JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Master of Computer Applications

18MCA301 – **NoSQL Databases**

Module -3

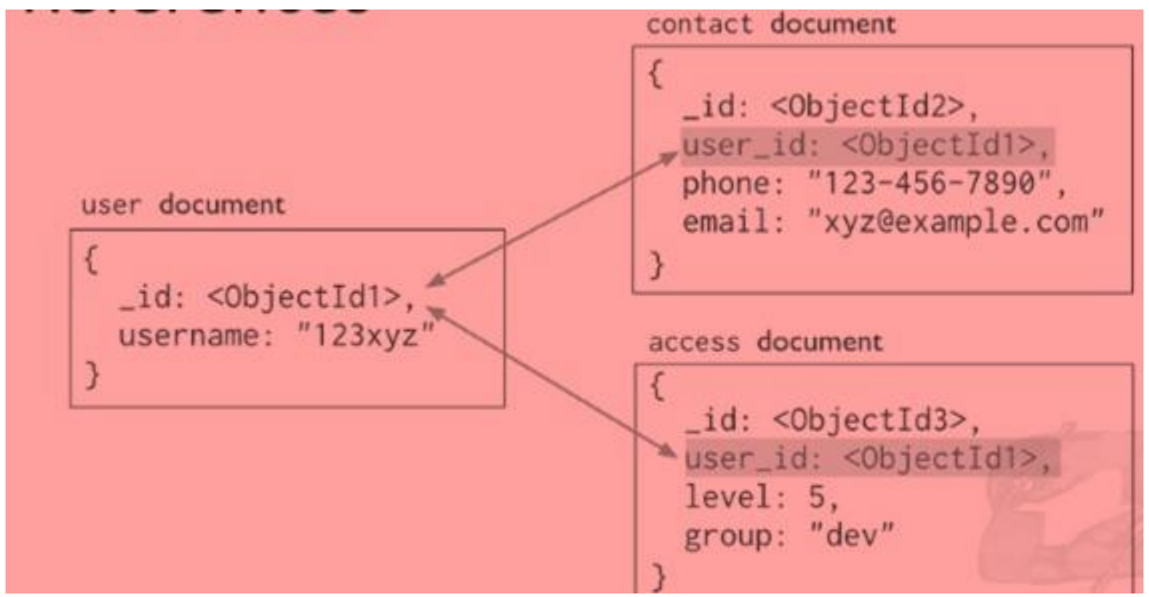
Document Oriented Databases

III General

Dr Gangothri

September 2020

- Document refers to a hierarchical structure of data that can contain substructures.
- Documents can consist of only binary data or plain text.
- They can be semi-structured when self-describing data formats like JavaScript Object Notation (JSON) or Extensible Markup Language (XML) are used.
- They can even be well-structured documents and always conform to a particular data model, such as an XML Schema Definition (XSD).
- Document NoSQL databases are flexible and schema agnostic, which means you can load any type of document without the database needing to know the document's structure up front.



```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

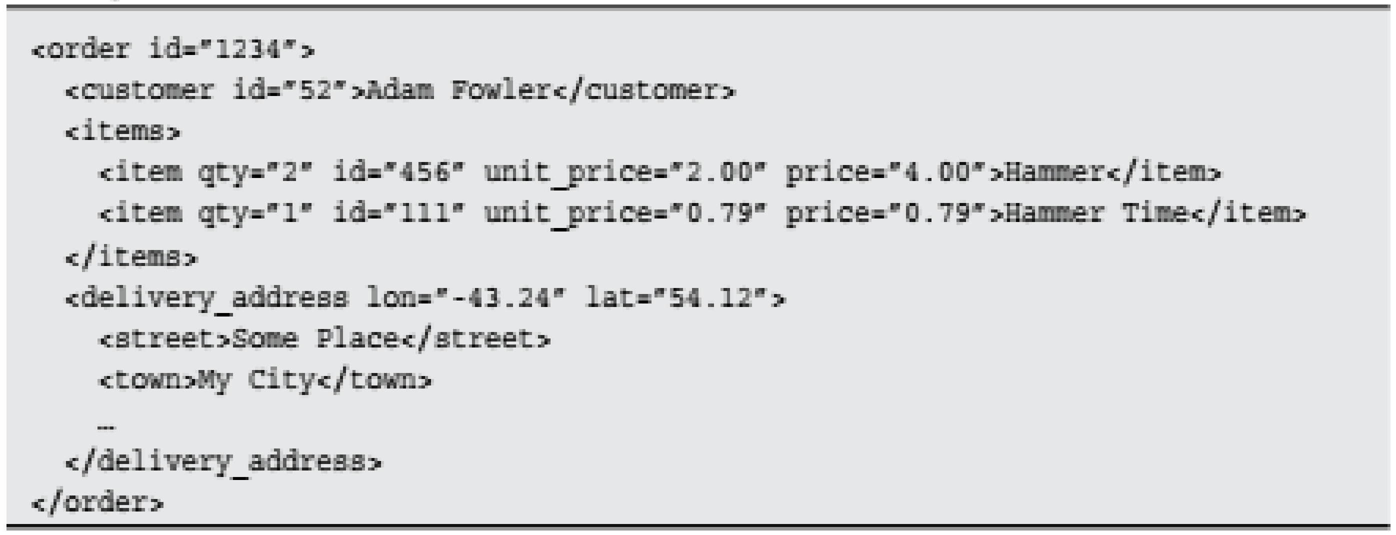
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Using a Tree-Based Data Model

- In a document NoSQL database, you can load any type of data without the database having prior knowledge of the data's structure or what the values mean.
- This flexibility makes these databases great both for prototyping a solution in an "agile" development process and for permitting changes in the stored data after a system goes live.
- Listing 14-1 is an example of an order document with enhanced information and ample flexibility.

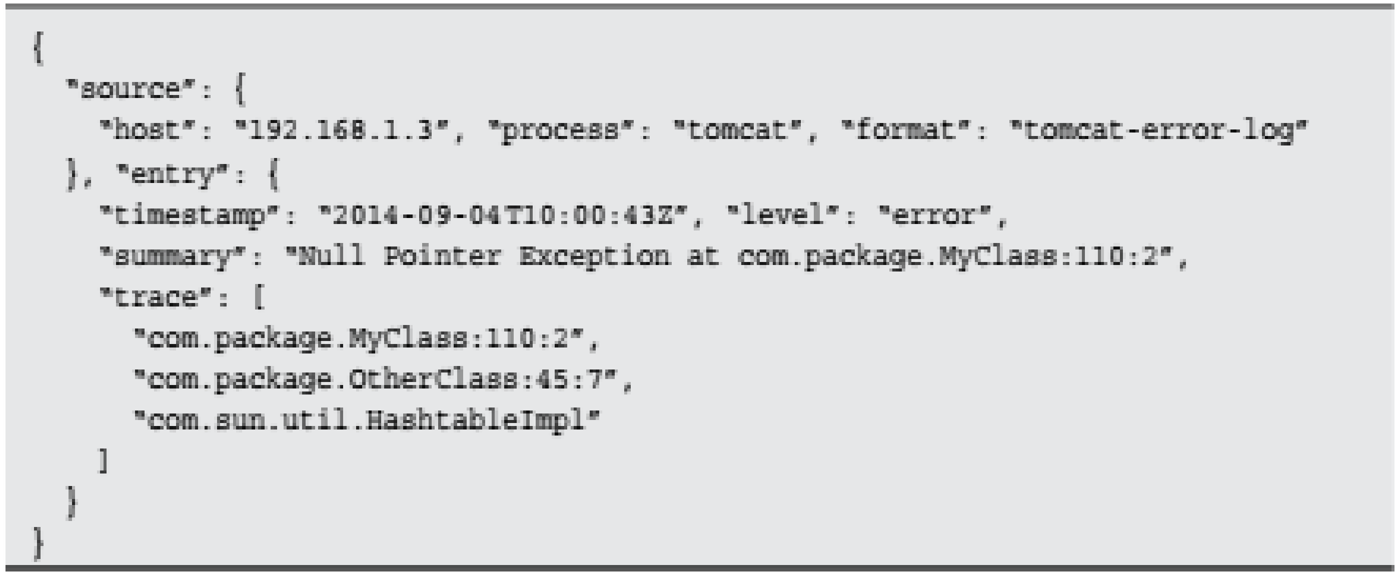
Listing 14-1: Order XML Document



```
<order id="1234">
  <customer id="52">Adam Fowler</customer>
  <items>
    <item qty="2" id="456" unit_price="2.00" price="4.00">Hammer</item>
    <item qty="1" id="111" unit_price="0.79" price="0.79">Hammer Time</item>
  </items>
  <delivery_address lon="-43.24" lat="54.12">
    <street>Some Place</street>
    <town>My City</town>
    …
  </delivery_address>
</order>
```

- Of course, you are free to use a less hierarchical, flatter, structure.
