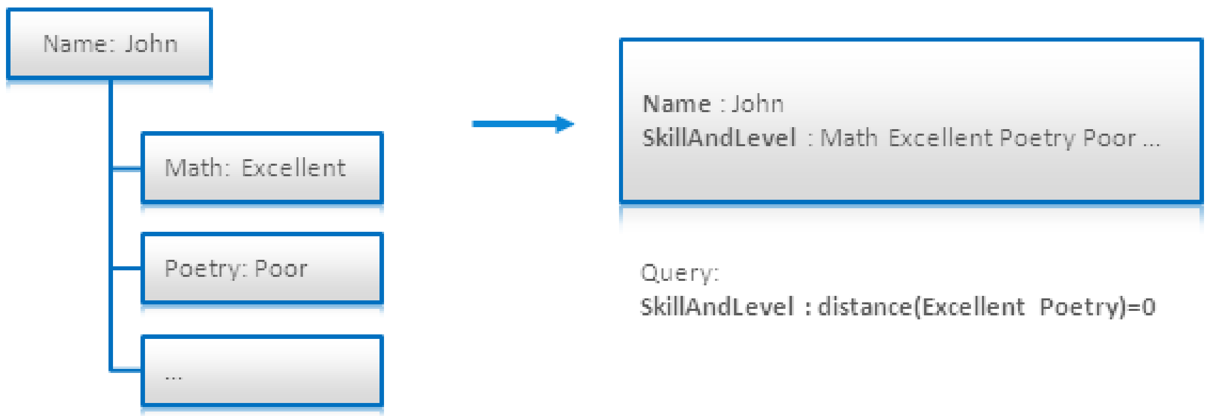
Nested Document Modeling using Numbered Field Names

This approach is not really scalable because query complexity grows rapidly as a function of the number of nested structures.

Applicability: Search Engines

(16) Nested Documents Flattening: Proximity Queries

The problem with nested documents can be solved using another technique that were also described in [4.6]. The idea is to use proximity queries that limit the acceptable distance between words in the document. In the figure below, all skills and levels are indexed in one field, namely, SkillAndLevel, and the query indicates that the words "Excellent" and "Poetry" should follow one another:


Name: John
Math: Excellent
Poetry: Poor
...
Name : John
SkillAndLevel : Math Excellent Poetry Poor ...
Query:
SkillAndLevel : distance(Excellent Poetry)=0

(https://highlyscalable.files.wordpress.com/2012/02/nested-documents-2.png)

Nested Document Modeling using Proximity Queries

[4.3] describes a success story for this technique used on top of Solr.

Applicability: Search Engines

(17) Batch Graph Processing

Graph databases like neo4j are exceptionally good for exploring the neighborhood of a given node or exploring relationships between two or a few nodes. Nevertheless, global processing of large graphs is not very efficient because general purpose graph databases do not scale well. Distributed graph processing can be done using MapReduce and the Message Passing pattern that was described, for example, in one of my previous articles (https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/). This approach makes Key-Value stores, Document databases, and BigTable-style databases suitable for processing large graphs.

Applicability: Key-Value Stores, Document Databases, BigTable-style Databases

References

Finally, I provide a list of useful links related to NoSQL data modeling:

