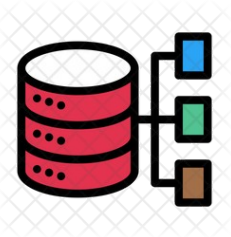
NoSQL Database Concepts

Tushar B. Kute,
http://tusharkute.com



NoSQL

- A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- Such databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 21st century, triggered by the needs of Web 2.0 companies.
- NoSQL databases are increasingly used in big data and real-time web applications.
- NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures

NoSQL

- Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), finer control over availability and limiting the object-relational impedance mismatch.
- The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL.
- The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

NoSQL – Types

- Wide column: Azure Cosmos DB, Accumulo, Cassandra, Scylla, HBase.
- Document: Azure Cosmos DB, Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, eXist-db, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
- Key–value: Azure Cosmos DB, Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper
- Graph: Azure Cosmos DB, AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso

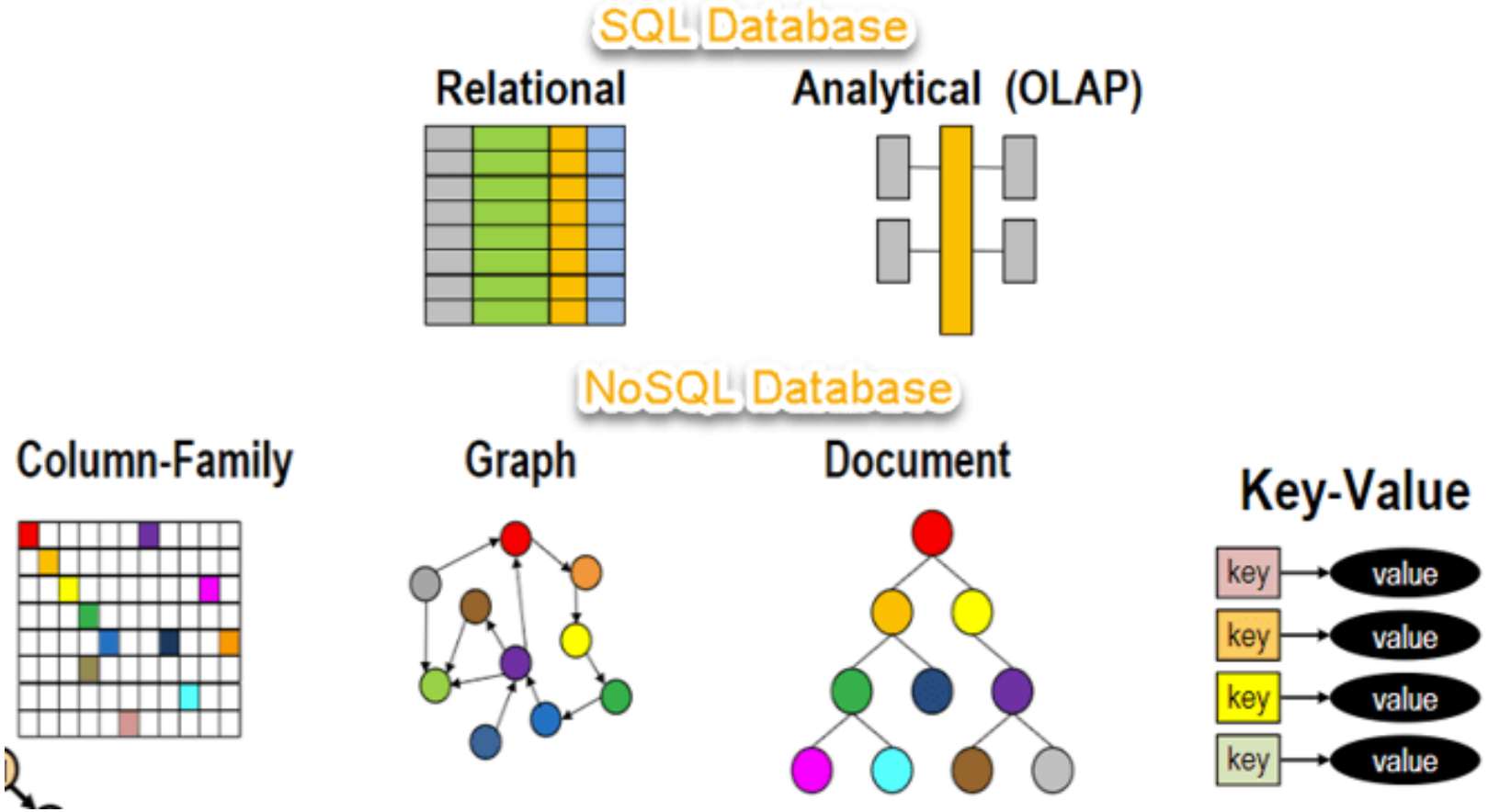
Database as a Service – SQL

- Amazon Aurora, MySQL based service
- Amazon Relational Database Service
- Clustrix Database as a Service
- Crunchy Bridge, PostgreSQL as a Service.
- EnterpriseDB Postgres Plus Cloud Database
- Google Cloud SQL
- Heroku PostgreSQL as a Service (shared and dedicated database options)
- MariaDB SkySQL
- Microsoft Azure SQL Database (MS SQL)