- Listing 14-2 shows a log file management as a tree structure.
- In this case, a stack trace error report can be a tree structure. You could, for example, dump information about every live executing process into a file, rather than just the section of code that reported the error. This approach is particularly useful for parallel debugging.

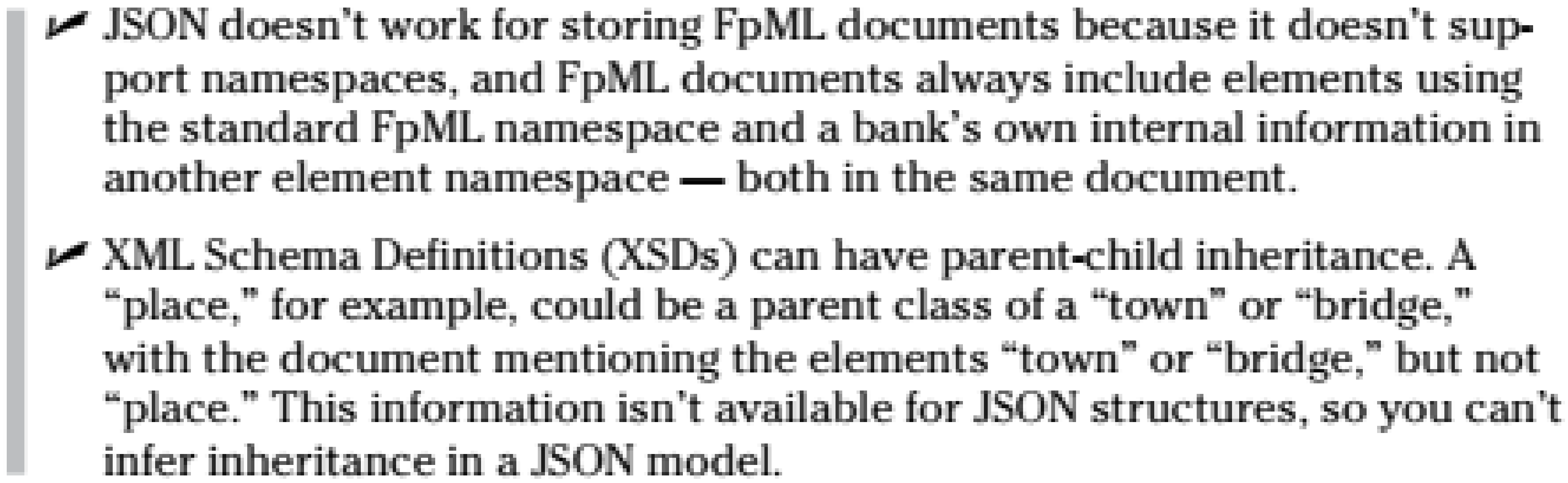
Listing 14-2: A mock Log File in a JSON tree structure



```
{
  "source": {
    "host": "192.168.1.3", "process": "tomcat", "format": "tomcat-error-log"
  }, "entry": {
    "timestamp": "2014-09-04T10:00:43Z", "level": "error",
    "summary": "Null Pointer Exception at com.package.MyClass:110:2",
    "trace": [
      "com.package.MyClass:110:2",
      "com.package.OtherClass:45:7",
      "com.sun.util.HashtableImpl"
    ]
  }
}
```

Managing trades in financial services

- Financial products Markup Language (FpML) documents are a particular XML schema structure.
- They’re used for trading in long-running financial derivatives.
- There are a couple of reasons that FpML need an XML oriented document database rather than a JSON oriented one, or need to be stored in a relational database:



- ✓ JSON doesn't work for storing FpML documents because it doesn't support namespaces, and FpML documents always include elements using the standard FpML namespace and a bank's own internal information in another element namespace — both in the same document.
- ✓ XML Schema Definitions (XSDs) can have parent-child inheritance. A “place,” for example, could be a parent class of a “town” or “bridge,” with the document mentioning the elements “town” or “bridge,” but not “place.” This information isn't available for JSON structures, so you can't infer inheritance in a JSON model.

SCHOOL OF COMPUTER SCIENCE AND IT

Discovering document structure

- Document databases tend to store documents in a compressed on-disk format
- In order to do so, the databases need to understand the format of the documents they receive.
- When you submit JSON or XML documents, for example, a database uses that structure to better manage the data on disk.
- MongoDB stores -BSON binary representation
- MarkLogic Server -XML documents
- Microsoft's DocumentDB - JSON document
- Collections of files may contain a combination of
  - Structured files (such as CSV expense information)
  - Semi-structured files (such as XML and saved HTML web pages)
  - Unstructured files (such as JPEG images, movies, MP3s, word documents, PDFs, and plain text files)

- In reality, unstructured formats are actually semi-structured For example, JPEG images may contain metadata about
  - The camera that took the images
  - The prevailing conditions when the image was shot
  - The GPS coordinates and elevation where the image was taken
- Some databases come with built-in support for extracting this metadata and plain text from binary files.
- MarkLogic Server, for example, includes support for more than 200 binary formats through its use of binary data extraction libraries. These provide an XHTML output, using meta tags for metadata and the body tag for text content.
- Many document databases support the concept of a URI path as a primary key, or a unique document ID.

Document Databases as Key-Value Stores

- Document databases make great key-value stores.
- You can use a document URI path to represent a composite key like those in a key-value store.
- You can use document properties/elements, or metadata fields, to control how the database partitions data.
- Modeling values as documents:
  - Values can be binary information stored as a document.
  - In many uses of a key-value store, though, values are JSON or XML structures.
  - Document databases provide in-memory read caches, with some (MongoDB) even providing read-only replicas to allow greater parallel reads of the same data.
- Using value information:
  - Once the elements/attributes (XML) or properties (JSON) are indexed, you can perform data queries and aggregation processes over them.
  - Both MarkLogic Server and Microsoft DocumentDB provide user-defined functions (UDFs). These are server-side functions that take the set of documents matching a query and perform aggregation calculations on them.
  - These aggregations can be a mean average, a standard deviation, or any other scalar output you devise.