1. Key-Value Stores:
    1. http://www.devshed.com/c/a/MySQL/Database-Design-Using-KeyValue-Tables/ (http://www.devshed.com/c/a/MySQL/Database-Design-Using-KeyValue-Tables/)
    2. http://antirez.com/post/Sorting-in-key-value-data-model.htm (http://antirez.com/post/Sorting-in-key-value-data-model.html)l
    3. http://stackoverflow.com/questions/3554169/difference-between-document-based-and-key-value-based-databases (http://stackoverflow.com/questions/3554169/difference-between-document-based-and-key-value-based-databases)
    4. http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html (http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html)
2. BigTable-style Databases:
    1. http://www.slideshare.net/ebenhewitt/cassandra-datamodel-4985524 (http://www.slideshare.net/ebenhewitt/cassandra-datamodel-4985524)
    2. http://www.slideshare.net/mattdennis/cassandra-data-modeling (http://www.slideshare.net/mattdennis/cassandra-data-modeling)
    3. http://nosql.mypopescu.com/post/17419074362/cassandra-data-modeling-examples-with-matthew-f-dennis (http://nosql.mypopescu.com/post/17419074362/cassandra-data-modeling-examples-with-matthew-f-dennis)
    4. http://s-expressions.com/2009/03/08/hbase-on-designing-schemas-for-column-oriented-data-stores/ (http://s-expressions.com/2009/03/08/hbase-on-designing-schemas-for-column-oriented-data-stores/)
    5. http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable (http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable)
3. Document Databases:
    1. http://www.slideshare.net/mongodb/mongodb-schema-design-richard-kreuters-mongo-berlin-preso (http://www.slideshare.net/mongodb/mongodb-schema-design-richard-kreuters-mongo-berlin-preso)
    2. http://www.michaelhamrah.com/blog/2011/08/data-modeling-at-scale-mongodb-mongoid-callbacks-and-denormalizing-data-for-efficiency/ (http://www.michaelhamrah.com/blog/2011/08/data-modeling-at-scale-mongodb-mongoid-callbacks-and-denormalizing-data-for-efficiency/)
    3. http://seancribbs.com/tech/2009/09/28/modeling-a-tree-in-a-document-database/ (http://seancribbs.com/tech/2009/09/28/modeling-a-tree-in-a-document-database/)
    4. http://www.mongodb.org/display/DOCS/Schema+Design (http://www.mongodb.org/display/DOCS/Schema+Design)
    5. http://www.mongodb.org/display/DOCS/Trees+in+MongoDB (http://www.mongodb.org/display/DOCS/Trees+in+MongoDB)
    6. http://blog.fiesta.cc/post/11319522700/walkthrough-mongodb-data-modeling

(http://blog.fiesta.cc/post/11319522700/walkthrough-mongodb-data-modeling)

4. Full Text Search Engines:
    1. http://www.searchworkings.org/blog/-/blogs/query-time-joining-in-lucene (http://www.searchworkings.org/blog/-/blogs/query-time-joining-in-lucene)
    2. http://www.lucidimagination.com/devzone/technical-articles/solr-and-rdbms-basics-designing-your-application-best-both (http://www.lucidimagination.com/devzone/technical-articles/solr-and-rdbms-basics-designing-your-application-best-both)
    3. http://blog.griddynamics.com/2011/07/solr-experience-search-parent-child.html (http://blog.griddynamics.com/2011/07/solr-experience-search-parent-child.html)
    4. http://www.lucidimagination.com/blog/2009/07/18/the-spanquery/ (http://www.lucidimagination.com/blog/2009/07/18/the-spanquery/)
    5. http://blog.mgm-tp.com/2011/03/non-standard-ways-of-using-lucene/ (http://blog.mgm-tp.com/2011/03/non-standard-ways-of-using-lucene/)
    6. http://www.slideshare.net/MarkHarwood/proposal-for-nested-document-support-in-lucene (http://www.slideshare.net/MarkHarwood/proposal-for-nested-document-support-in-lucene)
    7. http://mysolr.com/tips/denormalized-data-structure/ (http://mysolr.com/tips/denormalized-data-structure/)
    8. http://sujitpal.blogspot.com/2010/10/denormalizing-maps-with-lucene-payloads.html (http://sujitpal.blogspot.com/2010/10/denormalizing-maps-with-lucene-payloads.html)
    9. http://java.dzone.com/articles/hibernate-search-mapping-entit (http://java.dzone.com/articles/hibernate-search-mapping-entit)
5. Graph Databases:
    1. http://docs.neo4j.org/chunked/stable/tutorial-comparing-models.html (http://docs.neo4j.org/chunked/stable/tutorial-comparing-models.html)
    2. http://blog.neo4j.org/2010/03/modeling-categories-in-graph-database.html (http://blog.neo4j.org/2010/03/modeling-categories-in-graph-database.html)
    3. http://skillsmatter.com/podcast/nosql/graph-modelling (http://skillsmatter.com/podcast/nosql/graph-modelling)
    4. http://www.umiacs.umd.edu/~jimmylin/publications/Lin_Schatz_MLG2010.pdf (http://www.umiacs.umd.edu/~jimmylin/publications/Lin_Schatz_MLG2010.pdf)
6. Demensionality Reduction:
    1. http://www.slideshare.net/mmalone/scaling-gis-data-in-nonrelational-data-stores (http://www.slideshare.net/mmalone/scaling-gis-data-in-nonrelational-data-stores)
    2. http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves (http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves)
    3. http://www.trisis.co.uk/blog/?p=1287 (http://www.trisis.co.uk/blog/?p=1287)