- Oracle Database Cloud Service
- Snowflake Cloud Data Warehouse
- Xeround Cloud Database* – MySQL front-end (*service no longer available)

Database as a Service – NoSQL

- Amazon DynamoDB
- Amazon SimpleDB
- Azure Cosmos DB
- Cloudant Data Layer (CouchDB)
- EnterpriseDB Postgres Plus Cloud Database
- Google Cloud Bigtable
- Google Cloud Datastore
- MongoDB Database as a Service (several options)
- RavenDB Cloud Database as a Service
- Oracle NoSQL Database Cloud Service
- Amazon DocumentDB

Database Types



Comparison

SQL	NOSQL
Relational Database management system	Distributed Database management system
Vertically Scalable	Horizontally Scalable
Fixed or predifined Schema	Dynamic Schema
Not suitable for hierarchical data storage	Best suitable for hierarchical data storage
Can be used for complex queries	Not good for complex queries

Why NoSQL?

- NoSQL databases are used in nearly every industry.
- Use cases range from the highly critical (e.g., storing financial data and healthcare records) to the more fun and frivolous (e.g., storing IoT readings from a smart kitty litter box).

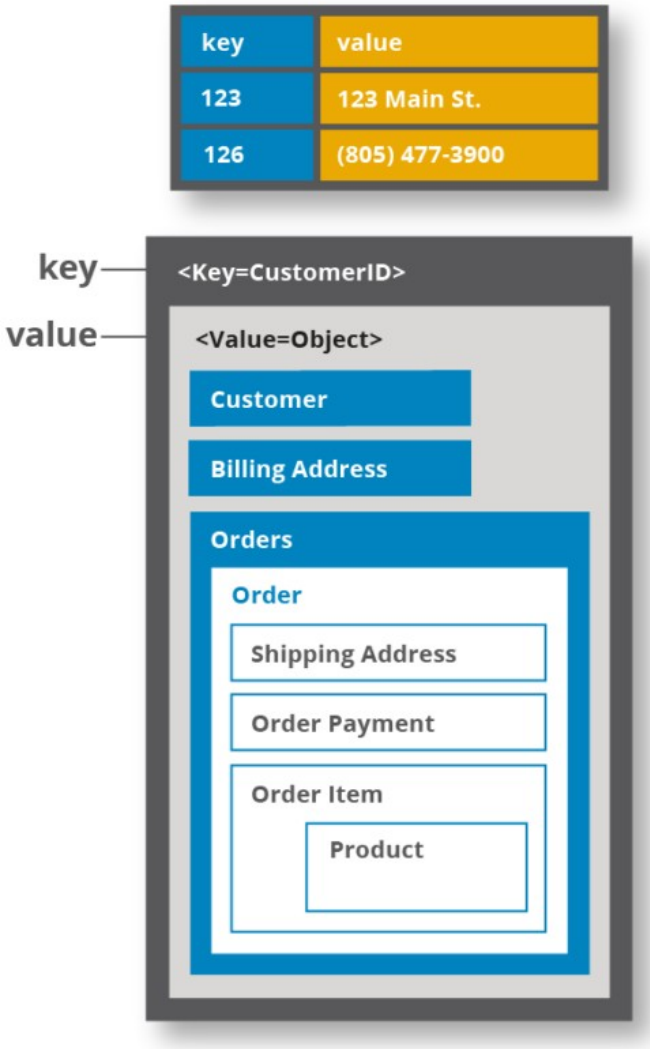
Storage Architectures

- Within the NoSQL system, there are multiple ways to store and retrieve records that surpass the limitations of relational systems. The primary ways data can be stored in NoSQL include:
  - Key-Value Store
  - Document Store
  - Column Store
  - Graph Store
  - Time Series Store
- Some NoSQL databases work with more than one type of record store.

Key-Value Store

- The key-value store is a database system that stores record assets of unique identifiers with an associated (paired) value.
- This data pairing is referred to as a “key-value pair.” The “key” is the unique identifier. The “value” is the data being identified, or its location.
- A major benefit of key-value stores is that they are fast for data retrieval. Where relational systems store data across rows and columns and need to query across the database to return a record, the key-value store is more flexible and only has to search for the key, then return the associated value.

Key-Value Store



Key-Value Store

- Due to the speed of returns, and their flexibility, key-value stores are particularly useful in certain cases such as:
  - Storing, recalling, and updating product information, pricing, categories, and other eCommerce-related functions.
  - Storing user details, preferences, and session information for rapid recall and rewrites.
  - Generating real-time data to provide relevant advertising as users move through different areas of a platform or website.

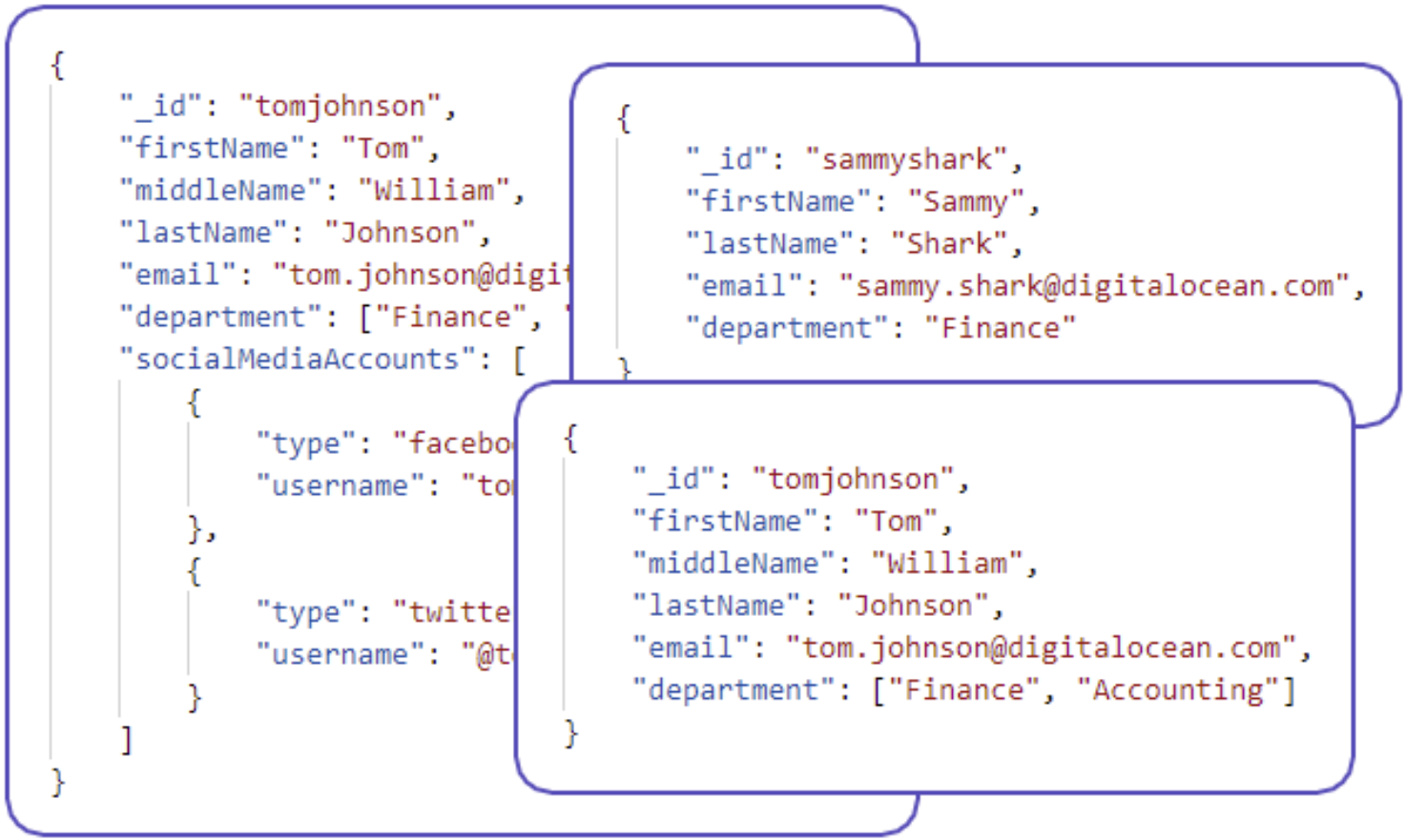
Key-Value Store

- The ability to minimize reads and writes, and to quickly locate datasets based on unique identifiers makes the key-value store a blazing fast option that outperforms relational databases in almost every way for businesses that deal in retail, advertising, eCommerce, and other web applications.

Document Store

- Another storage option, the document-store, stores data in a semi-structured document. The data can then be ordered with markers.
- Information in this data type needs to be encoded in XML, JSON, BSON, or as a YAML file and is never stored in a table (which is why it is unsuitable for relational storage). Instead, complex datasets are contained in a single record.
- Retrieval occurs when a key is used to locate the document, and then that document is searched for the information required.

Document Store



Document Store

- A benefit of the document store is that different types of documents can be contained within a single store, and updates to those documents do not need to be related to the database.
- Also, because there are no fields within this store, and therefore, no empty cells for missing records, the document store is incredibly efficient at returning data fast.

Document Store

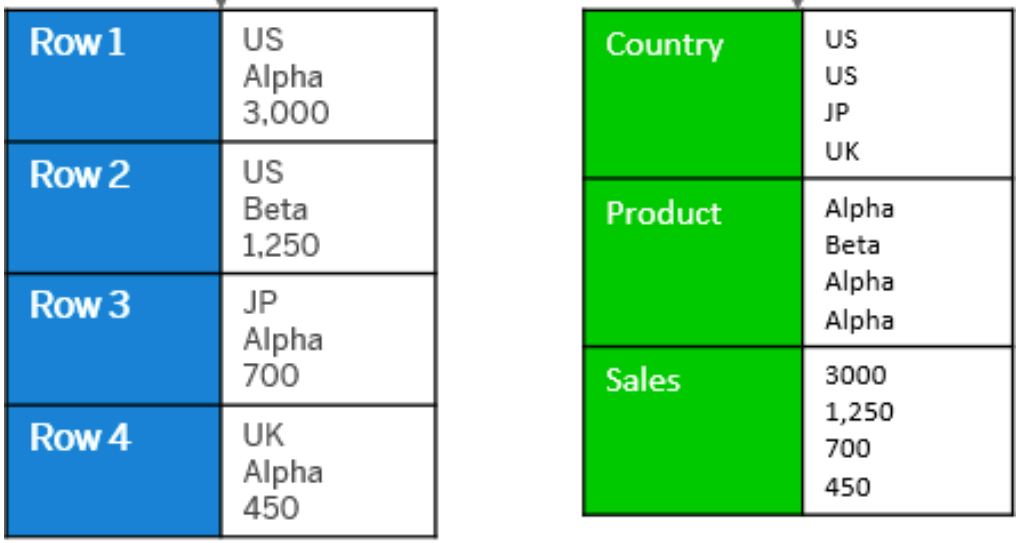
- Document stores are highly useful when:
  - When working with JSON, BSON, XML, YAML files
  - You need to make changes to your data schema often
  - When you work with unstructured or semi-structured data
  - When you need something simple for development
- Document stores are flexible and easily scalable, and developers can work within them, even without prior knowledge of the system.

Column Store

- Column-based stores can also be a good NoSQL storage option. It records data in columns, rather than in rows.
- By storing data in columns, it is contained as a single, ongoing entry.
- This minimizes the number of disks accessed and avoids pulling in unnecessary memory, which speeds up record retrieval since a query does not need to pass over irrelevant rows to return information. Instead, only the information within the column is queried.

Column Store

Country	Product	Sales
US	Alpha	3,000
US	Beta	1,250
JP	Alpha	700
UK	Alpha	450



Row-based storage

Column-based storage

Column Store

- Column stores are most frequently used by companies that deal with large data warehousing setups.
- The data is structured as a table with columns and rows, and is then stored logically in a column-wise format so irrelevant data does not have to be bypassed before the target data is accessed and returned.

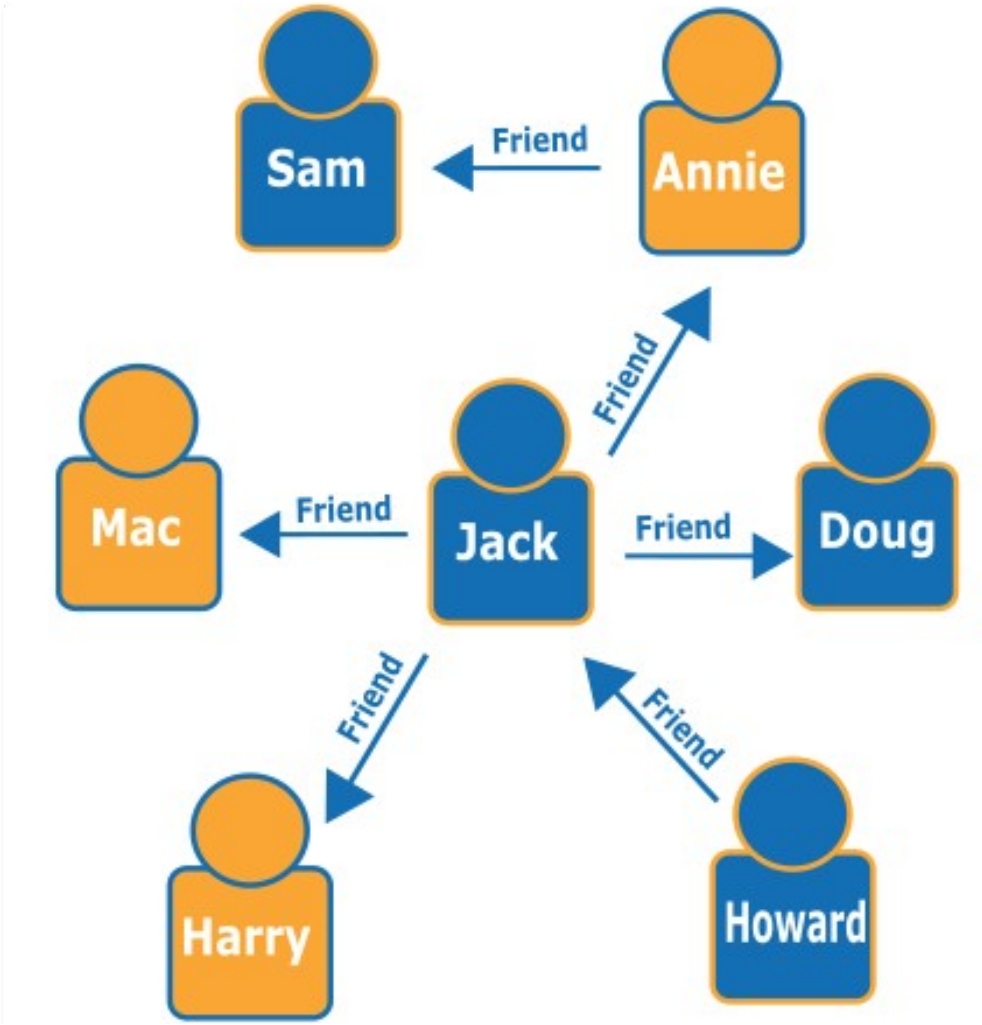
Column Store

- Column store databases are best for:
  - Applications with many reads and few writes
  - When your data has a lot of repetitive records for each value
  - Data warehousing operations
  - Increasing retrieval speed and decreasing memory usage
- Column stores can save you time and computing power, especially when you have a lot of information that repeats, such as rows of names, addresses, phone numbers, and any other records that might be stored under individual data points.

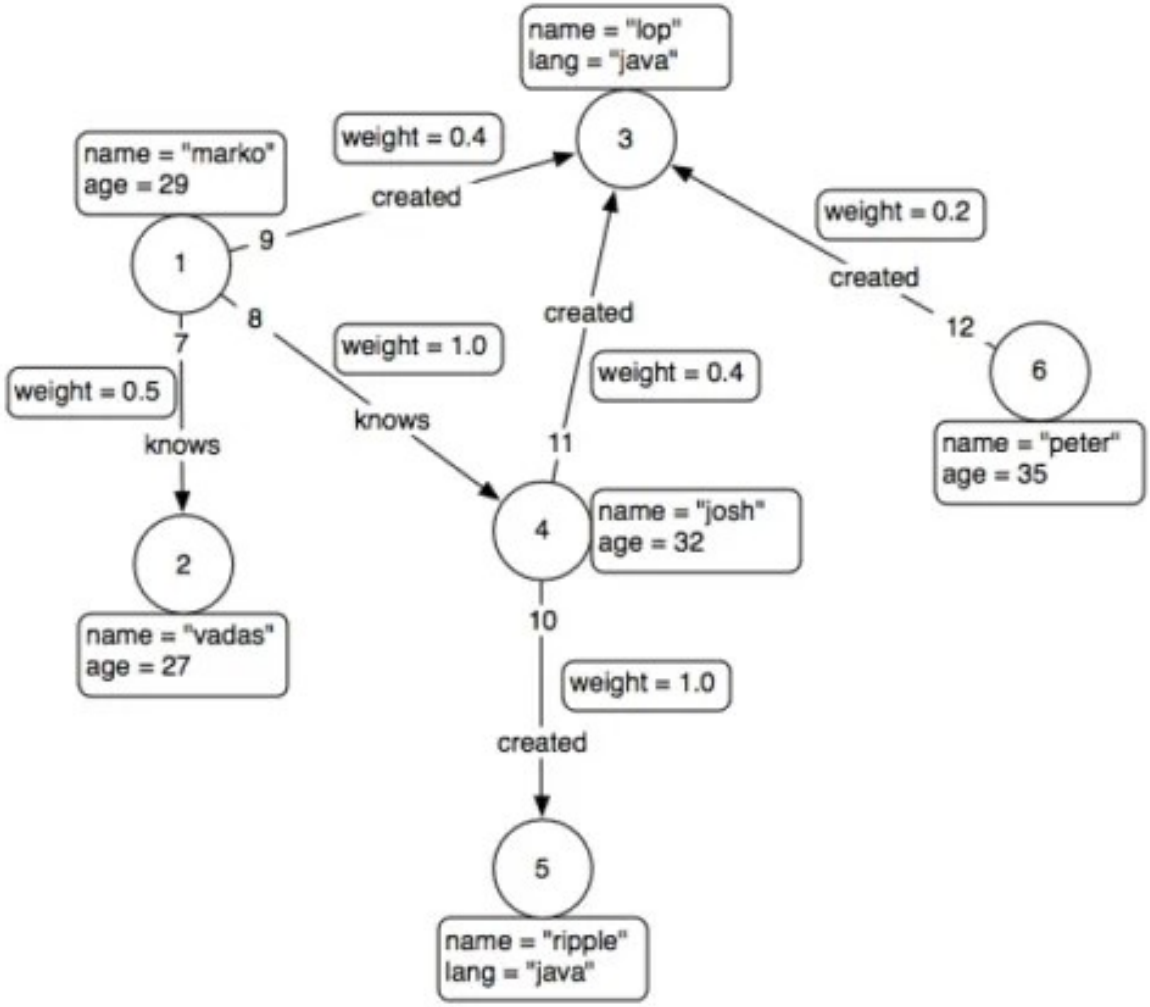
Graph Store

- For some businesses, relationships and connections between data take priority, so a graph NoSQL storage makes the most sense.
- Graph stores represent data in graphs instead of tables which makes them highly flexible and easily extendable.
- The graph NoSQL storage database returns search results fast and speeds up indexing by representing data as networks of nodes and edges.
- In a graph store, data is stored in nodes and then connected with relationships in edges which are then grouped by labels.

Graph Store


Sam
Friend
Annie
Friend
Mac
Friend
Jack
Friend
Doug
Friend
Friend
Harry
Howard

Graph Store


name = "lop"
lang = "java"
3
weight = 0.4
created
name = "marko"
age = 29
1
9
created
weight = 0.2
created
12
6
name = "peter"
age = 35
8
7
weight = 1.0
weight = 0.4
created
weight = 0.5
knows
knows
11
4
name = "josh"
age = 32
2
name = "vadas"
age = 27
10
weight = 1.0
created
5
name = "ripple"
lang = "java"

Graph Store

- Graph stores are most useful for things like:
  - Data visualization and graph-style analytics
  - Fraud prevention and enterprise operations
  - Geospatial routing
  - Payment systems
  - Social networking systems

Time-Series Store

- A final NoSQL storage option is the time series store which is primarily used for managing datasets that change over time.
- Time series store captures fixed and dynamic value sets and returns timely analytics.
- Imagine a car lot with multiple cars. The time series store might have a fixed value data point for each car and then tracks the dynamic values within each car such as oil levels, tire pressure, etc. alongside a timestamp to allow the end-user to see how these metrics have changed over time.

Time-Series Store

- Time series stores are valuable for:
  - Continuously capturing a stream of metrics
  - Analyzing datasets over periods
  - Predictive analysis (i.e. predicting when a car's oil will need to be changed)
  - Monitoring the status of various systems with easily accessible analytics
- Time series store is best for businesses where multiple systems require ongoing measurements within individual data points.

Why NoSQL?

- Support large numbers of concurrent users (tens of thousands, perhaps millions)
- Deliver highly responsive experiences to a globally distributed base of users
- Be always available – no downtime
- Handle semi- and unstructured data
- Rapidly adapt to changing requirements with frequent updates and new features

When NoSQL?

- When deciding which database to use, decision-makers typically find one or more of the following factors lead them to selecting a NoSQL database:
  - Fast-paced Agile development
  - Storage of structured and semi-structured data
  - Huge volumes of data
  - Requirements for scale-out architecture
  - Modern application paradigms like microservices and real-time streaming

Misconcepts

- Over the years, many misconceptions about NoSQL databases have spread throughout the developer community. Two of the most common misconceptions:
  - Relationship data is best suited for relational databases.
  - NoSQL databases don't support ACID transactions.

Misconcepts

- A common misconception is that NoSQL databases or non-relational databases don’t store relationship data well. NoSQL databases can store relationship data — they just store it differently than relational databases do.
- In fact, when compared with relational databases, many find modeling relationship data in NoSQL databases to be easier than in relational databases, because related data doesn’t have to be split between tables.
- NoSQL data models allow related data to be nested within a single data structure.

Misconcepts

- Another common misconception is that NoSQL databases don't support ACID transactions. Some NoSQL databases like MongoDB do, in fact, support ACID transactions.
- Note that the way data is modeled in NoSQL databases can eliminate the need for multi-record transactions in many use cases. Consider the earlier example where we stored information about a user and their hobbies in both a relational database and a document database.
- In order to ensure information about a user and their hobbies was updated together in a relational database, we'd need to use a transaction to update records in two tables.
- In order to do the same in a document database, we could update a single document — no multi-record transaction required.

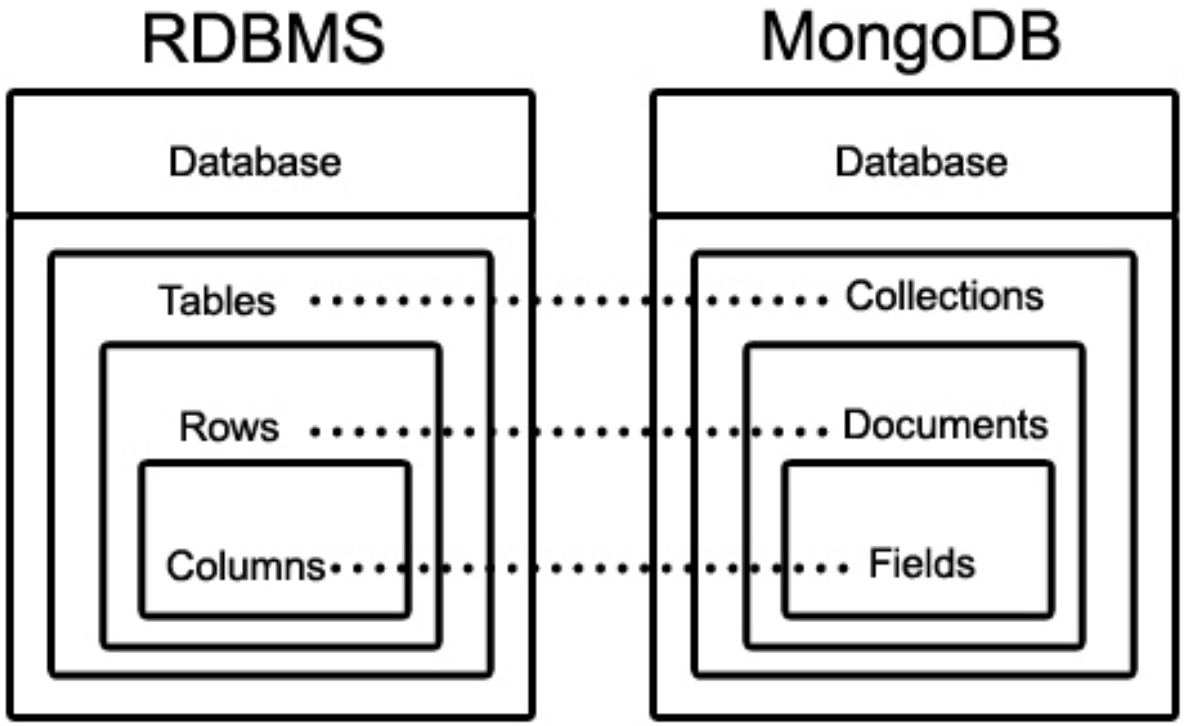
Terminologies

- Database
  - Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.
- Collection
  - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database.
  - Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.
- Document
  - A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

RDBMS vs. MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

RDBMS vs. MongoDB



Example Document

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'MiTU Skillologies',
   url: 'https://www.mitu.co.in',
   tags: ['mongodb', 'database', 'NoSQL'],
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

Advantages

- Schema less – MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Ease of scale-out – MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why to use mongodb?

- Document Oriented Storage – Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to use ?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Columnar Database

- Columnar databases are NoSQL databases built for highly analytical, complex-query tasks. Unlike relational databases, columnar databases store their data by columns, rather than by rows. These columns are gathered to form subgroups.
- The keys and the column names of this type of database are not fixed. Columns within the same column family, or cluster of columns, can have a different number of rows and can accommodate different types of data and names.
- These databases are most often utilized when there is a need for a large data model. They are very useful for data warehouses, or when there is a need for high performance or handling intensive querying.

Columnar Database : How?

- Relational databases have a set schema and they function as tables of rows and columns. Wide-column databases have a similar, but different schema.
- They also have rows and columns. However, they are not fixed within a table, but have a dynamic schema. Each column is stored separately.
- If there are similar (related) columns, they are joined into column families and then the column families are stored separately from other column families.

Columnar Database : How?

- The row key is the first column in each column family, and it serves as an identifier of a row.
- Furthermore, each column after that has a column key (name). It identifies columns within rows and thus enables the querying of the columns.
- The value and the timestamp come after the column key, leaving a trace of when the data was entered or modified.

Columnar Database : How?

- The number of columns pertaining to each row, or their name, can vary. In other words, not every column of a column family, and thus a database, has the same number of rows.
- In fact, even though they might share their name, each column is contained within one row and does not run across all rows.

Column oriented vs. Row Oriented

ID NUMBER	LAST NAME	FIRST NAME	BONUS
513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

Row-oriented database

513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

Column-oriented database

513001	502333	455332
Jones	Smith	Beck
Joanna	Jamie	Samuel
8000	4000	1000

Columnar Database

- Those who have encountered relational databases know that each column of a relational database has the same number of rows, but it happens that some of the fields have a null value, or they appear to be empty.
- With wide-column databases, rather than being empty, these rows simply do not exist for a particular column.

Columnar Database

- The column families are located in a keyspace.
- Each keyspace holds an entire NoSQL data store and it has a similar role or importance that a schema has for a relational database.
- However, as NoSQL datastores have no set structure, keyspaces represent a schemaless database containing the design of a data store and its own set of attributes.

Columnar Database

- One of the most popular columnar databases available is MariaDB.
- It was created as a fork of MySQL intended to be robust and scalable, handle many different purposes and a large volume of queries.
- Apache Cassandra is another example of a columnar database handling heavy data loads across numerous servers, making the data highly available.
- Some of the other names on this list include Apache HBase, Hypertable and Druid specially designed for analytics.
- These databases support certain features of platforms such as Outbrain, Spotify and Facebook.

Column family types

- Standard column family.
  - This column family type is similar to a table; it contains a key-value pair where the key is the row key, and the values are stored in columns using their names as their identifiers.

Column family types

- Super column family.
  - A super column represents an array of columns. Each super column has a name and a value mapping the super column out to several different columns.
  - Related super columns are joined under a single row into super column families. Compared to a relational database, this is like a view of several different tables within a database.
  - Imagine you had the view of the columns and values available for a single row -- that is a single identifier across many different tables -- and were able to store them all in one place: That is the super column family.

Columnar Databases: Advantages

- Scalability. This is a major advantage and one of the main reasons this type of database is used to store big data. With the ability to be spread over hundreds of different machines depending on the scale of the database, it supports massively parallel processing.
- Compression. Not only are they infinitely scalable, but they are also good at compressing data and thus saving storage.
- