



Master of Computer Applications

18MCA301 – NoSQL Databases

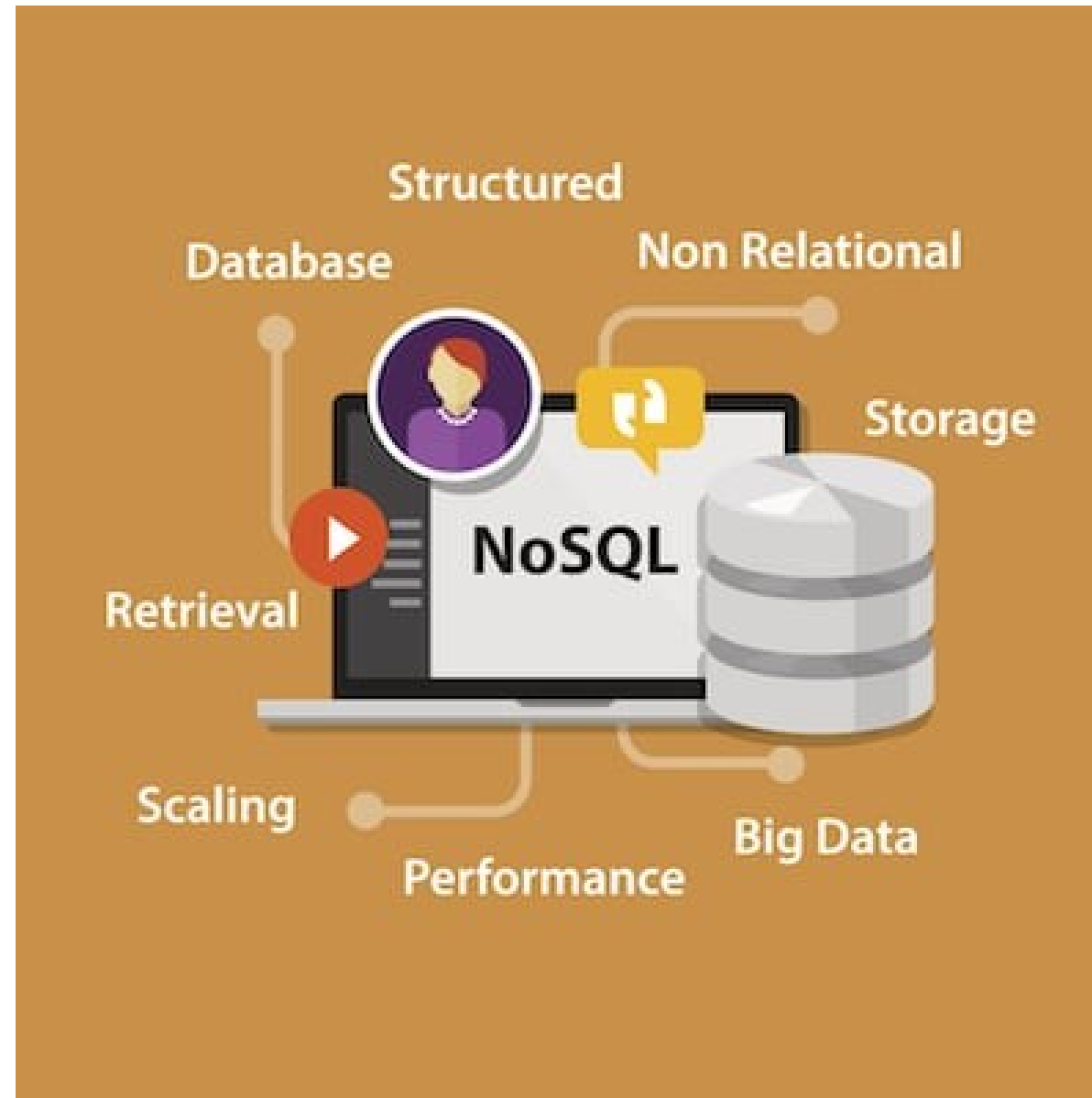
Module -2

Scaling & Visualizing NoSQL

III General

Dr Gangothri

September 2020



Consistency

- The consistency property of a database means that once data is written to a database successfully, queries that follow are able to access the data and get a consistent view of the data.
- In practice, this means that if you write a record to a database and then immediately request that record, you're guaranteed to see it.
- In the NoSQL world, consistency generally falls into one of two camps:
 - ▶ **ACID Consistency** : ACID means that once data is written, you have full consistency in reads.
 - ▶ **Eventual Consistency (BASE)**: BASE means that once data is written, it will eventually appear for reading.

ACID

- ACID basically means, “This database has facilities to stop you from corrupting or losing data”
- In fact, the vast majority of NoSQL databases don’t provide ACID guarantees.
- ACID transactional consistency can be provided various ways:
 - In the locking model, you stop data from being read or written on the subset of information being accessed until the transaction is complete, which means that during longer-running transactions, the data won’t be available until all of the update is committed.
 - An alternative mechanism is multiversion concurrency control (MVCC), which bears no resemblance to document versioning; instead, it’s a way of adding new data without read locking.

BASE

- BASE means that rather than make ACID guarantees, the database has a tunable balance of consistency and data availability.
- This is typically the case when nodes in a given database cluster act as primary managers of a part of the database, and other nodes hold read-only replicas.
- To ensure that every client sees all updates (that is, they have a consistent view of the data), a write to the primary node holding the data needs to lock until all read replicas are up to date.
- This is called a two-phase commit — the change is made locally but applied and confirmed to the client only when all other nodes are updated.
- BASE relaxes this requirement, requiring only a subset of the nodes holding the same data to be updated in order for the transaction to succeed.
- Sometime after the transaction is committed, the read-only replica is updated.
- The advantage of this approach is that transactions are committed faster.
- Having readable live replicas also means you can spread your data read load, making reading quicker.

Choosing ACID or BASE?

- Some NoSQL databases have ACID-compliance on their roadmap, even though they are proponents of BASE, which shows how relevant ACID guarantees are to enterprise, mission-critical systems.
- Many companies use BASE-consistency products when testing ideas because they are free but then migrate to an ACID-compliant paid-for database when they want to go live on a mission-critical system.
- The easiest way to decide whether you need ACID is to consider the interactions people and other systems have with your data.
- For example, if you add or update data, is it important that the very next query is able to see the change?
- In other words, are important decisions hanging on the current state of the database?
- Would seeing slightly out-of-date data mean that those decisions could be fatally flawed?

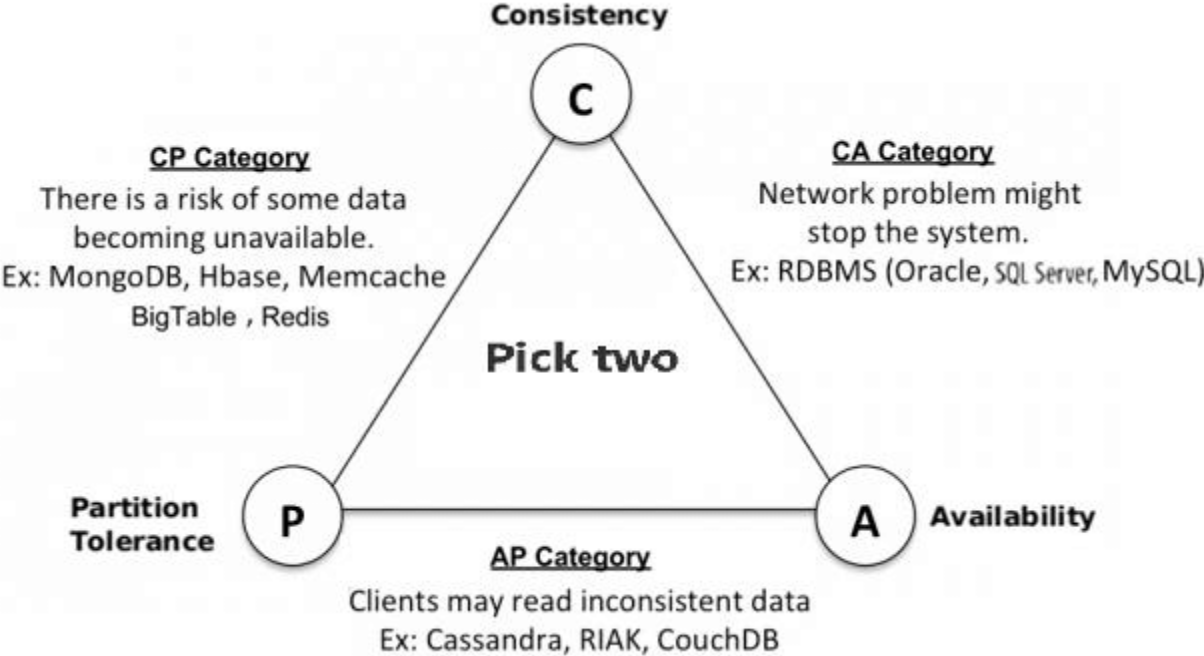
Availability approaches

- Consistency is a sliding scale, not an absolute. Many NoSQL databases allow tuning between levels of consistency and availability, which relates to the CAP theorem.
- CAP theorem shows the balance between consistency, availability, and partitioning can be maintained in a BASE database system.
- CAP stands for Consistency, Availability, and Partitioning, which are aspects of data management in databases.
- Here are some questions to consider when considering a BASE and thus CAP approach:
 - ✓ Consistency: Is the database fully (ACID) consistent, or eventually consistent, or without any consistency guarantees?
 - ✓ Availability: During a partition, is all data still available (that is, can a partitioned node still successfully respond to requests)?
 - ✓ Partitioning: If some parts of the same database cluster aren't communicating with each other, can the database still function separately and correct itself when communication is restored?

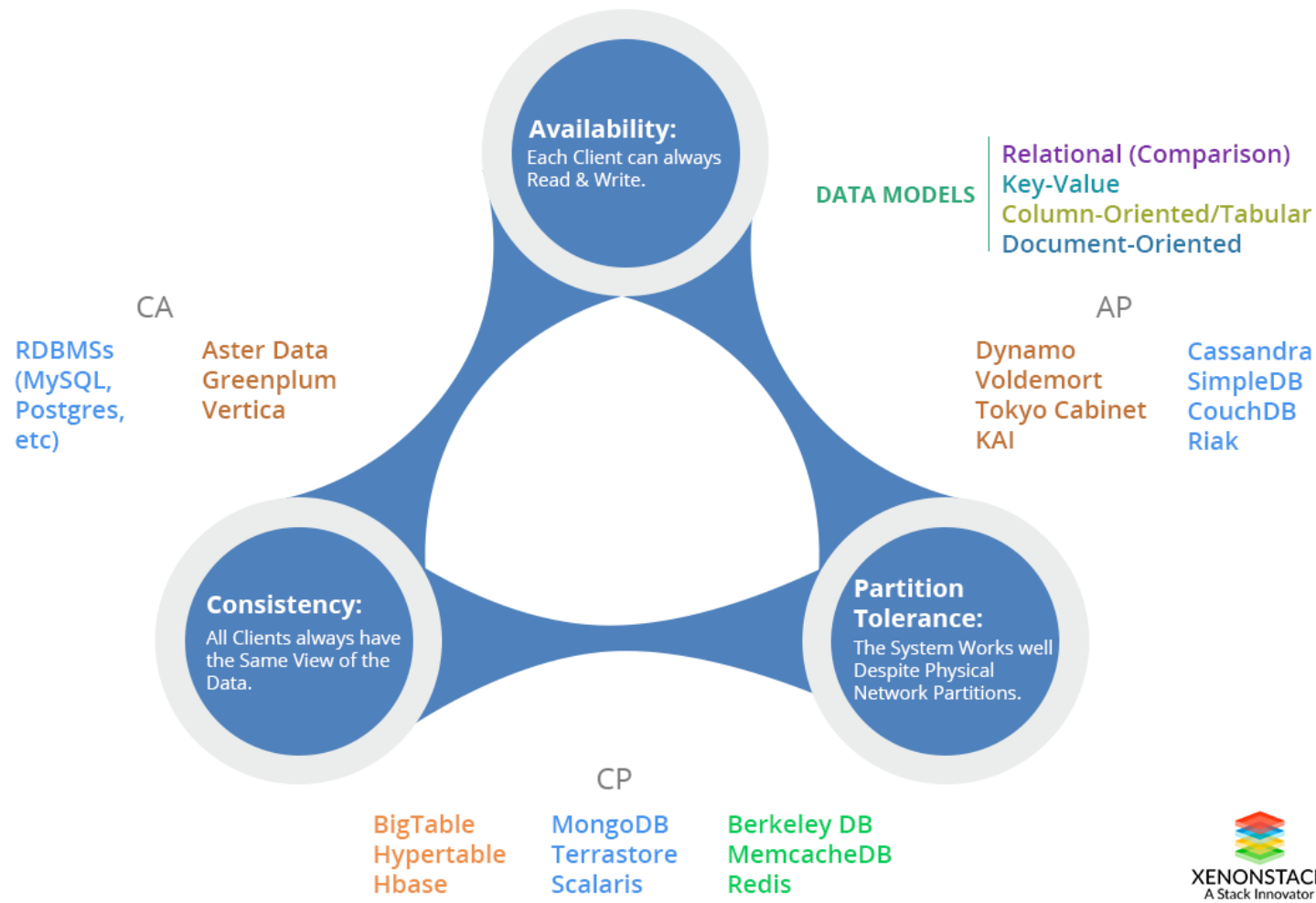
- The CAP theorem states that you cannot have all features of all three at the same time.
- Most of the time, this is claimed to mean that you can have only two of the three.
- The definition of consistency in ACID isn't the same definition as in CAP:
 - ✓ In ACID, it means that the database is always in a consistent state.
 - ✓ In CAP, it means that a single copy of the data has been updated.
- Therefore, in CAP, a system that supports BASE can be consistent.
- A particular NoSQL database generally provides either
 - ❖ CA (consistency and availability)
 - ❖ AP (availability and partition tolerance)