- Date: on March 1, 2012October 16, 2012

- Category: Big Data, Fundamentals

- Tag: big table, data modeling, document, geohash, graph, index, key value, lucene, nosql

- Comments: 78 Comments

78 Comments

LEAVE A COMMENT

1.
   Dominique Plante says:
   MARCH 1, 2012 AT 9:15 PM
   Thanks for the great article! The diagram of the NoSQL data models is hilarious.

   I think I found a small typo – "Denormalization can be defined as coping of the same data.." coping should be copying

   REPLY
   1.
      Ilya Katsov says:
      MARCH 2, 2012 AT 6:10 AM
      Thanks for pointing this. Fixed.

      REPLY
      1.
         Sam Wesley says:
         MARCH 27, 2012 AT 3:22 AM
         Nice article, but it could really use a proof read to correct the grammar. There are many more problems than that one typo (lack of articles – a, the, etc…. – for one).

REPLY

1. **Ilya Katsov says:**
   MARCH 27, 2012 AT 7:59 AM
   I'm doing my best I'm not a native English speaker though

2. **TkTech says:**
   MARCH 2, 2012 AT 2:04 AM
   Excellent article and due diligence providing references. Long, but a worthwhile read. Wish articles like this made it to the frontpage of reddit more often.

   REPLY

3. **Voice in the wind says:**
   MARCH 2, 2012 AT 3:51 AM
   A very interesting post and nice graphics. However in the interest of clarity there are a few things I disagree with and as someone who has designed databases for more than 20 years in the interest of the new developers who may read this I feel some history should be injected into these conversations so the same mistakes are not repeated.

   When relational databases began there was a thing called ISAM which was the system dejour for large databases at the time – the interesting part about ISAM in reference to NoSql was that the application code navigated indexes to find the data, and the indexes were designed into the application. Changing these became a nightmare as the tight coupling of data and access methods was very difficult to change, often it was easier to chuck it and start again. Relational databases provided the freedom to model the data, and then as the system evolved and you needed to provide different queries for reporting and so on, you could tune your queries by adding indexes, add columns here and there and modify the schema.

   There was much argument at the time from the ISAM guys saying this couldn't possibly work, how can you optimise your access paths without knowing before hand, also query optimisation will be to slow and so on. It's funny to be seeing the same old things going back the other way.

   I'll pick one of your points above among the many – 'Relational modeling is typically driven by structure of available data', this is not true. Relational databases are designed with the view of what data is needed to be stored for the application. You could be referring to queries – in which case this would be true, and query and reporting, by it's nature, can only give you the data you have. There is an incredible body of work on how to design databases to suit applications, not the other way round.

   High transaction rates are not a problem with RDBMSes, likewise availability, but here's the rub the present incumbents are very expensive Oracle et al, they are also complex, the black arts of being a DBA are legendary. So to create a high availability platform using a commercial RDBMS is expensive, because of licensing and specialized skills. So what's happening imho is that we're going back to the bad old days – not because it's better, but because it's cheaper, and NoSql is free, and in uncomplicated data models – like shopping sites – then it probably will do. *However* know what your trading off, because as your site gets larger, and more complicated access paths are required as the CEO wants sales reports based every two weeks but your data structure stores them monthly, then it will hit the wall.

So like everything NoSql is a trade off and in this case cost versus coding hours amongst other things, if your coding time is cheap then by all means reproduce all the things an RDBMS does, however know that you'll be delaying the inevitable, something will come along that your NoSql model doesn't cover and the pain of changing the model isn't as easy as typing 'Create Index…'

A very nice exposition of NoSql though, and it does have it's place, much the same way as Microsoft access does

REPLY

1. **Scott Lynch says:**
   MARCH 2, 2012 AT 8:37 PM
   Not to mention the current move towards NoSQL implies that any and all access to the data is through application code only. If somebody needs ad-hoc queries, you can't write a couple lines of SQL to get the answer. There are never any bugs in the code are there? What could possibly go wrong?