Patching Documents

- A read, modify, update (RMU) operation on the entire content of a document is quite expensive in terms of processing time, and in many NoSQL databases isn't ACID — meaning another operation could update the document between your application's read and update steps!
- <u>Supporting partial updates:</u>
  - A partial update is one where you are updating just one of two values in a document, rather than replacing the whole thing. Perhaps there is a field that holds the number of times a document has been read, or the current product quantity in a warehouse.
- <u>Streaming changes:</u>
  - The append operation enables you to handle a range of streaming cases efficiently. This operation will identify where in a document new data needs adding, and insert the data.
  - This prevents performance impacts caused by the alternative read-modify-update approach.
- <u>Providing alternate structures in real time:</u>
  - Denormalization duplicates some information at ingestion time in order to provide access to high-speed reads and queries. Duplication of data is done so that the database doesn't have to process joins at query time.

Examples of using denormalization include

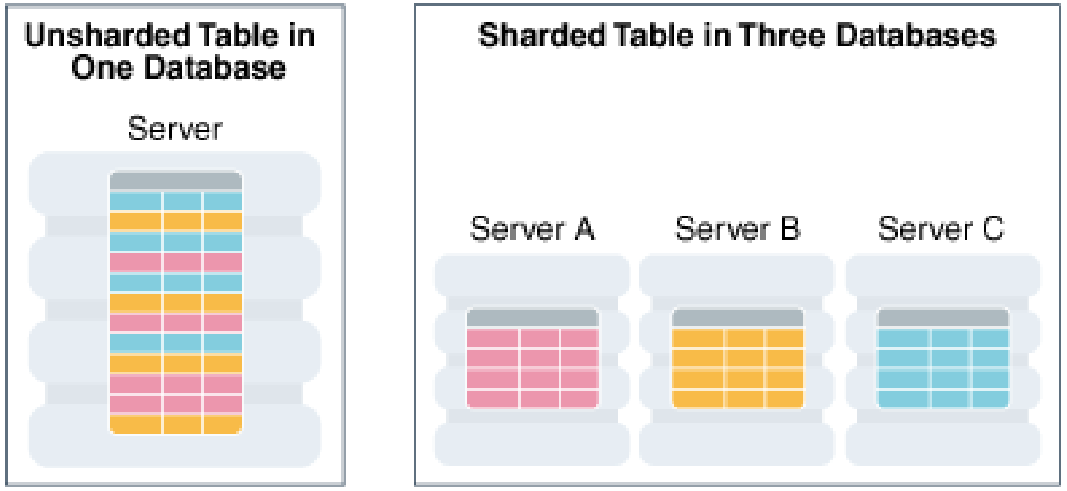
- Updating a summary document showing the latest five news items when a new item arrives.
- Updating multiple searchable program-availability documents on a catchup TV service when a new scheduling document arrives. This merges data from an episode, genre, brand, and scheduling set of documents into multiple program-availability documents.
- Taking an order document and splitting it into multiple order-item documents in order to allow a relational business intelligence tool to query at a lower granularity

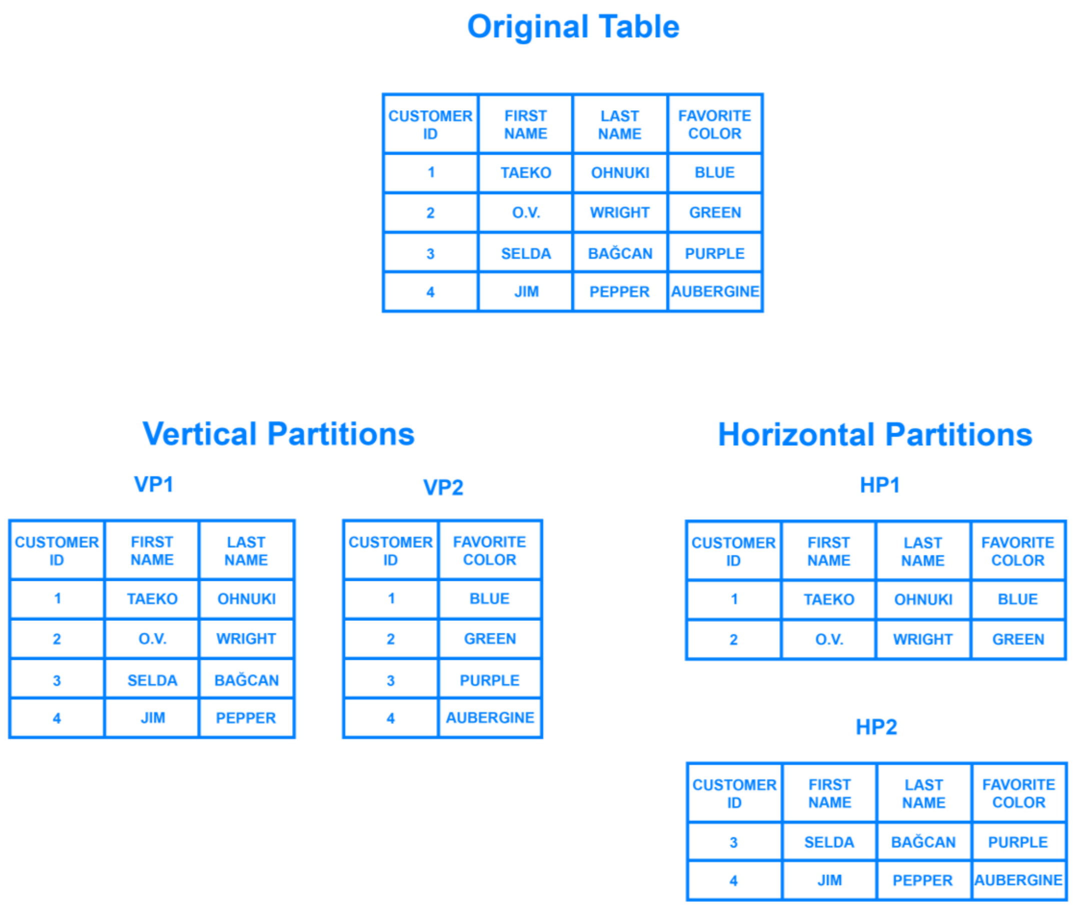
These use cases require the following database features:

- If they're easy to update, you can generate these alternative structures in a pre-commit trigger. This means the denormalizations are generated as the new document arrives. When the transaction to add documents ends, the denormalizations will become visible at exactly the same time.
- You can use a post-commit trigger if either of the following occur:
  - Immediate consistency in these views isn't required.
  - Ingest speed is more important than consistency.
- A post-commit trigger allows the transaction to complete and guarantees that from that point, the denormalizations are generated. This improves the speed of writes at the cost of a few seconds of inconsistency.

Sharding

- Sharding is the process of ensuring that data is spread evenly across an entire cluster of servers.