Module No.1

Scaling & Visualizing NoSQL

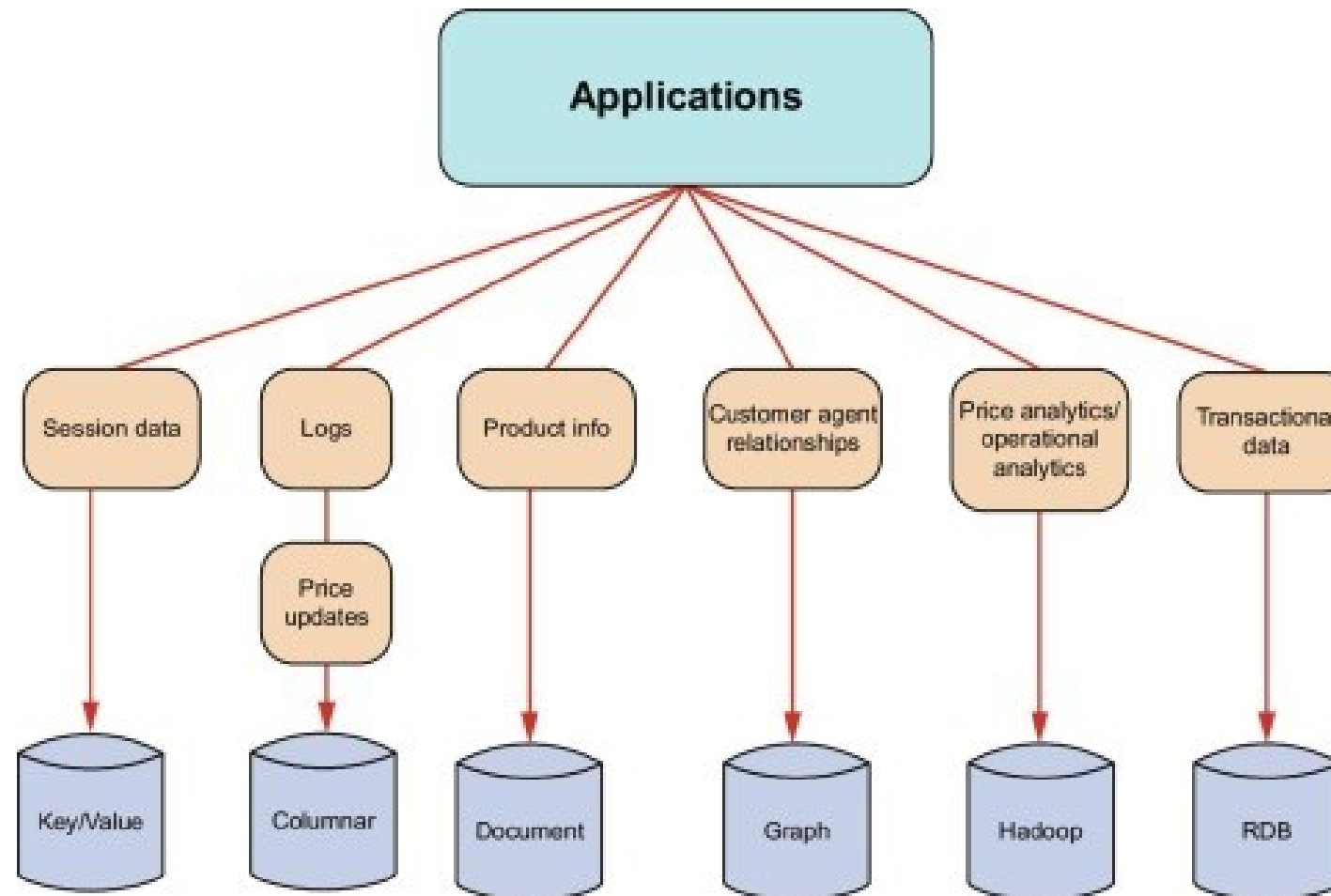


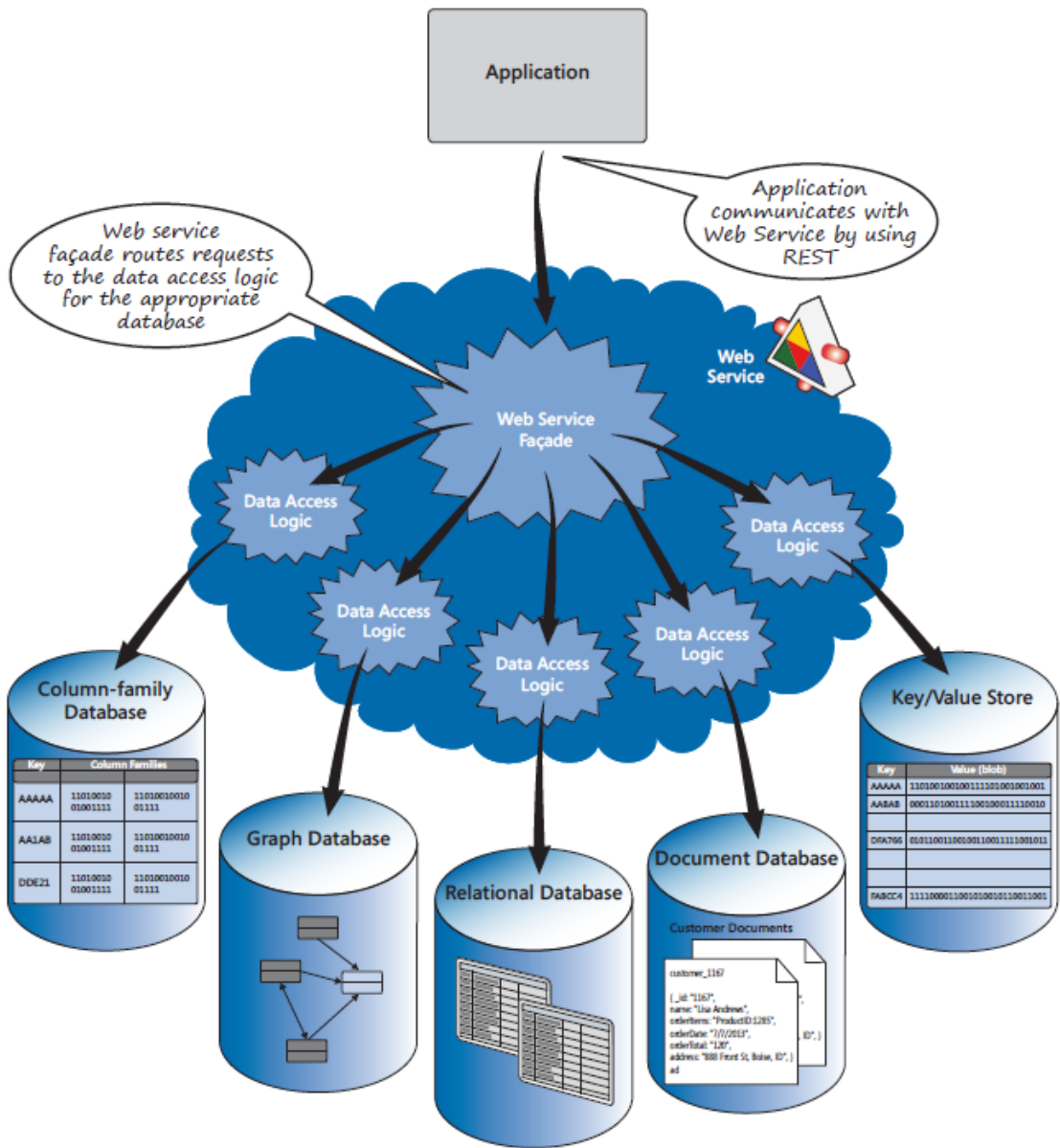
Visual Guide to NoSQL Systems



Polyglot persistence

- Polyglot persistence means that, in order to write a single complete application, the application's developer must use multiple types of databases, each for its most appropriate data type.
- Polyglot persistence is the idea that a single application that uses different types of data needs to use multiple databases behind that application.
- NoSQL databases are gradually crossing over and covering multiple data types.
- ✓ If you're considering a key-value store but some features are missing that handle specifics of the data, then consider a column store.
- ✓ If a column store can't handle a very complex structure in your application, then consider a document store.
- ✓ If you need to manage relationships or facts in your data, then you need features of a triple store, too.





Search engine techniques

- NoSQL databases are used to manage volumes of unstructured content.
- Be they long string fields, tweet content, XML medical notes, or plain text and PDF files. As a result, search functionality is important.
- People generally associate search engines only with full-text searches. However, there are many other uses for search engines.
- MarkLogic, for example, comes with a built-in search engine developed specifically for the data it was designed to store — documents.
- The database indexes are the same as those used for full-text search. MarkLogic includes a universal index.
- As well as indexing text content, it indexes exact field values, XML and JSON elements, and attribute and property names, and it maintains document ID and collection lexicons.
- Range indexes can be added to this mix after data is loaded and explored.
- Some NoSQL databases embed the common Apache Lucene engine to add full-text indexes for string fields.

Business Intelligence, dashboarding, and reporting

- It'd be nice to reuse stored data for strategic work as well as for operational workloads.
- Data warehouses hold the same information as an operational system but in a structure that's more useful for reporting tools to query.
- The problem with this approach is that the source system and the warehouse are never up to date.
- So, there's a need to perform business intelligence-style queries of data held in operational data stores, showing the current real-time state of the database.
- In NoSQL column stores, data is still held in tables, rows, and column families in a structure suited for the operational system, not a warehousing one.
- Logging databases are a good example. Cassandra has been used to store log files from systems as events occur.
- These live events are aggregated automatically to provide hourly, daily, weekly, and monthly statistics.

- Document NoSQL databases take a different approach.
- They store primary copies of data but allow data transformation on query and allow denormalizations to be computed on the fly.
- NoSQL databases can be used simultaneously because both the operational data store and for warehousing workloads.
- Business Intelligence (BI) reporting can be achieved by using a BI tool that understands the internal NoSQL databases structure.
- Tableau, for example, has native connectors to several NoSQL databases.
- Many people simply want a solid operational view of the current state of the world — in other words, dashboards.
- You can create dashboards by using aggregate functions over indexes of column or document stores.
- You can use search to restrict which data is aggregated .

- Having a NoSQL database with a rich REST (REpresentational State) API that you can rapidly plug into web widgets is advantageous when building out dashboarding apps.
- And it's even better if internal data structures (like search results, for example) are supported by a NoSQL vendors' JavaScript API.
- Using these API removes a lot of the plumbing code you will need to write to power a dashboard.