   REPLY

   1. **Ilya Katsov says:**
      MARCH 3, 2012 AT 5:55 AM
      Scott,
      I would like to mention that many NoSQL systems provide data introspection tools, so it is often possible to do ad-hoc queries. Of course, many applications use their own binary data format, but in this case custom introspection tools are often developed.

      REPLY

   2. **Voice in the wind. says:**
      MARCH 3, 2012 AT 12:52 PM
      Indeed. Recovery and consistency are other issues as you allude to. I am disappointed a bit, I thought there'd be some refutation, maybe I was missing something – but obviously it's more 'we're building something new man – you just don't get it'. As long as this crowd doesn't start writing software for my bank/car etc.

      REPLY

      1. **Ilya Katsov says:**
         MARCH 4, 2012 AT 5:45 AM

This article is about NoSQL data modeling, not more, not less. It doesn't criticize RDBMSes or claim that NoSQL is superior in any sense.

2.

Voice in the wind says:
MARCH 5, 2012 AT 2:58 AM
Well there are a number of statements that do criticize RDBMses

NoSQL data modeling often starts from the application-specific queries as opposed to relational modeling:

Relational modeling is typically driven by structure of available data, the main design theme is "What answers do I have?"

Relational databases are not very convenient for hierarchical or graph-like data modeling and processing

and I'm afraid all these statements are false. The implication being that RDBMses are deficient and NoSql is superior: again an incorrect inference, and that was my statement that NoSql is a back to the past technology and I gave the reasons why. As I've said NoSql has it's place but look at it with open eyes. If you said that NoSql can be used for performance and cost reasons in particular cases then I would be less inclined to argue.

REPLY

1.

Ilya Katsov says:
MARCH 5, 2012 AT 7:26 AM
Actually, I can not understand why do you consider the first two statements as a criticism of RDBMSes. Both approaches (I mean answer- or question-driven) have their own pros/cons and "typically"/"often" doesn't mean "always" – relational modeling, of course, allows query-driven schemes and denormalization if necessary, but I can not admit that these techniques are the first-class citizens.

I admit that the third statement may be controversial, but I don't think that judgement like "are not very convenient" can be true or false. There is a group of people, including me, who think that graph databases are more convenient than RDBMSes in certain cases, but of course there are different opinions.

I completely agree with you that performance, scalability and cost reasons are the main drivers of NoSQL. Obviously, complex data modeling is not an end in itself

REPLY

3.

Vincent Lowe says:
JANUARY 16, 2014 AT 1:19 AM
…Ummmm, yeah, but some of the arguments you present here sound like the cries of the ISAM guys in the face of advancing RDBMS technology.

The real equation is that storage of data in a structured store represents a significant investment in software, hardware, and particularly in human capital. Then maintenance and updates to this store require significant additional investment of human capital.

It's justified and it really works well, but things have changed. Here's how:

Cost of storage hardware has decreased by 1400x in past years.
Cost of network transport has decreased by around 400x in the same time period.
Data read times have improved only 12x in that same time.

So to get at data in the scale we now see becoming common, the data must be stored in a fashion that can be distributed over a large compute range.

In light of that, key-value stores now find themselves being the Belle of the Ball again.

Cost of insertion and maintenance for data in a Hadoop cluster is lower than RDBMS by a couple of orders of magnitude. It's not completely clear yet, but cost of design and maintenance will probably be a full order of magnitude improvement.

Ignore this paradigm evolution at your own peril.

REPLY

1.

cgkapp says:
JUNE 23, 2014 AT 1:42 PM
sure – things like MapReduce help get over some of the rigidity of ISAM . But the code is still coupled to the data store and that has never proven to be a great thing for transactional applications. But many big-data apps aren't transactional are they? So just like always – pick the best tool for the requirement – we just have another suite of tools now.