- Shards can be set up at the time a cluster is implemented (MongoDB)
- They can be fixed into a number of buckets or partitions that can be moved later (Couchbase, Microsoft DocumentDB)
- They can be managed automatically simply by moving a number of documents between servers to keep things balanced (MarkLogic Server)


Unsharded Table in One Database
Server
Sharded Table in Three Databases
Server A
Server B
Server C


Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

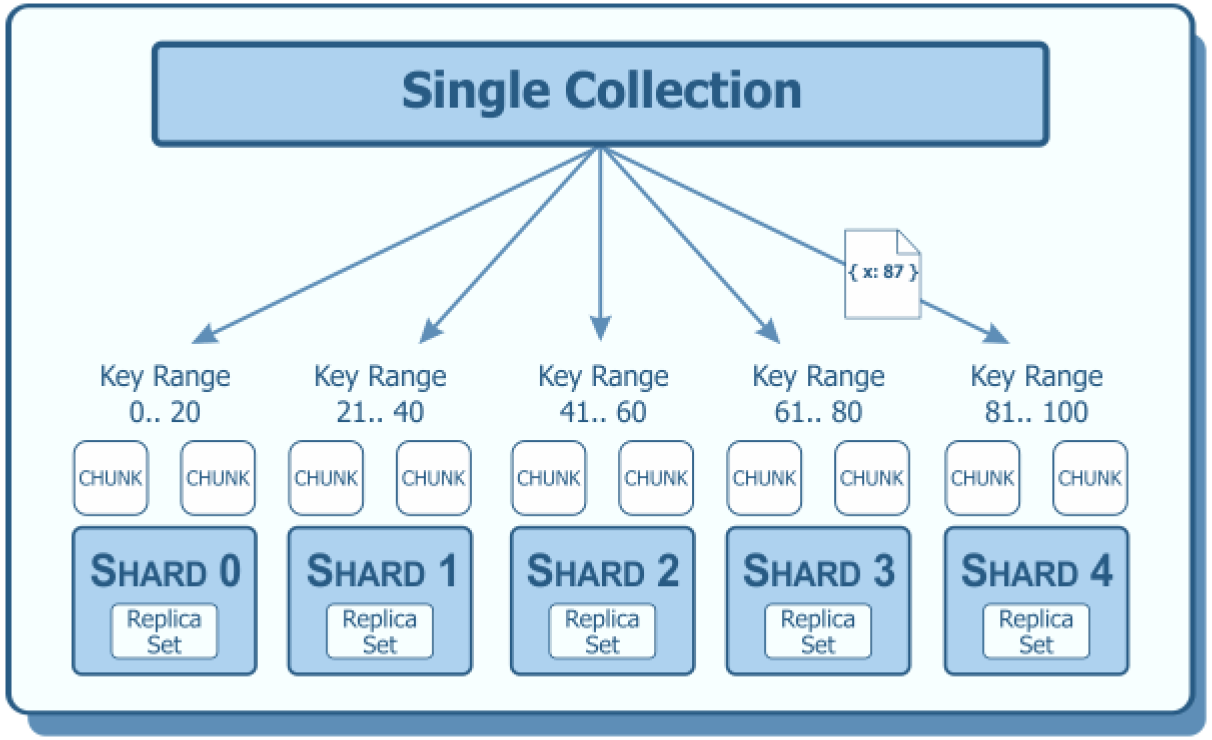
HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

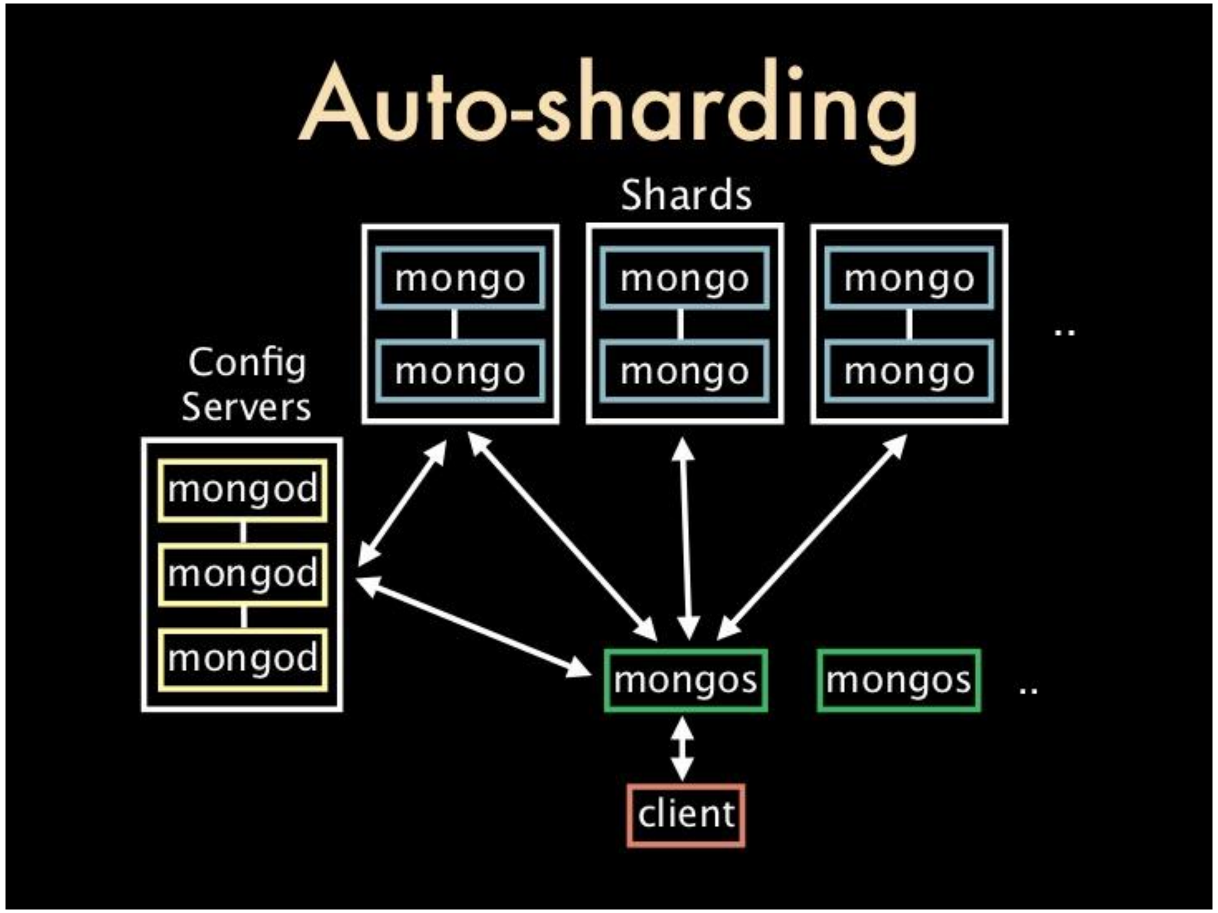
CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Key-based sharding



- With key-based sharding, the key — the name, URI, or ID — of a document determines which server it's placed on.
- Based on the key of the record and the ranges assigned to each server, a client connector can determine exactly which server to communicate with in order to fetch a particular document.
- MongoDB and Couchbase database drivers use this approach to sharding.
- Some document NoSQL databases, such as MongoDB, allow their replicas to be queried, rather than exist purely as a backup for a primary shard. This allows for greater read parallelization.