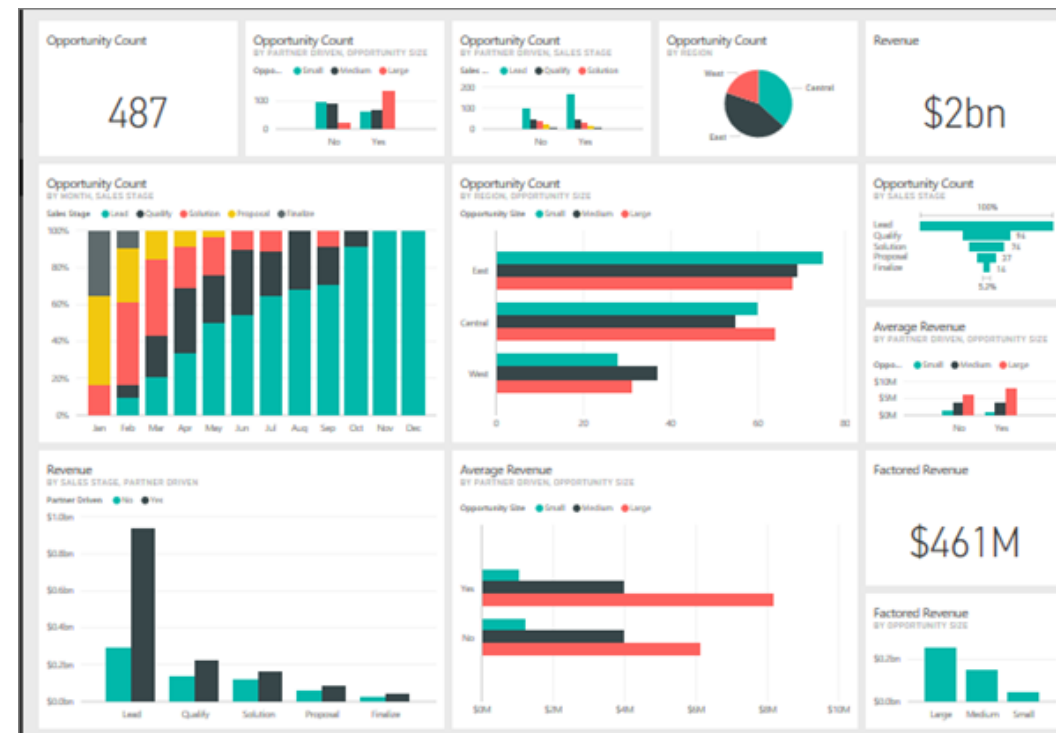


Table 3-1 NoSQL Data Management Use Cases

<i>Data to Manage</i>	<i>NoSQL Database</i>
Trade documents (FpML), Retail insurance policies (ACORD), healthcare messages, e-form data	Document database with XML support
Monthly data dumps in text delimited (CSV, TSV) files, or system/web log files	Bigtable clone for simple structures Document database for very complex structures
Office documents, emails, PowerPoint	Document database with binary document text and metadata extraction support
Web application persistent data (JavaScript Object Notation — JSON)	Document database with JSON support and a RESTful API
Metadata catalog of multiple other systems (for example, library systems)	Bigtable for simple list of related fields and values Document database for complex data structures or full text information
Uploaded images and documents for later retrieval by unique ID	Key-value store for simple store/retrieval Document store with binary text extraction and search for more complex requirements
RDF, N-Triples, N3, or other linked (open) data	Triple store to store and query facts (assertions) about subjects Graph store to query and analyze relationships between these subjects
Mix of data types in this table	Hybrid NoSQL database

Search features: Query versus search

- ✓ Any NoSQL database should be able to handle basic queries. Retrieving a record by an exact property, value, or ID match is the minimum functionality you need in a database. This is what key-value stores provide. These basic queries match exact values, such as
- By the record's unique ID in the database
 - By a metadata property associated with the record
 - By a field within the record

Search features: Query versus search

✓ Comparison queries, also commonly called *range queries*, find a stored value within a range of desired values. This can include dates, numbers, and even 2D geospatial coordinates, such as searching:

- By several matching fields, or fields within a range of values
- By whether a record contains a particular field at all (query on structure)

Comparison queries typically require a reverse index, where target values are stored in sequence, and record IDs are listed against them. This is called a Term List.

Search features: Query versus search

- ✓ Handling of free text, including more advanced handling such as language selection, stemming, and thesaurus queries, which are typically done by search engines. In the NoSQL world (especially document NoSQL databases), handling unstructured or poly-structured data is the norm, so this functionality is very desirable for such a use case, including support for searching:

- By a free text query
- By a stemmed free text query (both *cat* and *cats* stem to the word *cat*) or by a thesaurus
- By a complex query (for example, geospatial query) across multiple fields in a record
- By calculating a comparison with a query value and the value within a record's data (for example, calculated distance of five miles based on a point within a record, and the center of a City — Finding hotels in London.)

Search features: Query versus search

- ✓ In the world of analytics, you calculate summaries based on the data in matching records, and perhaps as compared to the search criteria. It's common to calculate relevancy based on distance from a point, instead of simply returning all records within a point and radius geospatial query. So, too, is returning a heat map of the data rather than all matching data. These tools are useful for complex searches such as the following:

- By calculating the above while returning a summary of all results (for example, heat map, faceted search navigation, co-occurrence of fields within records)
- By an arbitrarily complex set of AND / OR / NOT queries combining any of the previously mentioned query terms
- By including the above terms in a giant OR query, returning a higher relevancy calculation based on the number of matches and varying weights of query terms

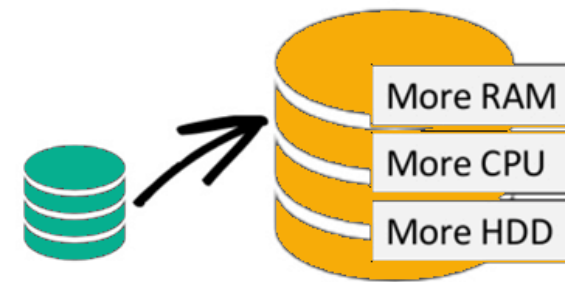
Search features: Query versus search

- ✓ Facetted search navigation where you show, for example, the total number of records with a value of Sales in the Department field is also useful. This might be shown as “Department — Sales (17)” in a link within a user interface. Faceting is particularly useful when your result list has 10,000 items and you need to provide a way to visually narrow the results, rather than force the user to page through 10,000 records.

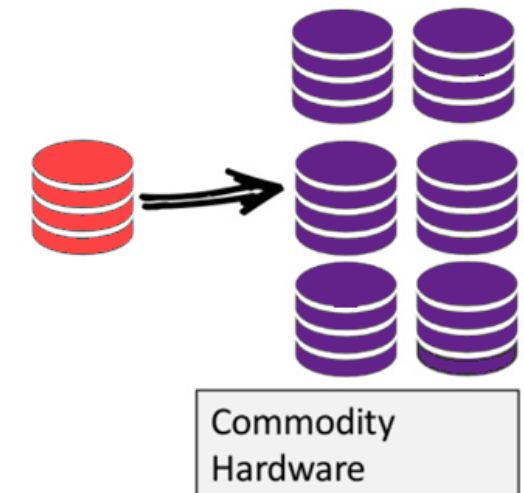
Scaling NoSQL

- NoSQL systems have the ability to scale across many commodity servers.
- There are high-volume use cases that will quickly force you to scale out. These include:
 - You receive status reports and log messages from across an IT landscape.
 - You want high-speed caching for complex queries.
- Many vendors quote high ingest speeds against an artificial use case that is not a realistic use of their database, as proof of their database's supremacy.
- However, the problem is that these same studies may totally ignore query speed. What's the point in storing data if you never use it?

Scale-Up (*vertical scaling*):



Scale-Out (*horizontal scaling*):



- Questions to ask to a NoSQL vendor:
 - Can you ensure that all sizing and performance figures quoted are for systems that ensure ACID transactions during ingest that support real-time indexing, and that include a realistic mix of ingest and read/query requests?
 - Does your product provide features that make it easy to increase a server's capacity?
 - Does your product provide features that make it easy to remove unused server capacity?
 - Is your product's data query speed limited by the amount of information that has to be cached in RAM?
 - Does your product use a memory map strategy that requires all indexes to be held in RAM for adequate performance (memory mapped means the maximum amount of data stored is the same as the amount of physical RAM installed)?

- Can your database maintain sub-second query response times while receiving high-frequency updates?
- Does the system ensure that no downtime is required to add or remove server capacity?
- Does the system ensure that information is immediately available for query after it is added to the database?
- Does the system ensure that security of data is maintained without adversely affecting query speed?
- Does the system ensure that the database's scale-out and scale-back capabilities are scriptable and that they will integrate to your chosen server provisioning software (for example, VMWare and Amazon Cloud Formation)?

Keeping data safe

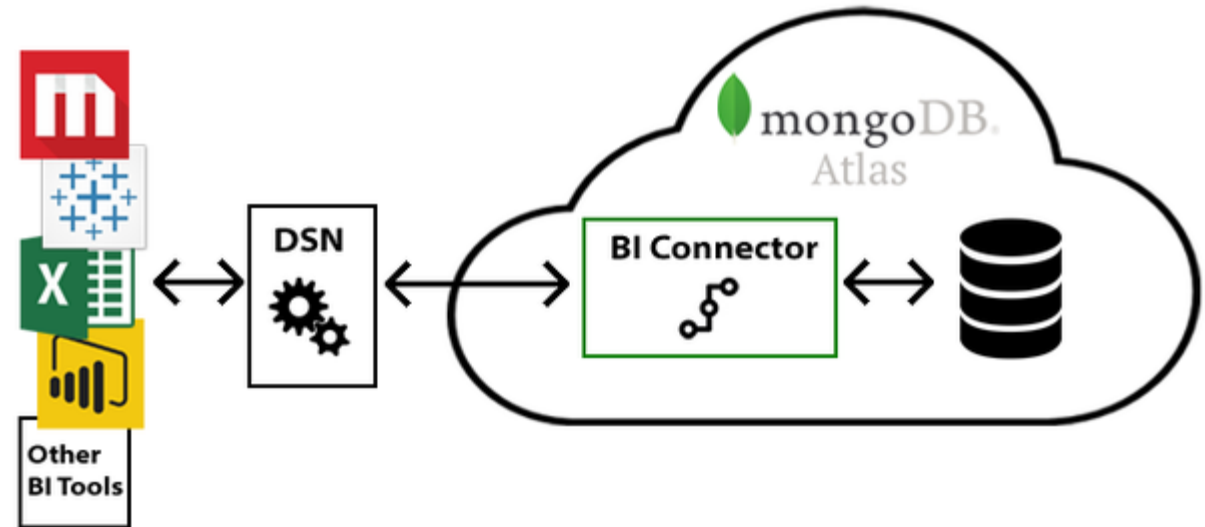
- Here are some thoughts that can help you discover the data safety features in NoSQL databases:
 - The vendor should ensure that all sizing and performance figures quoted are for systems that ensure strongly consistent (ACID) transactions during ingest, real time indexing, and a real-life mix between ingest and read/query requests.
 - Vendor should provide information about cases in which the database is being used as the primary information store in a mission-critical system.
 - This should not include case studies where a different database held the primary copy, or backup copy, of the data being managed.
 - TSSE that, once the database confirms data is saved, it will be recoverable (not from backups) if the database server it's saved on fails in the next CPU clock cycle after the transaction is complete.
 - Does the database ensure data is kept safe (for example, using journal logs or something similar)?
 - Does the system support log shipping to an alternative DR site?