REPLY

4.

h.ash says:
MARCH 2, 2012 AT 8:22 AM
impressive and nice article. this gives detailed history of nosql systems and ways they could be used efficiently.

REPLY

1.

Ilya Katsov says:
MARCH 2, 2012 AT 8:29 AM
I would like to notice that this "history" has nothing to do with the real timeline of NoSQL developments. This is just an imaginary concept that helps to explain relationships.

REPLY

5.

Chris Comella (@chrisco) says:

MARCH 2, 2012 AT 8:27 AM
Very interesting, although as a beginner it's going to take time and practice for me to learn well. Question: I wonder if you have any suggestion(s) for what might be the best data structure for a database of 10,000 products sold on Amazon? I have a freelancer collecting the data and I need a data structure in which to put it. I think it will come in an XML file. I will collect the data over time, so the Big Table, Materialized Paths, and Nested Sets caught my eye. Once I have to data, I want will have someone help me gather statistics and metrics, analyze, looking for trends, correlations, etc. Some will be basic query and some may be machine learning. Thanks for any tips

REPLY

1. **Ilya Katsov says:**
   MARCH 3, 2012 AT 6:28 AM
   Chris,
   I worked with eCommerce systems that used Nested Sets or Materialized Paths, but these structures were chosen to meet high performance requirements, say, thousands requests/sec for a set of tens of thousands of products. But high performance doesn't come for free – these structures are relatively difficult to implement and update. Of course I don't know your functional and performance requirements, but from what you said (small data size, statistical processing) I can advice you to consider something straightforward – per-product documents in MongoDB, search engine like Solr (can be helpful to deal with free text data like product description), or standard relational DB. Complicated modeling should be avoided unless it is unavoidable because of performance requirements or whatever.

   REPLY

6. **Mikhail Khludnev says:**
   MARCH 2, 2012 AT 9:20 AM
   Some recent stuff in inverted index camp

   http://blog.mikemccandless.com/2012/01/searching-relational-content-with.html
   https://issues.apache.org/jira/browse/SOLR-3076

   It's worth to mention it too

   REPLY

7. **Martin Horvath says:**
   MARCH 2, 2012 AT 5:18 PM
   An excellent article which motivated me to re-think about my modeling strategies again. Thanks for your work! Very much appreciated!

   REPLY

8. **CraigL says:**

MARCH 3, 2012 AT 4:56 AM
This is the most lucid and well organized presentation of NoSQL techniques that I've seen. Thank you!

REPLY

9.

Scooletz (@Scooletz) says:
MARCH 3, 2012 AT 9:47 PM
Thanks for the great article as well as references, which allows readers to dig dipper. I'd like to add, that paradigms like Nested Sets, Materialized Paths can be also used with SQL-based solutions, especially, when sql dbs are connected with some cache and intelligent expiration (like one in NHibernate).

REPLY

10.

Hermann Schmidt (@alpengeist_de) says:
MARCH 5, 2012 AT 10:53 AM
Thanks a lot! You've saved me days of research. Almost no articles about NoSQL address data modelling. Most stir the scaling and CAP soup. Understanding the nature of the data to evaluate the utility of a NoSQL DB is much more important for ordinary (non-Google-sized) businesses than the simple realization "I have lots of data".

REPLY

11.

mhamrah (@mhamrah) says:
MARCH 5, 2012 AT 7:57 PM
Great article- thanks for citing my post!

REPLY

12.

robert jones says:
MARCH 7, 2012 AT 4:01 PM
Relational modeling is typically driven by structure of available data, the main design theme is "What answers do I have?"

Wrong. I have done rdb modeling for 20 years and we don't do that. Statements like this undermine the integrity of your otherwise useful article.

REPLY

1.

Ilya Katsov says:
MARCH 8, 2012 AT 5:53 AM
Robert,
Can I ask you what exactly is wrong?

REPLY

1.

cgkapp says:
JUNE 23, 2014 AT 1:38 PM
What exactly is wrong is that good relational data modeling is driven by "what questions do I have", not by "what answers do I have". So you wrote it exactly backwards from the reality of relational data modeling in a commercial environment.

REPLY

13.
jmdev says:
MARCH 9, 2012 AT 3:24 PM
Great overview of NoSQL modeling! I wish more articles were researched as much as the ones on your blog. Quick question, what tool did you use to draw the first NoSQL evolution diagram? It is incredibly crisp and really adds a level of polish to your wealth of material. Thanks again!

REPLY

1.
Ilya Katsov says:
MARCH 11, 2012 AT 6:24 AM
what tool did you use to draw the first NoSQL evolution diagram – CorelDraw

REPLY

14.
andee_marks says:
MARCH 15, 2012 AT 2:00 AM
Reblogged this on Blah Blah Blog and commented:
Really nice overview of data modelling techniques for NoSQL databases.

REPLY

15.
Chandermani says:
MARCH 25, 2012 AT 1:53 PM
Excellent overview about data modeling. I was design for a nosql document database and this post helped me a lot. I have created post with has some good resources around document database modelling available here http://chandermani.blogspot.in/2012/03/nosql-data-modelling.html

REPLY

16.
cresports says:
APRIL 9, 2012 AT 7:58 AM
Reblogged this on wenfengsun.

REPLY

17.
Ki Chul Park says:
APRIL 16, 2012 AT 4:13 PM

Hi, I’m voluntarily contributing some translation works to a community. (http://engfordev.com)
If you don’t mind, I’d like to translate your article into Korean and publish it onto my community website.
And I promise you that agree your all rights and never use for profit.
Would you give me a permission to do so, please?
Thanks for your time and concern. I’m looking forward to hearing from you.

REPLY

1.

Ilya Katsov says:
APRIL 16, 2012 AT 7:03 PM
Sure, please go ahead. I really appreciate this.

REPLY

1.

Ki Chul Park says:
APRIL 17, 2012 AT 1:03 AM
Thanks a lot. =)

REPLY

18.

Praveen Kumar Jayaram says:
APRIL 20, 2012 AT 1:32 PM
Great article! Thanks.

REPLY

19.

Jamison White says:
APRIL 25, 2012 AT 3:52 PM
Reblogged this on Jamison White's Blog and commented:
Great, long discussion on NoSQL data modeling.

REPLY

20.

Artur Ejsmont (@artur_ejsmont) says:
MAY 6, 2012 AT 3:26 AM
Very good post, thanks a lot.

REPLY

21.

Gaurav says:
JUNE 27, 2012 AT 7:24 AM
Very nice info

REPLY

22.

egs says:
JULY 13, 2012 AT 3:49 PM
Sadly, this article misses one of the biggest new developments in the NoSQL space, namely, HyperDex. HyperDex is a consistent and fault-tolerant data store, with support for efficient retrieval by secondary attributes.

Existing NoSQL systems are little more than bucketed hashtables, and provide terrible consistency properties (i.e. eventual consistency) as a result. HyperDex provides linearizability. It does so while supporting retrieval by secondary attributes. And it does so with very high performance.

I liked the first few bullet points, but the article went south around the middle, when it started assuming that the only way to support retrieval by secondary attributes is by building indices. This is a very RDBMS-centric view, and it poses problems for consistency. HyperDex's internal data organization (called hyperspace hashing) enables it to sidestep these problems.

Overall, the author needs to do some further reading on recent developments in the NoSQL field.

REPLY

23.

Nantacoben Kim says:
JULY 14, 2012 AT 6:15 PM
This really great I added link to this blog at nantacoben.tumblr.com.

Also with all these controversies of RDB vs NoSQL, and How vs What,

By saying "What answers do I have?" I think what Ilya meant is by nature SQL is a declarative language. SQL tends specify "What should be accomplished" without worrying about what technologies are being used while getting the query results, and RDBMS uses SQL to manipulate and define data.

On the other hand NoSQL tends to focus on "how" to process data based on data access patterns.
NoSQL could also be declarative language, for example adding hive on top of hbase SQL could be used to query NoSQL system.

RDB could also be procedural by using PL/SQL

So in my point-of-view the boundaries between RDB and NoSQL being what and how is getting less definite over time.