- This splits the document access load between both the primary shard and its replicas, increasing overall cluster query performance

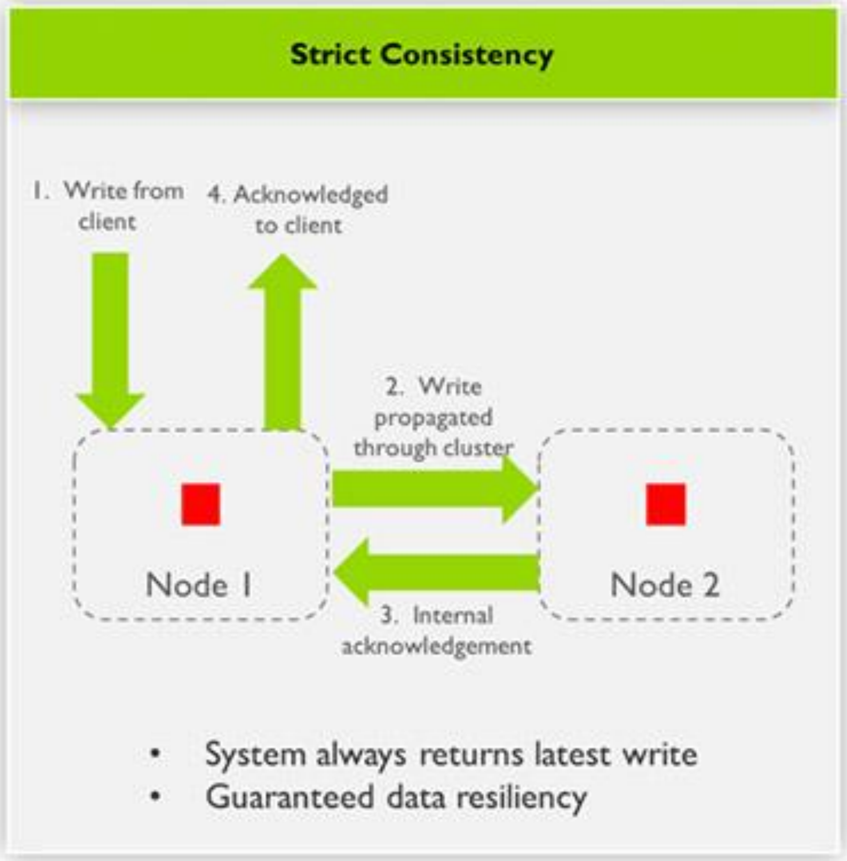
Automatic sharding

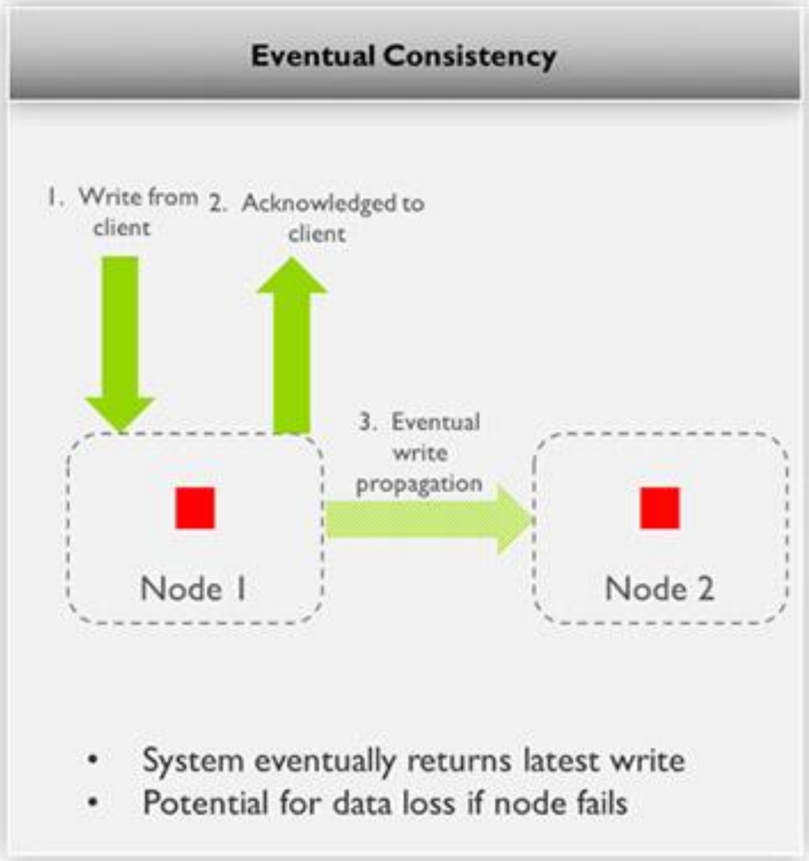


- With automatic sharding, the database randomly assigns a new document to a server, which means the database's developer doesn't have to carefully select a key in order to ensure good write throughput.
- Automatic sharding works well for a live cluster, but if you need to scale outor scale back during peak periods, you'll need to rebalance your partitions.
- Rebalancing automatically, rather than based on key range for each server, is an easier operation
- In some databases this rebalancing, or fixed range repartitioning, has to be initiated manually (Couchbase).
- Some document databases, like MarkLogic Server, perform this rebalancing live as it needs to.
- This evens out rebalancing load over time, rather than having to impact cluster performance when manually forced in a short time window.

Managing Consistency

- It’s important to understand the differences in approaches when considering a database for your application. Not all NoSQL databases support full ACID guarantees, unlike their relational database management systems counterparts.


Strict Consistency
1. Write from client
4. Acknowledged to client
2. Write propagated through cluster
Node 1
Node 2
3. Internal acknowledgement
• System always returns latest write
• Guaranteed data resiliency


Eventual Consistency
1. Write from client
2. Acknowledged to client
3. Eventual write propagation
Node 1
Node 2
• System eventually returns latest write
• Potential for data loss if node fails

Using eventual consistency

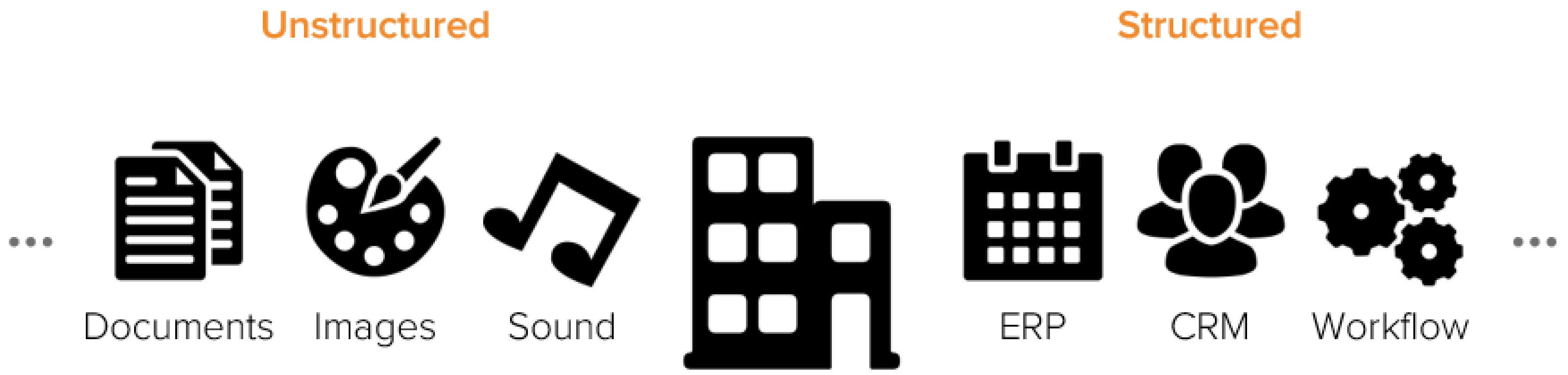
- With eventual consistency, a write operation is successful on the server that receives it but all replicas of that data aren’t updated at the same time.
- They are updated later based on system replication settings.
- Some databases provide only eventual consistency (Couchbase), whereas others allow tuning of consistency on a per operation basis, depending on the settings of the originating client request (MongoDB, Microsoft DocumentDB).
- Most social networks use this consistency model for new posts.
- This model gives you very fast write operations, because you don’t have to wait for all replicas to be updated in order for the write operation to be complete.
- Inconsistency tends to last only a few seconds while the replicas catch up.

Using ACID consistency

- ACID consistency is the gold standard of consistency guarantees. An ACID-compliant database ensures that:
  - ✓All data is safe in event of failure.
  - ✓Database updates always keep the database in a valid state (no conflicts).
  - ✓One set of operations doesn’t interfere with another set of operations.
  - ✓Once the save completes, reading data from any replica will always give the new “current” result.
- Some ACID databases go further and allow several changes to be executed within the same transaction. These changes are applied in a single set, ensuring consistency for all documents affected.
- Consistency is achieved by shipping all the changes you want applied from the server where a transaction is started, to each replica, then applying the changes, and if all is well the transaction completes.
- If any one action fails, the entire transaction of changes is rolled back on all replicas.
- Transaction roll back ensures the data is kept in a consistent state.
- MarkLogic Server provides ACID transactions both on the server-side and across several client requests in an application.
- Microsoft’s DocumentDB provides ACID transactions only on the server-side, when executing a JavaScript stored procedure.

Managing Changing Data Structures

- Some applications, however, consolidate data from a number of systems within an organization, as well as third party or public data, in order to provide a more flexible application.
- This provides for a rich array of information with which to mine for useful business answers.
- Organizations don’t have control over external data-feed formats, and they can be changed at any time without warning.
- Even worse, many data structures in use today change depending on the application that creates them, rather than the system that stores them.



Handling variety

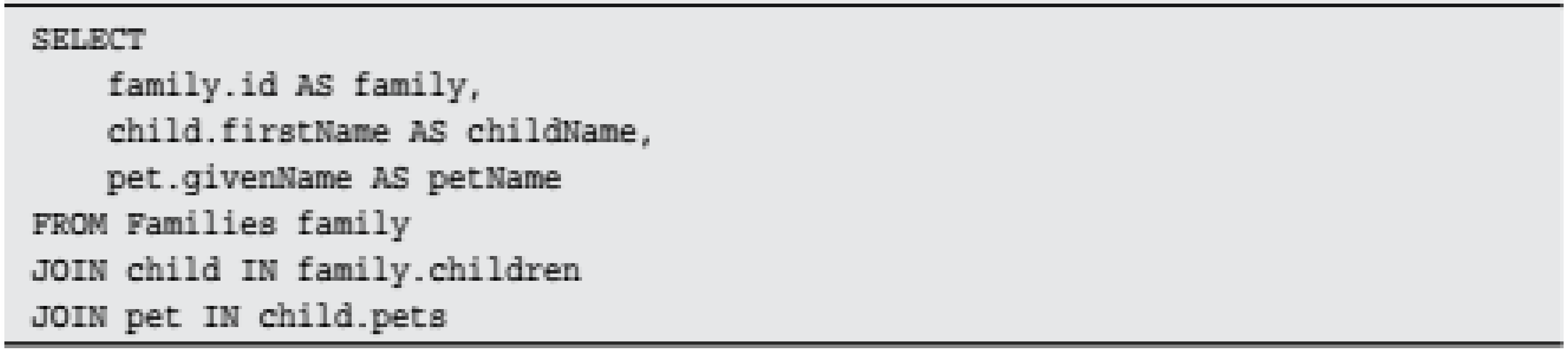
- Storing and retrieving variable information is a key benefit of all NoSQL document databases. Indexing and searching their structure, though, is difficult.
- Many document databases don’t perform these operations natively; instead, they use an external search engine such as Solr (MongoDB) or Elasticsearch (Couchbase) to do so.
- These search engines typically index either the text in the document or specifically configured elements within the document.
- Microsoft’s DocumentDB and MarkLogic Server both take a different approach. They have a universal index that examines the documents as they’re ingested. They both index the structure as well as the values.
- This enables you to perform element-value queries (exact match queries) as soon as a document is stored.

Managing change over time

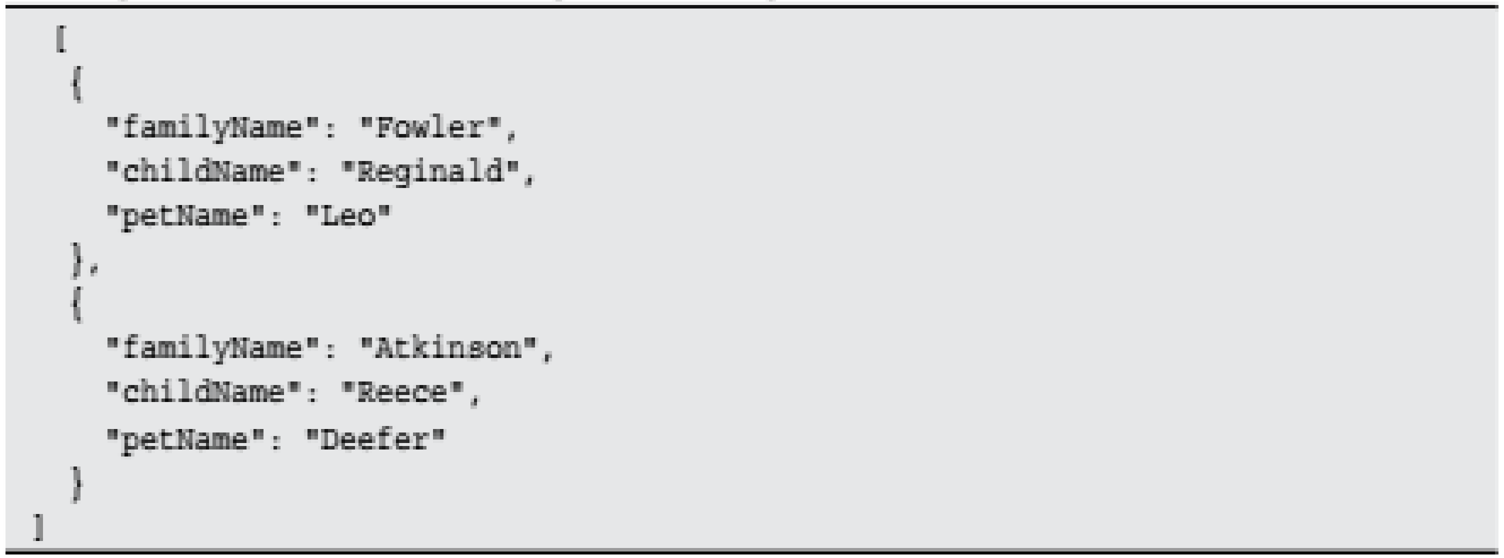
- If you change the format of documents, you need to rework your search index configurations.
- Having to manually reconfigure indexes delays the handling of new types of content. You have to perform index changes before making use of the data.
- Having a universal structural index allows you to search and explore the data you’ve ingested and then decide to perform different queries over it.
- You don’t have to manually configure the structure before loading content in order to search it immediately

Providing a Familiar Developer Experience

- A database needs to be easy to get started with, and it needs to provide powerful tools that don't require lots of legwork and that they can put to use in five minutes.
- **Indexing all your data:**
  - Having a schema-free database is one thing, but in order to have query functionality, in most NoSQL databases you must set up indexes on specific fields.
  - Microsoft's DocumentDB and MarkLogic Server however, both provide a universal index, which indexes the structure of documents and the content of that document's elements.
  - DocumentDB provides structural indexing of JSON documents and even provides range queries (less than, greater than operations) alongside the more basic equal/not equal operations. DocumentDB does all this automatically.
- **Using SQL:**
  - Microsoft provides a RESTful (Representational State Transfer) API that accepts SQL as the main query language.
  - This SQL language is ANSI SQLcompliant, allowing complex queries to be processed against DocumentDB using the exact syntax that you would against a relational database.



```
SELECT
    family.id AS family,
    child.firstName AS childName,
    pet.givenName AS petName
FROM Families family
JOIN child IN family.children
JOIN pet IN child.pets
```



```
 [
  {
    "familyName": "Fowler",
    "childName": "Reginald",
    "petName": "Leo"
  },
  {
    "familyName": "Atkinson",
    "childName": "Reece",
    "petName": "Deefer"
  }
]
```

- **Linking to your programming language:**
  - You can use and serialize plain .NET objects directly to and from DocumentDB.
  - These object classes can have optional annotations, which helps migrate an “id” field in the JSON representation to the .NET object’s“Id” field.
  - So, when using DocumentDB, programmers just use the standard coding practices they’re used to in .NET.
- **Evaluating Microsoft DocumentDB:**
  - With tunable consistency, a managed service cloud offering, a universal index for JSON documents with range query support, and JavaScript server-side scripting, DocumentDB is powerful enough for many public cloud NoSQL scenarios.
  - NET developers with exposure to SQL or Microsoft SQL Server will find it easy to understand DocumentDB. As a result, DocumentDB will challenge MongoDB in the public cloud — because DocumentDB is more flexible and has better query options when a JSON document model is used.
  - For private cloud providers and large organizations able to train people on how to maintain free, open-source alternatives, MongoDB will likely maintain its position.
  - MarkLogic Server is likely to continue to dominate in cases of advanced queries with geospatial and semantic support; or where binary, text, and XML documents with always-ACID guarantees need to be handled; or in highsecurity and defense environments.

Providing an End-to-End Document Platform

- XML is the lingua franca (default language) of systems integration. It forms the foundation of web services and acts as a self-describing document format.

1. In addition to XML, plain text documents, delimited files like comma separated values (CSV), and binary documents still need to be managed.
   – If you want to store these formats, extract information from them, and store metadata against the whole result, then you need a more comprehensive end-to-end solution.
2. After managing the data, you may also need to provide very advanced full text and geospatial and semantic search capabilities.
   – Perhaps you also want to return faceted and paginated search results with matching text snippets, just like a search engine.
3. Perhaps you want to go even further, and take the indexes of all the documents and conduct co-occurrence analysis.
4. This analytic capability is more than summing up or counting existing fields. It requires more mathematical analysis that needs to be done at high speed, preferably over in-memory, ordered range indexes.

- ***Ensuring consistent fast reads and writes***:
- MarkLogic Server writes all data to memory and writes only a small journal log of the changes to disk, which ensures that the data isn’t lost.
- If a server goes down, when it restarts, the journal is replayed, restoring the data.
- Writing a small journal, rather than all the content, to disk minimizes the disk I/O requirements when writing data, thus increasing overall throughput.
- Many databases also lock information for reads and writes while a write operation is updating a particular document. MongoDB currently locks the entire database within a given instance of MongoDB!
- To avoid this situation, MarkLogic Server uses the tried-and-true approach called multi-version concurrency control (MVCC). Here’s what happens if you have a document at an internal timestamp version of 12:

  ❑ When a transaction starts and modifies the data, rather than change the data, an entirely new document is written.

  ❑ Once the document transaction finishes, the new version is given the timestamp version of 13:

  • All reads that happen after this point are guaranteed to see only version 13. • Replicas are also updated within the transaction, so even if the primary server fails, all replicas will agree on the current state of the document.

  ❑ All changes are appended as new documents to the end of a database file, both in-memory and on disk. In MarkLogic, this is called a stand. There are many stands within a forest, which is the unit of partitioning in MarkLogic Server. A single server manages multiple forests. They can be merged or migrated as required while the service is live, maintaining a constant and consistent view of the data.

- ***Supporting XML and JSON***:
  - MarkLogic Server is built on open standards like XML, XQuery, XPath, and XSLT. XQuery is the language used for stored procedures, reusable code modules, triggers, search alerts, and Content Processing Framework (CPF) actions.
  - MarkLogic can natively store and index XML and plain text documents, store small binary files within the stands, and manage large binaries transparently directly on disk (but within the managed forest folder).
  - You handle JSON documents by transposing them to and from an XML representation with a special XML namespace, which in practice, is handled transparently by the REST API.
  - The XML representation on disk is highly compressed. Each XML element name is very text-heavy, and there's a start tag and an end tag. Rather than store this long string, MarkLogic Server uses its own proprietary binary format.
- ***Using advanced content search:***
  - MarkLogic Server was built to apply the lessons from search engines to schema-less document management. This means the same functionality you expect from Microsoft FAST, HP Autonomy, IBM OmniFind, or the Google Search Appliance can be found within MarkLogic Server.
  - In addition to a universal document structure and value (exact-match) index, MarkLogic Server allows range index (less than, greater than) operations.
  - This range query support has been extended to include bounding box, circle radius, and polygon geospatial search, with server-side heat map calculations and "distance from center of search" analytics returned with search results.

Securing documents

- Many of MarkLogic’s customers are public sector organizations in the United States Department of Defense. In these areas, securing access to information is key. MarkLogic Server has advanced security features built in.
- MarkLogic supports authentication of users via encrypted passwords held in the database itself, or in external systems.
- These external systems include generic LDAP (Lightweight Directory Access Protocol) authentication support and Kerberos Active Directory token support, which allows single sign-on functionality when accessing MarkLogic from a Windows or Internet Explorer client.
- Digital certificates are also supported for securing access.
- Authorization happens through roles and permissions.
- Users are assigned roles in the system.
- These roles may come from internal settings or, again, from an external directory through LDAP. Permissions are then set against documents for particular roles. This is called role based access control (RBAC).

- If you add two roles to a system, both with read access to the same document, then a user with either one of those roles can access the document.
- In some situations, though, you want to ensure that only users with all of a set of roles can access a document. This is called compartment security and is used by defense organizations worldwide.
- Security permissions in MarkLogic are also applied to every function in the system. A set of execute permissions is available to lock down functionality.
- Custom permissions can be added and checked for by code in these functions.
- Execute permissions can even be temporarily bypassed for specific application reasons using amps — amps are configuration settings of a MarkLogic application server to allow privileged code to be executed only in very specific circumstances.
- All these features combined provide what MarkLogic calls government-grade security.

Providing a Web Application Back End

- NoSQL database is flexible enough to handle changes in document schema so that old and new versions of your app can coexist with the same data back end.
- If you support multiple platforms, you need this model to work seamlessly across platforms without any complex document parsers, which probably means JSON rather than XML documents.
- If you're building a web app that communicates either directly to the database or via a data services API that you create, then you want this functionality to be simple, too.
- In this case, a JSON data structure that can be passed by any device through any tier to your database will make developing an application easier.
- MongoDB, especially running as a managed cloud service, provides an inexpensive and easy-to-start API required for these types of applications.
- The software doesn't provide all the advanced query and analytics capabilities of Microsoft's DocumentDB or MarkLogic Server, but that isn't a problem because its primary audience consists of web application developers in startup companies.

Trading consistency for speed

- In many mobile and social applications, the latest data doesn’t return with every query.
- Allowing this inconsistency means that the database doesn’t have to update the master node and the replicas during a write operation.
- Trading this consistency in a document database like MongoDB means that the database doesn’t have to update the master node and the replicas during a write operation.
- MongoDB also supports writing to RAM and journaling to disk, so if you want durability of data with eventual consistency to replicas then MongoDB can be used for this.
- This allows MongoDB to provide high write speeds.
- MongoDB currently locks its database file on each MongoDB instance when performing a write. This means only one write can happen at a time.
- Even worse, all reads are locked out until a write completes.
- To work around this problem, run multiple instances per server (perhaps one per two CPU cores). This approach is called micro-sharding.
- MongoDB micro-sharding you have to run multiple copies of MongoDB on each server, each with a single shard. This is of course a higher CPU and I/O load on the server.

Sticking with JavaScript and JSON

- Developers want to use their existing set of skills when it comes to operating database.
- They have to save and retrieve JSON.
- When they query, they want a query definition that's in JSON.
- They might not be a relational database expert with years of Structured Query Language (SQL) experience; however, it is needed with same functionality exposed in a way familiar to your JavaScript and JSON fluent web developers.
- MongoDB's query mechanism uses a JSON structure.
- The response you get is a JSON result set, with a list of JSON documents; it's all very easy to understand and use from a web developer's perspective, which is why MongoDB is so popular..

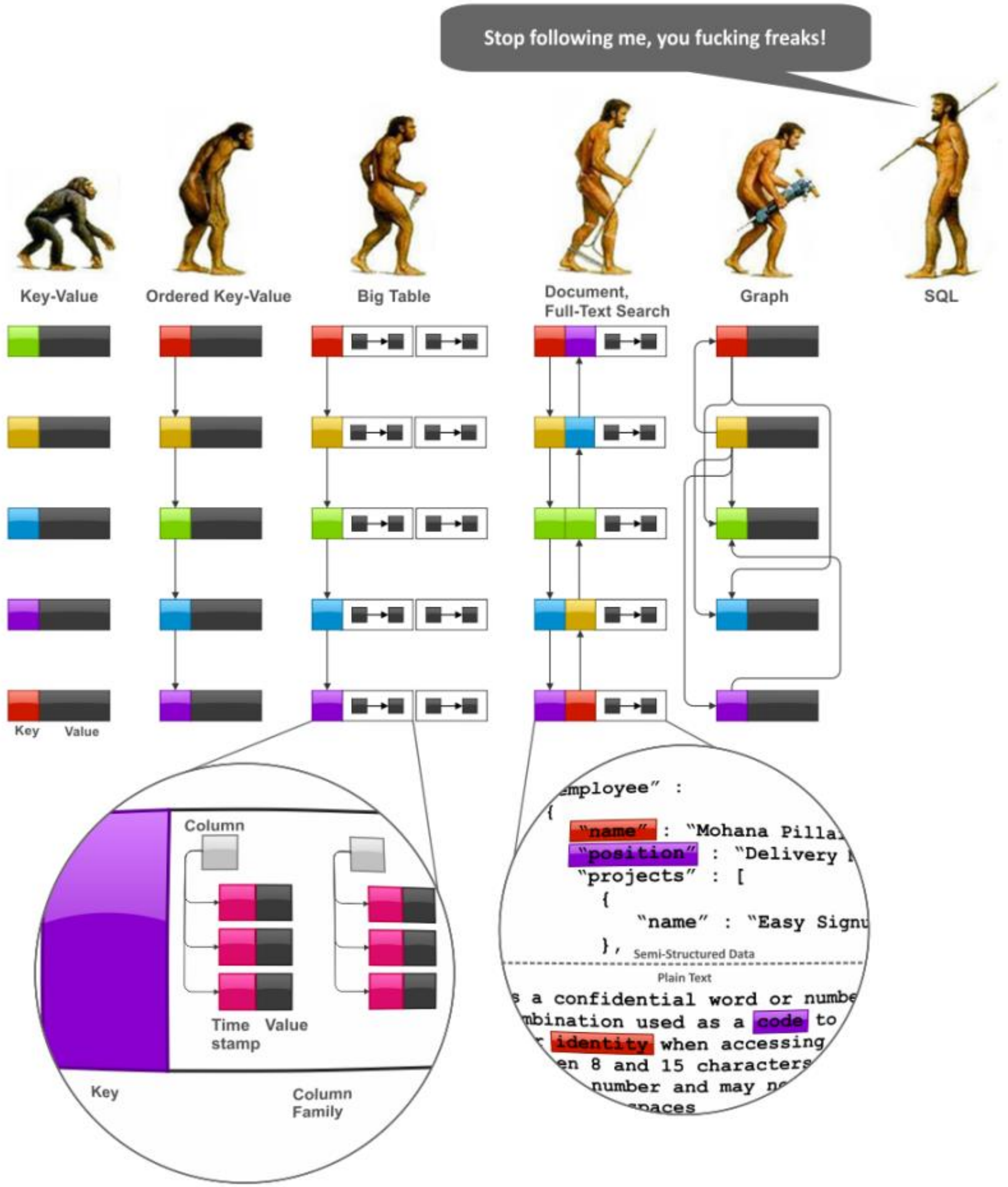
Finding a web community

- MongoDB’s simplicity and JavaScript-centric characteristics make it a natural starting place for developers.
- This is reflected by the strength of the MongoDB online community
- Partly it’s because MongoDB, Inc. (formerly 10gen) has an excellent developercentric, local meet-up strategy and strong marketing behind it.
- Every week in MongoDB offices worldwide, people show up and work on apps with their sales and consulting staff.
- MongoDB is also prevalent on web forums.

Evaluating MongoDB

- MongoDB is a solid database when used as a back end for web applications.
- Its JavaScript and JSON centricity make it easy to understand and use straightaway.
- Currently, the main thing limiting MongoDB's use in mission-critical enterprise installations — as opposed to large enterprises that are using MongoDB as a cache or for noncritical operations — is its lack of enterprise features.
- The recent 2.6 version did introduce rolling backups, data durability settings, and basic index intersection.
- Basic RBAC (role based access control, mentioned earlier in this chapter) was also added, but not at the document level.
- Also, fundamental changes need to be made to MongoDB's architecture to allow better scaling.
- Support for multi-core processors that doesn't require you to start multiple instances is one such change.
- Another is the need to define a compound index.
- Say that you have a query with three query terms and a sort order. In order to facilitate this query, rather than add an index for each term and define the sort order in the query, you must define a compound index for every combination of query terms you use and their sort order.

- This approach requires a lot of indexing and index space.
- MongoDB has begun to address this issue, but it hasn’t entirely removed the need for compound indexes.
- The database-wide write lock is also a problem in systems that require large parallel write operations.
- This fundamental design exists because MongoDB uses memory-mapped files.
- An entirely new persistence mechanism will be required to resolve this, and this will take time to test, debug, and prove.
- The bottom line is that MongoDB is easy to grasp and is already used by many companies to build very useful applications.
- If you have a web application or need a JSON data model in a NoSQL database with built-in querying, consider adopting MongoDB, especially if you need a free database or a private installation. Currently, neither of these scenarios are supported by Microsoft’s DocumentDB.


Stop following me, you fucking freaks!
Key-Value
Ordered Key-Value
Big Table
Document,
Full-Text Search
Graph
SQL
Key
Value
Column
Time stamp
Value
Key
Column Family
employee" :
{
"name" : "Mohana Pilla
"position" : "Delivery
"projects" : [
{
"name" : "Easy Signu
},
Semi-Structured Data
Plain Text
s a confidential word or numbe
mbination used as a code to
identity when accessing
en 8 and 15 characters
number and may n
spaces