Keeping data safe

- TSSE that the DR site's database is kept up to date. How does your database ensure this? For example, is the DR site kept up to date synchronously or asynchronously? If asynchronously, what is the typical delay?
- TTSP audit trails so that both unauthorized access and system problems can be easily diagnosed and resolved.
- What level of transactional consistency does your database provide by default (for example, eventual consistency, check and set, repeatable read, fully serializable)?
- What other levels of transactional consistency can your database be configured to use (for example, eventual consistency, check and set, repeatable read, fully serializable)? Do they include only the vendor's officially supported configurations?
- What is the real cost of a mission-critical system?
 - Ask the vendor to denote which versions of its product fully support high availability, disaster recovery, strong transactional consistency, and backup and restore tools.
 - Ask the vendor to include the complete list price for each product version that achieves the preceding requirements

Visualizing NoSQL

- Storing and retrieving large amounts of data and doing so fast is great, and once you have your newly managed data in NoSQL, you can do great things.
 1. Entity extraction and enrichment
 2. Search and alerting
 3. Aggregate functions
 4. Charting and business intelligence



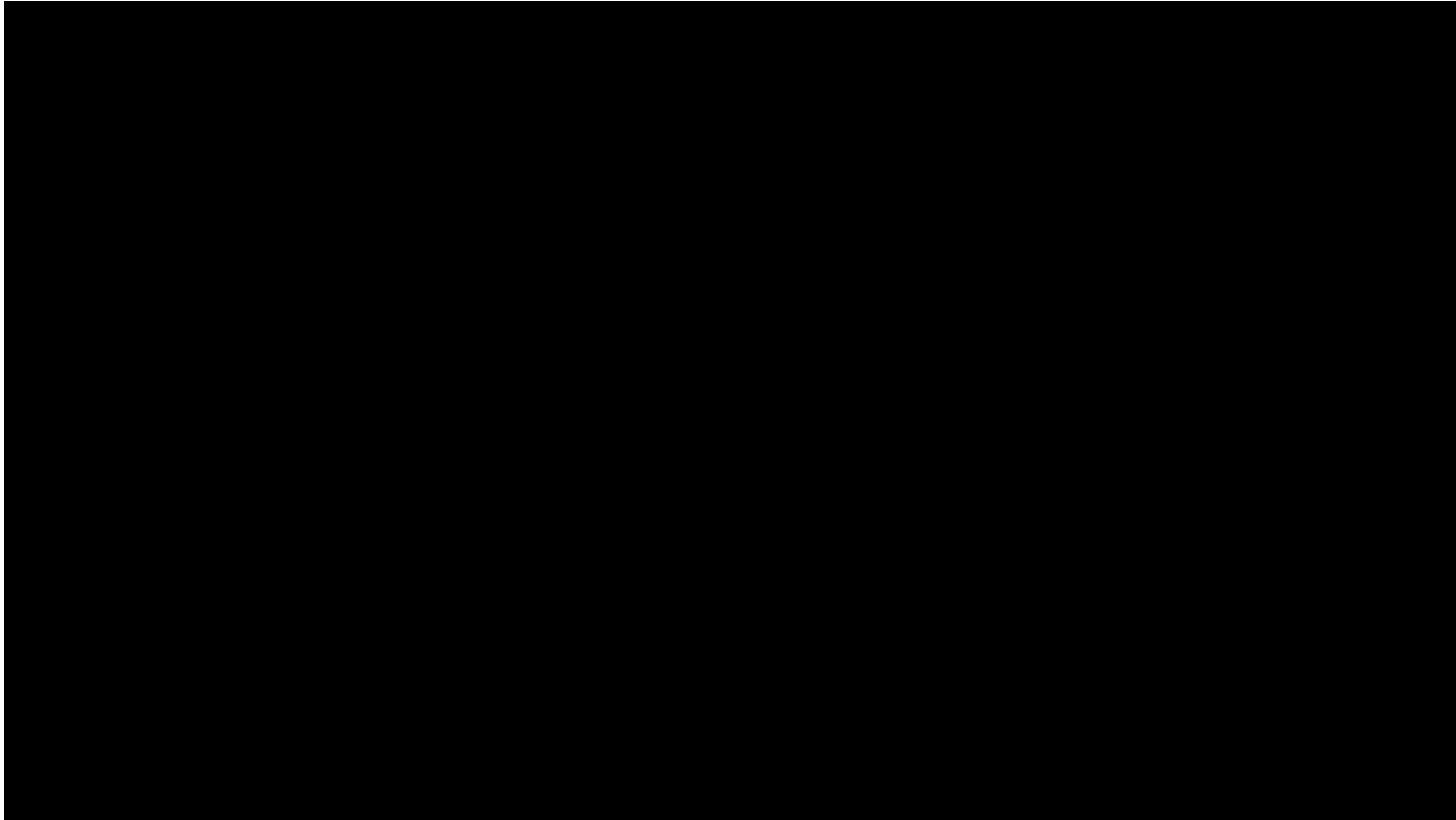


Extending your data layer

- Not ensuring that extended functionality is supported will mean you will end up installing several NoSQL databases at your organization.
- This means additional cost in terms of support, training and infrastructure.
- It's better to be sure you select a NoSQL database that can meet the scope of your goals, either through its own features or through a limited number of partner software products.
- Whether you select open-source or commercial software, be sure the developer documentation and training are first rate.
- These software extensions can be anything useful to your business, but typically they are on either the ingest side or the information analysis side of data management rather than purely about storage.
- Ingestion connectors to take information from Twitter, SharePoint, virtual file systems, and combine this data may be useful.

The Business Evaluation

- There are some of the areas of the non-technical, or business evaluation, you should consider when evaluating NoSQL databases.
 1. Developing skills
 2. Getting value quickly
 3. Finding help
 4. Deciding on open-source versus commercial software
 5. Building versus buying
 6. Evaluating vendor capabilities
 7. Finding support worldwide
 8. Expanding to the cloud



Deciding commercials

1. Business or mission-critical features
 2. Vendor claims
 3. Enterprise system issues
 4. Security
- If your organization's reputation or its financial situation will suffer if your system fails, then your system is, by definition, an enterprise class system.
 - Often, people confuse a large enterprise customer with a large enterprise system. Amazon, for example, is definitely a large-enterprise organization.
 - If a database is used to store customer orders and transactions, then it's a mission-critical enterprise system.
 - If a database is used behind an internal staff car sharing wiki page, then it's most definitely not a mission-critical application.

Preparing for failure

- When selecting a NoSQL database that needs to be resilient to individual hardware failures, watch for the following features.

High availability (HA)

- HA refers to the ability for a service to stay online if part of a system fails.
- In a NoSQL database, this typically means the ability for a database cluster to stay online, continuing to service all user requests if a single database servers within a cluster fail.
- HA requires either a shared storage system (like a NAS or a SAN) or stored replicas of the data.
- A Hadoop cluster, for example, stores all data locally but typically replicates data twice (resulting in three copies) so that, if the primary storage node fails, the data is still accessible.
- Some NoSQL databases that provide sharding don't replicate their data, or they replicate it just for read-only purposes.
- Therefore, losing a single node means some data can't be updated until the node is repaired.

Preparing for failure

Disaster recovery (DR)

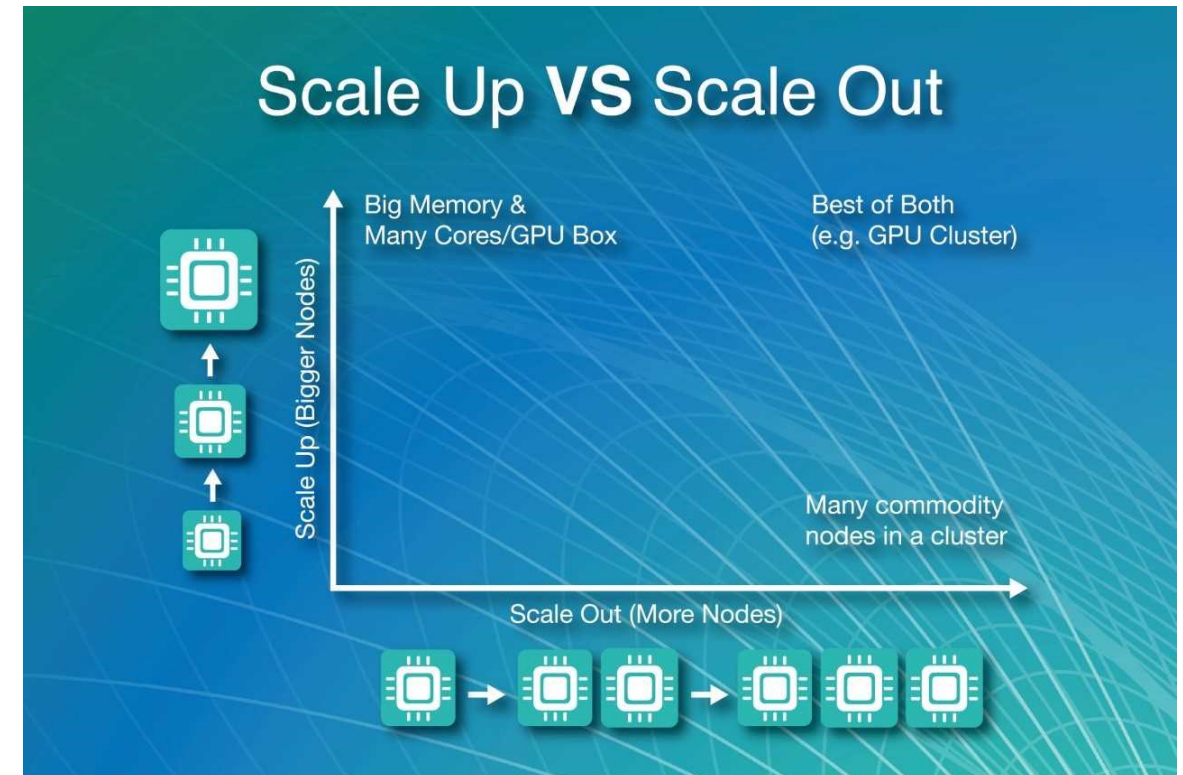
- DR is dramatically described as recovering from a nuke landing on your primary data center.
- No matter the cause, having a hot standby data center with up-to-date copies of your data ready to go within minutes is a must if your system is mission-critical.
- Typically, the second cluster is an exact replica of your primary data center cluster.
- There are several reasons databases fail: human error, network failure, hardware failure, and unanticipated load, to name a few.

Scaling up

- NoSQL databases were designed with considerable scalability in mind.
- **Query scalability**
 - Some NoSQL databases are designed to focus more on query scalability than data scalability.
 - Graph databases are good examples.
 - A very complex graph query like “Find me the sub graphs of this graph that most closely match these subjects” requires comparing links between many stored data entities (or subjects in graph speak).
 - Because the query needs data about the links between items, having this data stored across many nodes results in a lot of network traffic.
 - So, many graph databases store the entire graph on a single node.
 - This makes queries fast but means that the only multiserver scalabilities you get are multiple read-only copies of the data to assist multiple querying systems.
 - Conversely, a document database like MongoDB or MarkLogic may hold documents on a variety of servers (or database nodes).

- Because a query returns a set of documents, and each document exists only on a single node, it's easy to pass a query to each of the 20 database nodes and correlate the results together afterward with minimum networking communication.
- Each document is self-contained and evaluated against the query by only the database node it's stored on.
- This is the same MapReduce distributed query pattern used by Hadoop MapReduce.
- Storing your information at the right level means that the queries can be evaluated at speed.
- **Cluster scalability**
 - Some NoSQL databases have scale-out and scale-back support.
 - This is particularly useful for software as a service (SaaS) solutions on a public cloud like Amazon or Microsoft Azure.
 - Scale out is the ability to start up a new database instance and join it to a cluster automatically when a certain system metric is reached.
 - An example might be CPU usage on query nodes going and staying above 80 percent for ten minutes.
 - Cluster horizontal scaling support should include automated features and integration to cloud management software like AWS.

- Support for auto-rebalancing data transparently while the system is in use solves this problem rapidly, and without administrator intervention.
- Auto-rebalancing can be reliably implemented only on NoSQL databases with ACID compliance.
- Scale-back is a complex feature to implement and is still very rare.
- At the time of this writing, only MarkLogic Server can perform automatic scale-back on Amazon.
- MarkLogic Server also has an API that you can use to plug in scale-out/scale-back functionality with other public and private cloud management software.

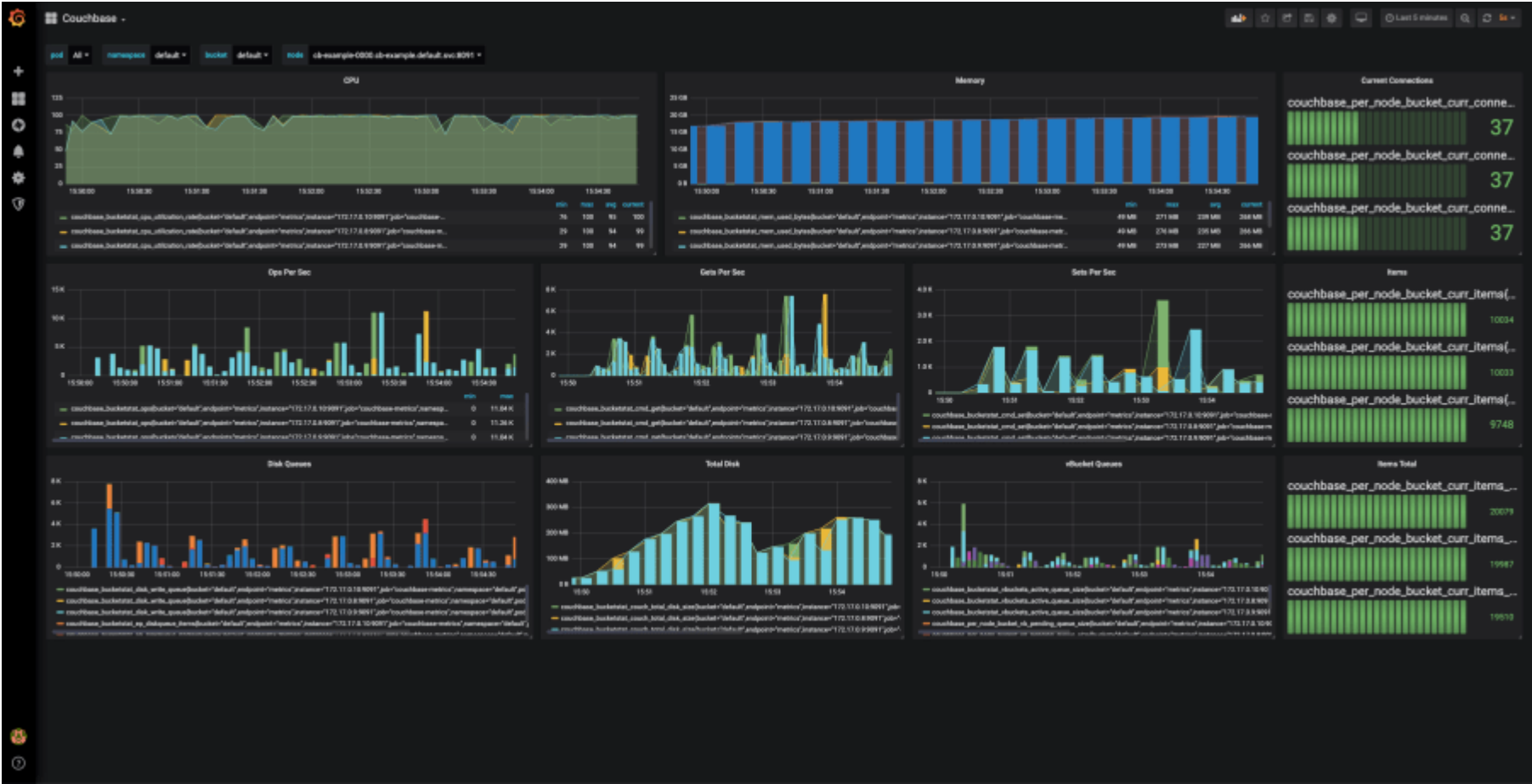


Acceptance testing

- User-acceptance tests ensure that the database is aligned with the business objectives.
- They are conducted by the customers of the database, usually people from the business who would use the system, developers of applications or downstream reporting systems, as well as training staff.
- Operations will also check that the database has met all the requirements for maintainability.
- Many (update) issues can often be avoided through a significant investment in testing, particularly User Acceptance Testing (UAT), before going live.
- Even something that you may think is a minor update can irritate and alienate customers.

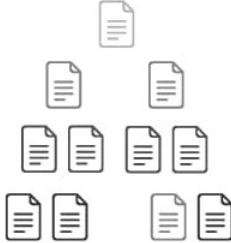
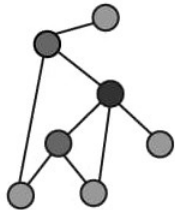
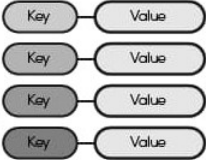
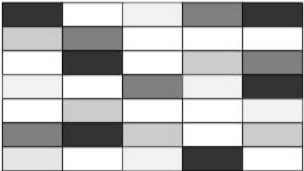
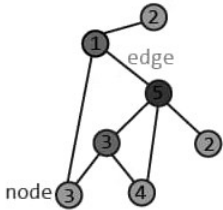
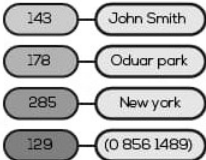
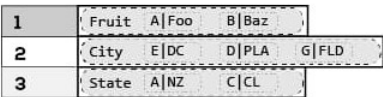




Monitoring

- NoSQL Database Monitoring helps you to monitor the status and performance of databases and provide extensive information real-time so that necessary steps can be taken before a breakdown of vital business processes occurs.
 - Database Monitoring helps you monitor database for performance and robustness.
 - Monitoring comes in two broad forms:
 - ✓ Systems monitoring watches components such as databases, storage, and network connectivity. This is the first form that you can enable without any database- or application-specific work.
 - ✓ Application monitoring spots potential performance issues before they bring a system down.
 - Ideally, you want at least a way to determine
 - ✓ What queries or internal processes are taking the longest to complete
 - ✓ What application or user asked for these queries or processes to be executed
- Also ideally, a monitoring dashboard that allows you to tunnel down into particular application queries on particular database nodes is helpful.



Data Architecture Pattern

- A data architecture pattern is a consistent way of representing data in a regular structure that will be stored in memory.
- Although the memory you store data in is usually long-term persistent memory, such as solid state disk or hard drives, these structures can also be stored in RAM and then transferred to persistent memory by another process.

Document	Graph	Key-Value	Wide-Column
			
<pre>{ "user": { "id": "143", "name": "improgrammer", "city": "New York" } }</pre>			
			

NoSQL data architecture patterns

Pattern name	Description	Typical uses
Key-value store	A simple way to associate a large data file with a simple text string	Dictionary, image store, document/file store, query cache, lookup tables
Graph store	A way to store nodes and arcs of a graph	Social network queries, friend-of-friends queries, inference, rules system, and pattern matching
Column family (Bigtable) store	A way to store sparse matrix data using a row and a column as the key	Web crawling, large sparsely populated tables, highly-adaptable systems, systems that have high variance
Document store	A way to store tree-structured hierarchical information in a single unit	Any data that has a natural container structure including office documents, sales orders, invoices, product descriptions, forms, and web pages; popular in publishing, document exchange, and document search

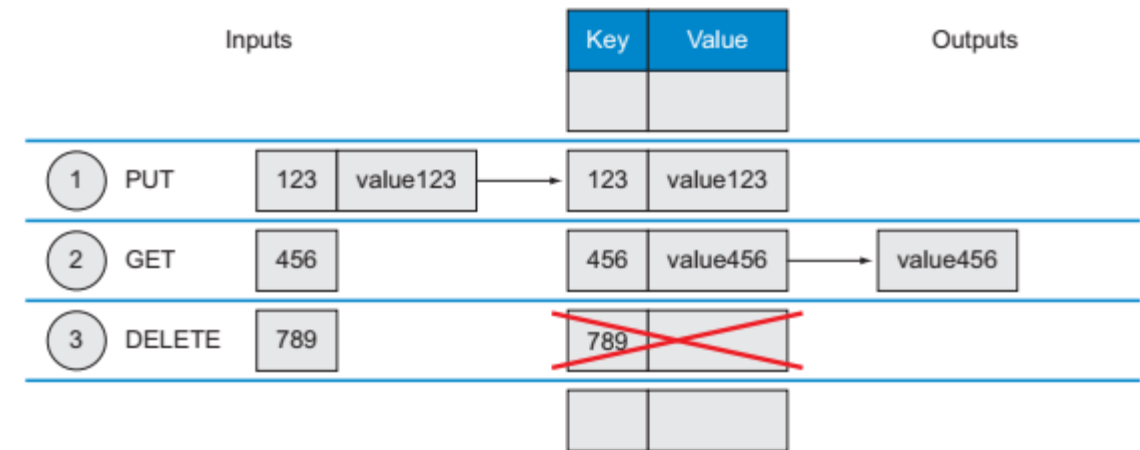
Key-value stores

- A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (the value).
- Key-value stores have no query language; they provide a way to add and remove key-value pairs into/from a database.
- A key-value store is like a dictionary. Like the dictionary, a key-value store is also indexed by the key; the key points directly to the value.
- Key-value stores have no query language.
- Benefits of using a key-value store: their simplicity and generality save you time and money by moving your focus from architectural design to reducing your data services costs through:
 - ☐ Precision service levels
 - ☐ Precision service monitoring and notification
 - ☐ Scalability and reliability
 - ☐ Portability and lower operational costs

	Key	Value
Image name →	image-12345.jpg	Binary image file
Web page URL →	http://www.example.com/my-web-page.html	HTML of a web page
File path name →	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash →	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call →	view-person?person-id=12345&format=xml	<Person><id>12345</id>.</Person>
SQL query →	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id>.</Person>

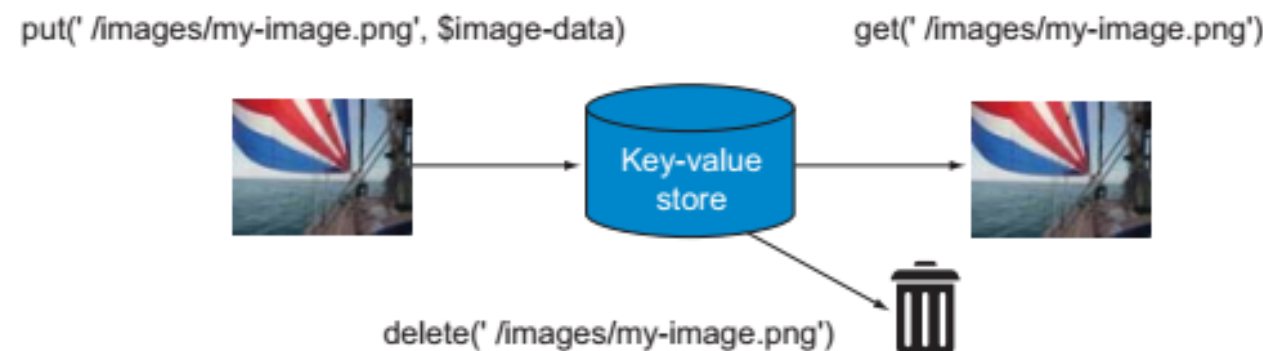
Using a key-value store

- The best way to think about using a key-value store is to visualize a single table with two columns.
- The first column is called the key and the second column is called the value.
- There are three operations performed on a key-value store:
 - put
 - get
 - delete
- These three operations form the basis of how programmers interface with the key-value store.
- We call this set of programmer interfaces the application program interface or API.



Using a key-value store

- Instead of using a query language, application developers access and manipulate a key-value store with the put, get, and delete functions, as shown here:
1. `put($key as xs:string, $value as item())` adds a new key-value pair to the table and will update a value if this key is already present.
 2. `get($key as xs:string) as item()` returns the value for any given key, or it may return an error message if there's no key in the key-value store.
 3. `delete($key as xs:string)` removes a key and its value from the table, or it may return an error message if there's no key in the key-value store.



- In addition to the put, get, and delete API, a key-value store has two rules: distinct keys and no queries on values:
 1. Distinct keys—You can never have two rows with the same key-value. This means that all the keys in any given key-value store are unique.
 2. No queries on values—You can't perform queries on the values of the table.
- Restrictions of Keys and Values:
 - There are few restrictions about what you can use as a key as long as it's a reasonably short string of characters.
 - There are also few restrictions about what types of data you can put in the value of a key-value store.
 - As long as your storage system can hold it, you can store it in a key-value store, making this structure ideal for multimedia: images, sounds, and even full-length movies.

- **Use case: storing web pages in a key-value store**
- The URL is the key, and the value is the web page or resource located at that key.
- If all the web pages in only part of the web were stored in a single key-value store system, there might be billions or trillions of key-value pairs.
- But each key would be unique, like a URL to a web page is unique.
- The ability to use a URL as a key allows you to store all of the static or unchanging components of your website in a key-value store. This includes images, static HTML pages, CSS, and JavaScript code.

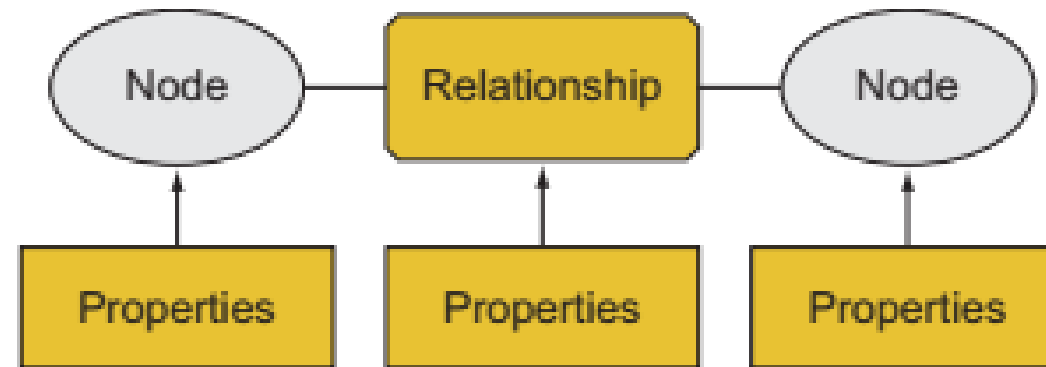
Key	Value
<code>http://www.example.com/index.html</code>	<code><html>...</code>
<code>http://www.example.com/about.html</code>	<code><html>...</code>
<code>http://www.example.com/products.html</code>	<code><html>...</code>
<code>http://www.example.com/logo.png</code>	Binary...

- **Use case: Amazon simple storage service (S3)**

- S3 is a simple key-value store with some enhanced features:
 - It allows an owner to attach metadata tags to an object, which provides additional information about the object; for example, content type, content length, cache control, and object expiration.
 - It has an access control module to allow a bucket/object owner to grant rights to individuals, groups, or everyone to perform put, get, and delete operations on an object, group of objects, or bucket.
- At the heart of S3 is the bucket.
- All objects you store in S3 will be in buckets.
- Buckets store key/object pairs, where the key is a string and the object is whatever type of data you have (like images, XML files, digital music).
- Keys are unique within a bucket, meaning no two objects will have the same key-value pair. S3 uses the same HTTP REST verbs (PUT, GET, and DELETE) discussed earlier in this section to manipulate objects:
 - New objects are added to a bucket using the HTTP PUT message.
 - Objects are retrieved from a bucket using the HTTP GET message.
 - Objects are removed from a bucket using the HTTP DELETE message.

Graph stores

- Applications that need to analyze relationships between objects or visit all nodes in a graph in a particular manner (graph traversal).
- Graph stores are highly optimized to efficiently store graph nodes and links, and allow you to query these graphs.
- A graph store is a system that contains a sequence of nodes and relationships that, when combined, create a graph.
- A graph store has three data fields: nodes, relationships, and properties.



- Graph nodes are usually representations of real-world objects like nouns.
- Nodes can be people, organizations, telephone numbers, web pages, computers on a network, or even biological cells in a living organism.
- Graph queries are similar to traversing nodes in a graph, things like these:
 - What's the shortest path between two nodes in a graph?
 - What nodes have neighbouring nodes that have specific properties?
 - Given any two nodes in a graph, how similar are their neighbouring nodes?
 - What's the average connectedness of various points on a graph with each other?
- Graph stores assign internal identifiers to nodes and use those identifiers to join networks together.
- Graph stores are difficult to scale out on multiple servers due to the close connectedness of each node in a graph.
- Similar to other types of databases, we load, query, update, and delete data. A graph query will return a set of nodes that are used to create a graph image on the screen to show you the relationship between your data.

A Graph Example

- You'll often see links on a page that take you to another page.
- These links can be represented by a graph or triple.
 - The current web page is the first or source node
 - The link is the arc that “points to” the second page
 - The second or destination page is the second node
- The W3C generalized this structure to store the information about the links between pages as well as the links between objects into a standard called Resource Description Format, more commonly known as RDF.

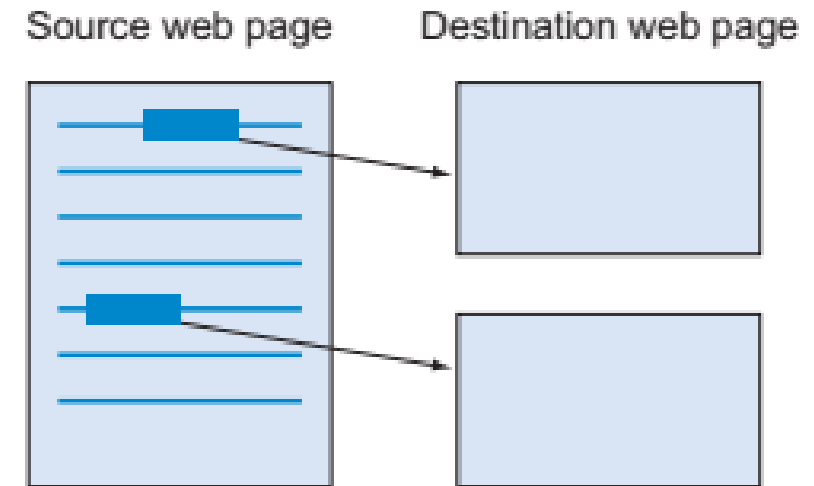
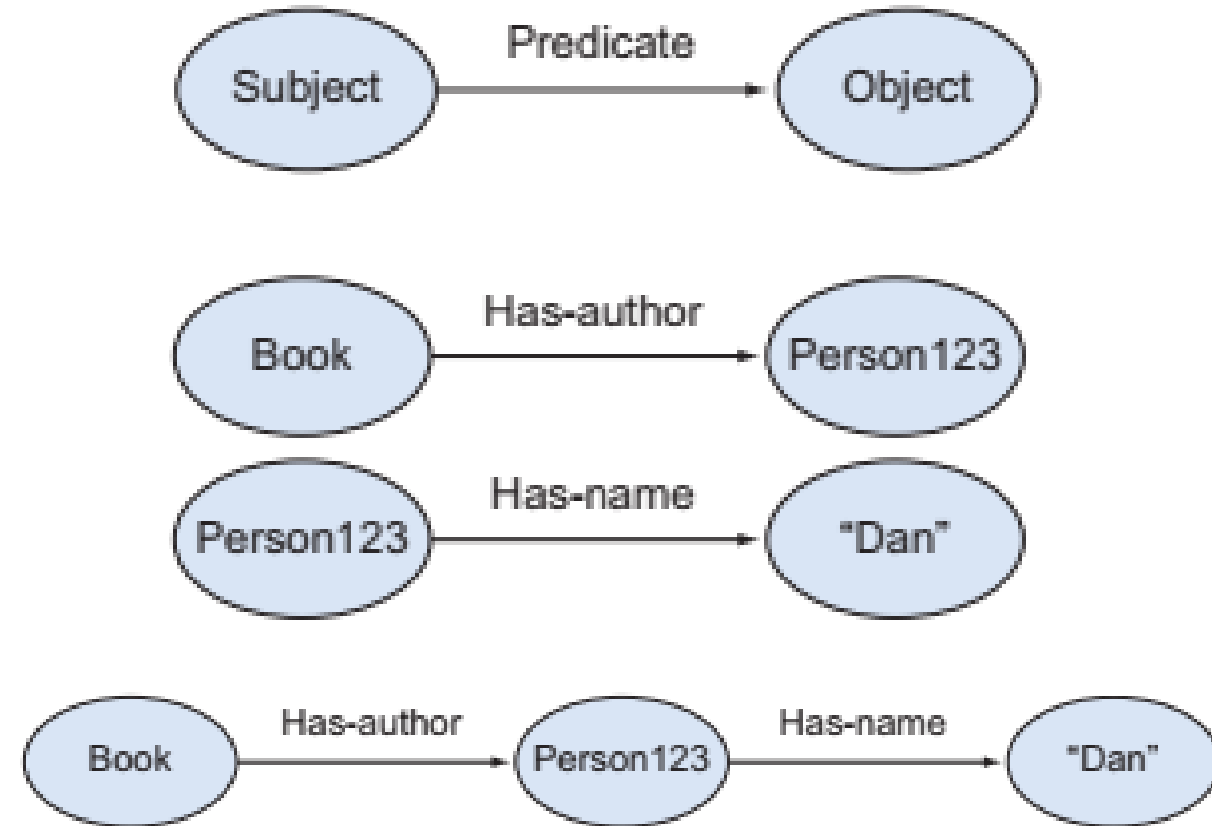


Figure 4.11 An example of using a graph store to represent a web page that contains links to two other web pages. The URL of the source web page is stored as a URL property and each link is a relationship that has a “points to” property. Each link is represented as another node with a property that contains the destination page’s URL.

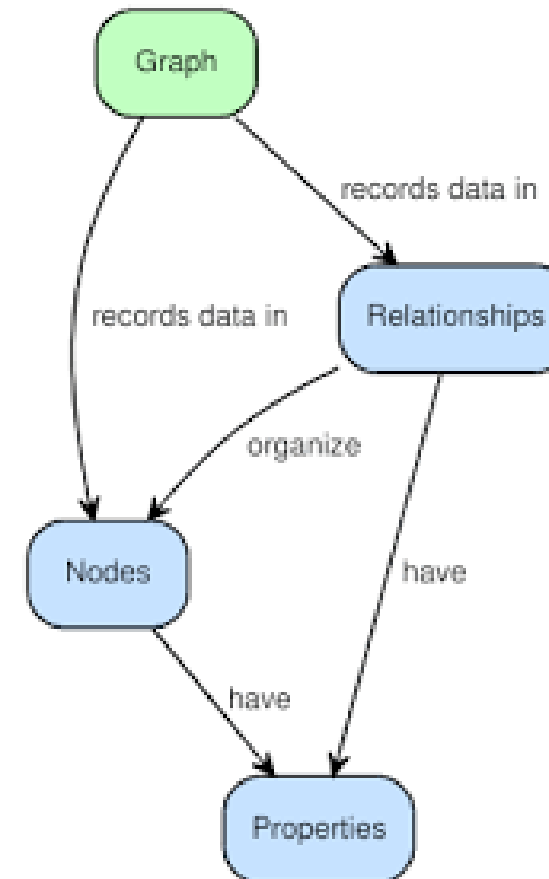
Linking external data with the RDF standard

- The W3C has focused on a process of using URL-like identifiers called uniform resource identifiers (URIs) to create explicit node identifiers for each node. This standard is called the W3C **Resource Description Format (RDF)**.
- RDF was specifically created to join together external datasets created by different organizations.
- Conceptually, you can load two external datasets into one graph store and then perform graph queries on this joined database.
- The terminology for the source, link, and destination may vary based on your situation, but in general the terms subject, predicate, and object are used.



Use cases for graph stores

- *Link analysis* is used when you want to perform searches and look for patterns and relationships in situations such as social networking, telephone, or email records.
- *Rules and inference* are used when you want to run queries on complex structures such as class libraries, taxonomies, and rule-based systems.
- *Integrating linked data* is used with large amounts of open linked data to do realtime integration and build mashups without storing data.



LINK ANALYSIS

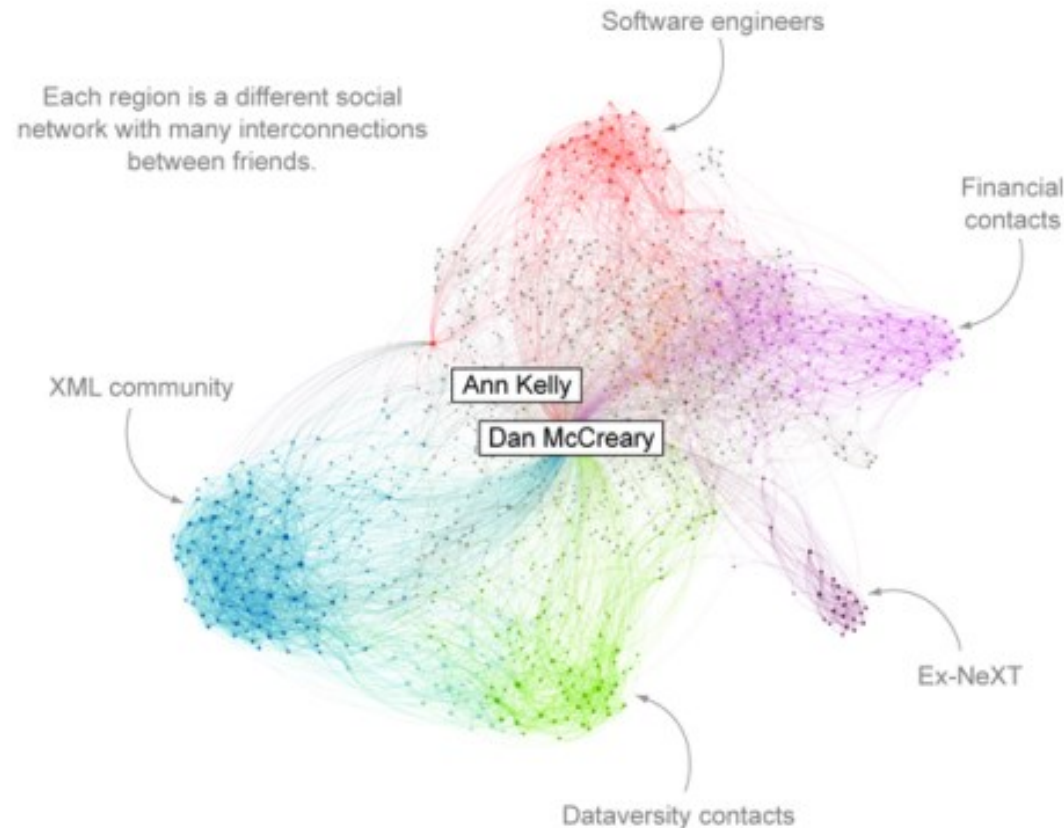


Figure 4.15 A social network graph generated by the LinkedIn InMap system. Each person is represented by a circle, and a line is drawn between two people that have a relationship. People are placed on the graph based on the number of connections they have with all the other people in the graph. People and relationships are shaded the same when there's a high degree of connectivity between the people. Calculating the placement of each person in a social network map is best performed by an in-memory graph traversal program.

- Graph stores are used for things beyond social networking—they're appropriate for identifying distinct patterns of connections between nodes.
- Creating a graph of all incoming and outgoing phone calls between people in a prison might show a concentration of calls (patterns) associated with organized crime.
- Analyzing the movement of funds between bank accounts might show patterns of money laundering or credit card fraud.
- Companies that are under criminal investigation might have all of their email messages analyzed using graph software to see who sent who what information and when.
- Entity extraction is the process of identifying the most important items (entities) in a document. Entities are usually the nouns in a document like people, dates, places, and products.
- Once the key entities have been identified, they're used to perform advanced search functions..

GRAPHS, RULES, AND INFERENCE

- We use the term to define abstract rules that relate to an understanding of objects in a system, and how the object properties allow you to gain insight into and better use large datasets.
- RDF was designed to be a standard way to represent many types of problems in the structure of a graph. A primary use for RDF is to store logic and rules.
- Suppose you have a website that allows anyone to post restaurant reviews.
- Would there be value in allowing you to indicate which reviewers you trust?
- You're going out to dinner and you're considering two restaurants. Each restaurant has positive and negative reviews.
- Can you use simple inference to help you decide which restaurant to visit?
- You could see if your friends reviewed the restaurants. But a more powerful test would be to see if any of your friends-of-friends also reviewed the restaurants.
- If you trust John and John trusts Sue, what can you infer about your ability to trust Sue's restaurant recommendations?

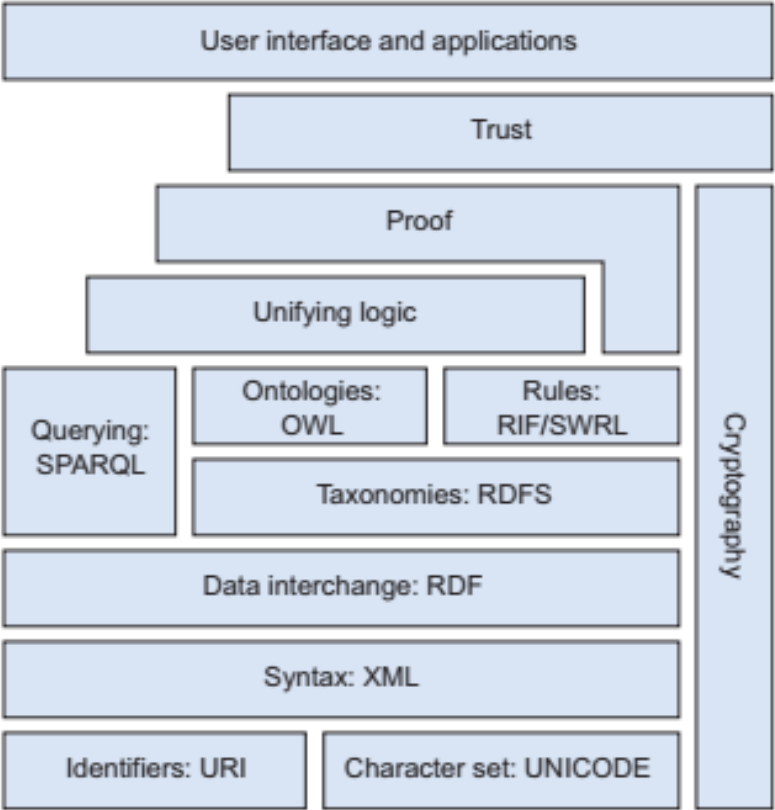


Figure 4.16 A typical semantic web stack with common low-level standards like URI, XML, and RDF at the bottom of the stack. The middle layer includes standards for querying (SPARQL) and standards for rules (RIF/SWRL). At the top of the stack are user interface and application layers above abstract layers of logic, proof, and trust building.

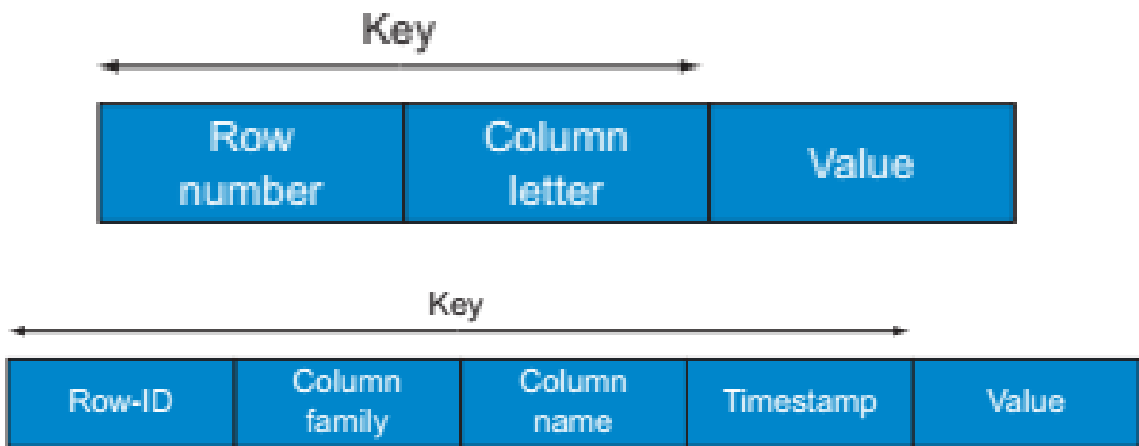
Integrating linked data

- Graph stores are also useful for doing analysis on data that hasn't been created by your organization.
- What if you need to do analysis with three different datasets that were created by three different organizations?
- Using a set of tools referred to as linked open data or LOD. it is an integration technique for doing joins between disparate datasets to create new applications and new insights.
- LOD strategies are important for anyone doing research or analysis using publicly available datasets.
- This research includes topics such as customer targeting, trend analysis, sentiment analysis.
- LOD integration creates new datasets by combining information from two or more publicly available datasets that conform to the LOD structures such as RDF and URIs
- The number of datasets that participate in the LOD community is large and growing, but as you might guess, there are few ways to guarantee the quality and consistency of public data.

Column family (Bigtable) stores

- One of the strengths of Column families is their ability of scaling to manage large volumes of data.
- They're also known to be closely tied with many MapReduce systems.
- Column family stores use row and column identifiers as general purposes keys for data lookup.
- They're sometimes referred to as data stores rather than databases.
- They lack typed columns, secondary indexes, triggers, and query languages
- Higher Scalability, Higher Availability, Ease to Add New Data

	A	B	C
1			
2			
3		Hello World!	
4			
5			
6			



Case study: storing analytical information in Bigtable

- The Bigtable is used to store website usage information in Google Analytics.
- The Google Analytics service allows you to track who's visiting your website. Every time a user clicks on a web page, the hit is stored in a single row-column entry that has the URL and a timestamp as the row ID.
- The row IDs are constructed so that all page hits for a specific user session are together.
- Viewing a detailed log of all the individual hits on your site would be a long process.
- Google Analytics makes it simple by summarizing the data at regular intervals (such as once a day) and creating reports that allow you to see the total number of visits and most popular pages that were requested on any given day.
- As each transaction occurs, new hit data is immediately added to the tables.
- The data in Google Analytics, like other logging-type applications, is generally written once and never updated

Case study: Google Maps stores geographic information in Bigtable

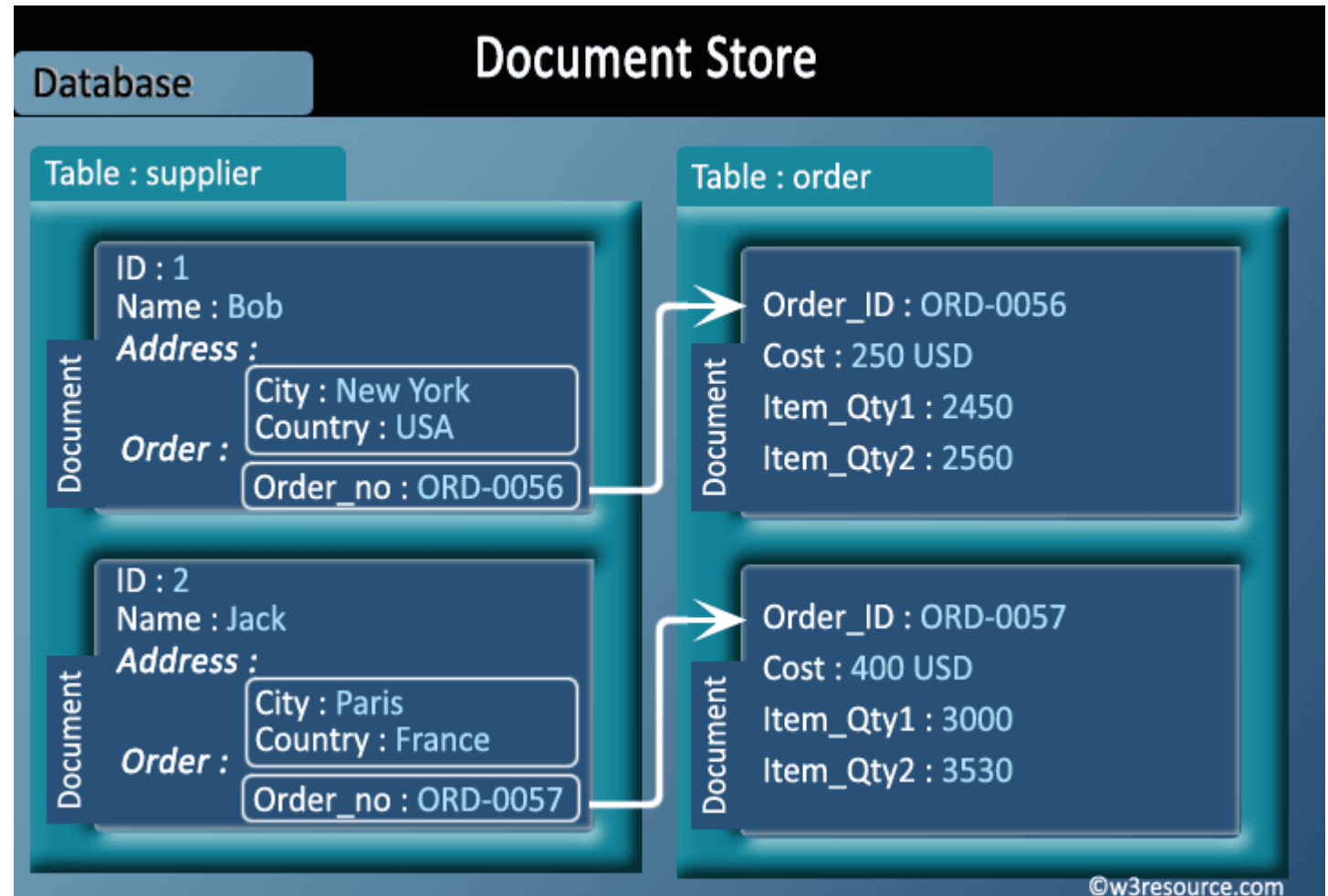
- Geographic information systems (GIS), like Google Maps, store geographic points on Earth by identifying each location using its longitude and latitude coordinates.
- The system allows users to travel around the globe and zoom into and out of places using a 3D-like graphical interface.
- When viewing the satellite maps, you can then choose to display the map layers or points of interest within a specific region of a map.
- GIS systems store items once and then provide multiple access paths (queries) to let you view the data.

Case study: using a column family to store user preferences

- Many websites allow users to store preference information as part of their profile
- The unique feature of user preference files is that they have minimal transactional requirements, and only the individual associated with the account makes changes (ensuring an ACID transaction isn't very important).
- It's important that these read-mostly events are fast and scalable so that when a user logs in, you can access the preferences and customize their screen regardless of the number of concurrent system users.
- Column family systems can provide the ideal match for storing user preferences when combined with an external reporting system.

Document stores

- The key-value store and Bigtable values lack a formal structure and aren't indexed or searchable.
- Everything inside a document is automatically indexed when a new document is added.
- Document stores can tell not only that your search item is in the document, but also the search item's exact location by using the document path, a type of key, to access the leaf values of a tree structure.



Document Store Basics

- Think of a document store as a tree-like structure
- Beneath the root element there is a sequence of branches
- Each branch has a related path expression that shows you how to navigate from the root of the tree to any given branch, sub-branch, or value.
- Each branch may have a value associated with that branch.

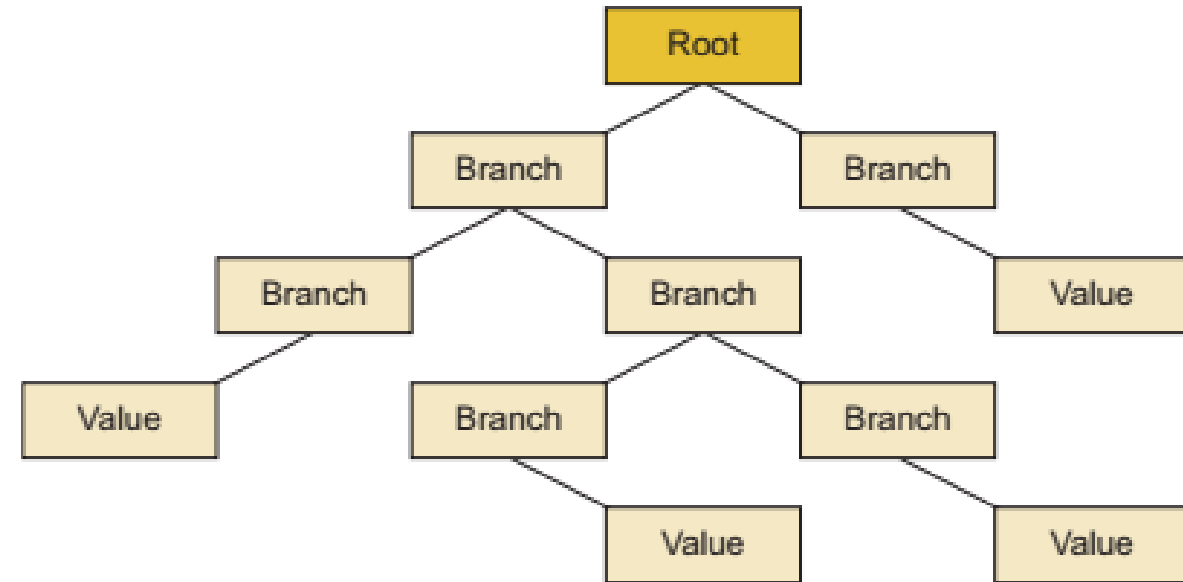


Figure 4.21 Document stores use a tree structure that begins with a root node, and have sub-branches that can also contain sub-branches. The actual data values are usually stored at the leaf levels of a tree.

Document collections

- Most document stores group documents together in collections that look like a directory structure of a filesystem.
- Document collections can be used in many ways to manage large document stores.
- They can serve as ways to navigate document hierarchies, logically group similar documents, and store business rules such as permissions, indexes, and triggers.
- Collections can contain other collections and trees can contain subtrees.



Application collections

- The collection in a document store is used as a container for a web application package.
- This packaging format, called a xar file, is similar to a Java JAR file or a WAR file on Java web servers.
- Packaged applications can contain scripts as well as data.
- These packaging features make document stores more versatile, expanding their functionality to become application servers as well as document stores.
- The use of collection structures to store application packages shows that a document store can be used as a container of high-level reusable components that can run on multiple NoSQL systems.



Figure 4.22 Document store collections can contain many objects, including other collections and application packages. This is an example of a package repository that's used to load application packages into the eXist native XML database.

Document store APIs

- Each document store has an API or query language that specifies the path or path expression to any node or group of nodes.
- Generally, nodes don't need to have distinct names; instead, a position number can be used to specify any given node in the tree.
- For example, to select the seventh person in a list of people, you might specify this query: `Person[7]`.

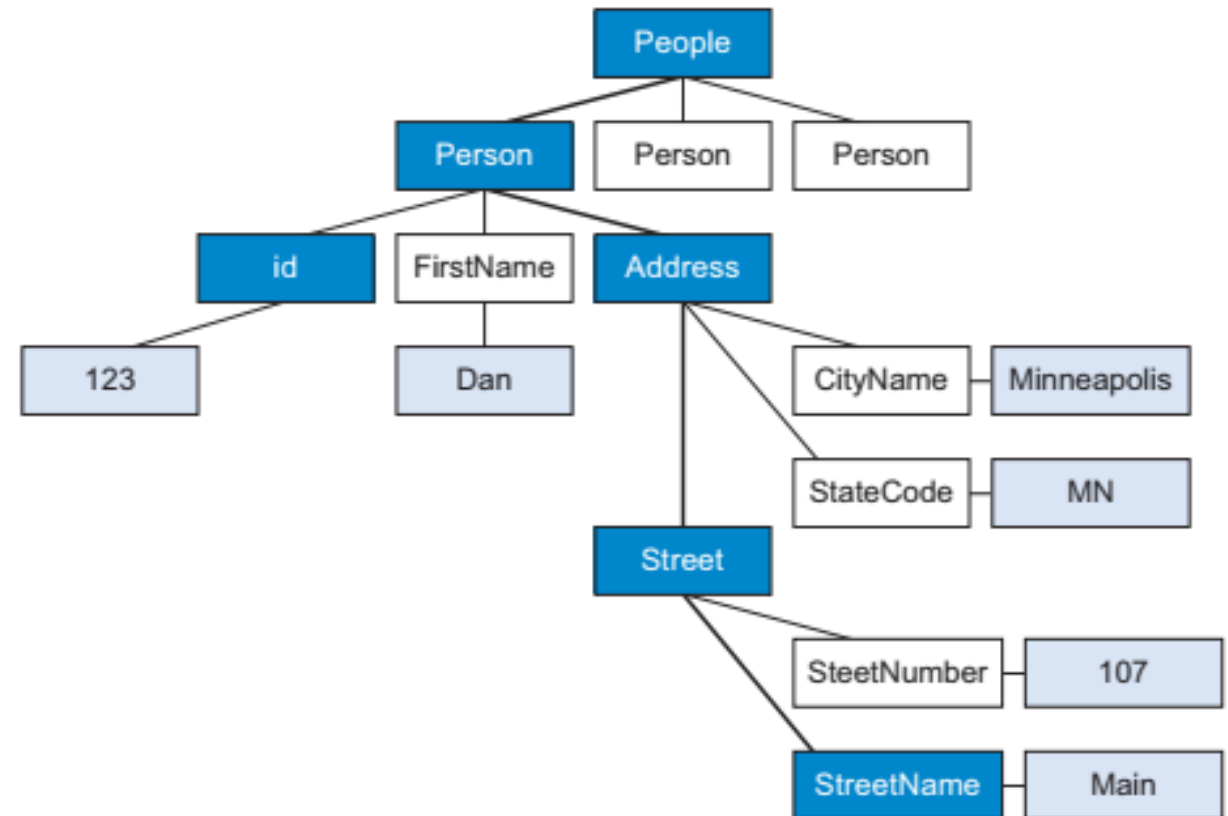


Figure 4.23 How a document path is used like a key to get the value out of a specific cell in a document. In this example, the path to the street name is `People/Person[id='123']/Address/Street/StreetName/text()`.

Document store implementations

- A document store can come in many varieties.
- Simpler document structures are often associated with serialized objects and may use the JavaScript Object Notation (JSON) format.












Key	Document
1001	<pre>{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }</pre>
1002	<pre>{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }</pre>

Case study: ad server with MongoDB

- Do you ever wonder how those banner ads show up on the web pages you browse or how they really seem to target the things you like or are interested in?
- It's done with ad serving. The original reason for MongoDB, a popular NoSQL product, was to create a service that would quickly send a banner ad to an area on a web page for millions of users at the same time.
- The primary purpose behind ad serving is to quickly select the most appropriate ad for a particular user and place it on the page in the time it takes a web page to load.
- Ad servers should be highly available and run 24/7 with no downtime. They use complex business rules to find the most appropriate ad to send to a web page.
- Ads are selected from a database of ad promotions of paid advertisers that best match the person's interest.
- There are millions of potential ads that could be matched to any one user. Ad servers can't send the same ad repeatedly; they must be able to send ads of a specific type (page area, animation, and so on) in a specific order. Finally, ad systems need accurate reporting that shows what ads were sent to which user and which ads the user found interesting enough to click on.

- MongoDB can be used in some of the following use cases:
 1. *Content management*—Store web content and photos and use tools such as geolocation indexes to find items.
 2. *Real-time operational intelligence*—Ad targeting, real-time sentiment analysis, customized customer-facing dashboards, and social media monitoring.
 3. *Product data management*—Store and query complex and highly variable product data.
 4. *User data management*—Store and query user-specific data on highly scalable web applications. Used by video games and social network applications.
 5. *High-volume data feeds*—Store large amounts of real-time data into a central database for analysis characterized by asynchronous writes to RAM..

MongoDB Use Cases!

Big Data	Product & Asset Catalogs	Security & Fraud	Internet of Things	Database-as-a-Service
  	  	  	   	  
Mobile Apps	Customer Data Management	Single View	Social & Collaboration	Content Management
   	   	    	   	   

Rich Query Model

Rich Queries	<ul style="list-style-type: none">Find Paul's carFind everybody in London with a car built between 1970 and 1980
Geo	<ul style="list-style-type: none">Find all car owners within 10m of Manhattan
Text Search	<ul style="list-style-type: none">Find all cars whose VIN starts with ZA1
Aggregation	<ul style="list-style-type: none">Calculate the average value of Paul's car collection
Map Reduce	<ul style="list-style-type: none">What is the ownership pattern of colors by geography over time? (Is silver trending up in the US?)

```
{  
  first_name: 'Paul',  
  surname: 'Miller',  
  cell: '+447557505611',  
  location: [45.123,47.232],  
  profession: [banking, finance,  
trader],  
  cars: [  
    { model: 'Bentley',  
      year: 1973,  
      value: 100000, ... },  
    { model: 'Rolls Royce',  
      year: 1965,  
      value: 330000, ... }  
  ]  
}
```

ADVANTAGES

LIMITATIONS