Also (Voice in the wind) mentioned we're going back to the bad old days, but nothing is going back. Whether NoSQL uses denormalization or not it is a perfectly normal chronological evolution of DB systems. In fact 5th RDB normalization is all about denormalization. NoSQL came into recent trend in DB systems because many DB engineers felt performance and

consistency issue with RDB in complex operations of large-data.
Who knows? in couple of years RDB may find right HW and technologies to support large-data.

As (Voice in the Wind) told, all DB's currently have trade offs and their place. Even within NoSQL key-value, bigtable, mongoDB all has pros/cons.
Unfortunately DB engineers today will need to be thoroughly accustomed to all of them to make right decisions for business needs.

REPLY

24.

tb says:
JULY 26, 2012 AT 4:20 PM
Great Job ! best I have seen on this subject on the web. You should write a ebook and sell it.

REPLY

25.

umangapps says:
JULY 28, 2012 AT 3:47 AM
Great article
I need full information for no sql,i need to give seminaar on no sql ,plz email me important docs

REPLY

26.

Fabrice says:
AUGUST 10, 2012 AT 9:53 AM
Very clear. no more, no less

REPLY

27.

ravi says:
AUGUST 16, 2012 AT 5:07 AM
Great Article, crisp and clear.

REPLY

28.

Dean Hiller says:
AUGUST 16, 2012 AT 2:15 PM
Nice patterns, I noticed you are missing one that playorm(github) uses…partitioning a table and using the wide row pattern for indexing just the partition. This works great when you have trillions of small businesses on your system and only need to query info on each business from an application point of view.

REPLY

29.

Terry Cho says:

AUGUST 22, 2012 AT 3:56 PM
Very great article Thank you.

REPLY

30.

Pawan says:
AUGUST 28, 2012 AT 8:45 PM
Best article on nosql data mode
ling…thanks!

REPLY

31.

Jan says:
SEPTEMBER 18, 2012 AT 6:58 AM
The chart absolutely made my day, thanks!!! We do have a comparison about different NoSql databases: http://www.kammerath.co.uk/nosql-on-the-spot.html – check it out! Hope to see more charts like that…

REPLY

32.

Java Geek says:
OCTOBER 12, 2012 AT 9:41 AM
This is the kind of article, I was looking for long time which can explain some key concepts around NoSQL and where it is better over SQL . Great job, worth bookmarking

REPLY

33.

ami says:
OCTOBER 15, 2012 AT 10:17 AM
Great Blog,
just a small typo:" entires can be partitioned across multiple servers",
I guess you meant "entries"

REPLY

1.

Ilya Katsov says:
OCTOBER 16, 2012 AT 5:30 AM
Thanks for pointing this out. Fixed.

REPLY

34.

Baxter says:
OCTOBER 22, 2012 AT 3:26 PM
Ilya, I had this saved in Pocket forever but finally read it. Really great article, love the depth!

REPLY

35.

nickikt says:
NOVEMBER 24, 2012 AT 9:23 PM
Where does Datomic fit in all this?

REPLY

1.
   Ilya Katsov says:
   DECEMBER 1, 2012 AT 1:49 PM
   Roughly speaking, any data model can be decomposed into a set of key-value pairs. So, models like document, graph, or relational are essentially features over key-values model that can be supported by a database in varying degrees and in different combinations. I would say that Datomic provides document-oriented features with traits of graph-orientation.

   REPLY

36.
Thomas says:
NOVEMBER 26, 2012 AT 8:27 PM
Another way to deal with nested document flattening is to have each document be based on a skill instead of a person. So you would split the original single document into two documents for each skill and have:

```
{
"skill" : "Math",
"level" : "Low",
"name" : "John"
}

{
"skill" : "Poetry",
"level" : "High",
"name" : "John"
}
```

Then you could run the query "skill:Math AND level:High", and then remove all duplicate names.

REPLY

37.
Ignacio T. says:
JANUARY 3, 2013 AT 1:16 PM
Great Blog. It was very helpful for me.

REPLY

38.
jtomasrl says:
JANUARY 10, 2013 AT 4:12 AM

Excelent blog post, just working with mongoDB for some new projects. thanks a lot

REPLY

39.

Ravishankar Haranath says:
JANUARY 15, 2013 AT 6:09 AM
Very very informative article! Kudos to u.

REPLY

40.

Prashant says:
JANUARY 15, 2013 AT 12:16 PM
Excellent post, very helpful

REPLY

41.

Kalyan says:
JANUARY 17, 2013 AT 3:52 PM
Great composition of various modelling patters. Good work.

REPLY

42.

Robert says:
JANUARY 20, 2013 AT 7:14 PM
Great job!! Good info on comparing the noSQL data models!!

REPLY

43.

vlad says:
JANUARY 22, 2013 AT 4:44 PM
Ilya Katsov, this is a great article. Actually I think the structure of the presentation, the content and even some of the conclusions — are worty of a book. You should serious concider it. You are basically describing architectural patterns in data modeling that span structured/unstructured data and SQL/noSQL data stores.

REPLY

44.

surinder77 says:
MARCH 24, 2013 AT 12:02 PM
Superb Article

REPLY

45.

elie says:
JUNE 1, 2013 AT 12:56 PM
This is a great article, thanks for writing it!

REPLY

46.

Mark T says:

JULY 6, 2013 AT 11:57 AM

This is be of the most awesome tech articles I've ever read. Thank you for taking the time to pull this together, it has been a tremendous asset getting me up to speed on noSQL design.

REPLY

47.

Rex says:

SEPTEMBER 9, 2013 AT 5:47 PM

from an earlier comment from Robert…

"Relational modeling is typically driven by structure of available data" … Statements like this undermine the integrity of your otherwise useful article.

Robert,
Can I ask you what exactly is wrong?

I think a more accurate statement about relational modeling should be something like Relational modeling is typically driven by the modeling the business.

Once the relational data model represents the business, then update anomalies are avoided and reporting is just a matter joining the appropriate structures. (With noted performance issues at scale)

I agree each modeling technique and platform has an appropriate use, based on data volumes, available funding, data source and access patterns

Excellent post, thank you Ilya for putting in the effort to make it so thorough and professional!

REPLY

48.

Michael says:

SEPTEMBER 9, 2013 AT 11:11 PM

Was the F-bomb really necessary?

REPLY

1.

Ilya Katsov says:

SEPTEMBER 10, 2013 AT 2:41 PM

Definitely not

REPLY

49.

Allan Ramirez says:

SEPTEMBER 10, 2013 AT 1:53 PM

Reblogged this on Note To Self and commented:
Great post. I'm reblogging this for my reference and of course for spreading the information to others.

REPLY

50\.

sreeni says:
SEPTEMBER 25, 2013 AT 11:37 AM
super article. examples based on vendor e.g mongodb would have made it a much simpler for all to digest. will try to add examples

REPLY

51\.

Heather says:
OCTOBER 31, 2013 AT 6:17 PM
"NoSQL Data Modeling Techniques | Highly Scalable Blog" was in fact
a good blog post. If it had even more photos this might be perhaps even far better.
Thanks -Seymour

REPLY

52\.

Carlos Quijano says:
FEBRUARY 5, 2015 AT 4:53 PM
Probably one of the Best NoSQL articles I've read. Thanks.

REPLY

53\.

Andreea says:
MARCH 27, 2015 AT 11:11 AM
Another great example of NoSQL databases is CryptonorDB (cloud – mobile database). It delivers high availability, fault tolerant database service accessible via a RESTful HTTP/JSON API. Moreover, CryptonorDB safeguard the data by encrypting it: it manages the storage of encrypted data, but only the client manages the key. For more information, visit: http://cryptonordb.com/

REPLY

54\.

Kinnar says:
APRIL 23, 2015 AT 6:38 PM
This is a great article, thanks a ton for writing the same.

REPLY

55\.

frankgoortani says:
APRIL 30, 2015 AT 6:22 PM
Great article. most complete one I could find! Keep up the good work.

REPLY

56.

Justas Azna says:
MAY 14, 2015 AT 4:15 PM
Thank you for the article!. I'm about to design a NoSQL database structure for an app and you gave me a quite a few ideas.

REPLY

1.

dannyn08 says:
MAY 18, 2015 AT 11:03 PM
Justas, can you blog about it and how you connected the nosql database to your app and how you store and retrieve the data?

REPLY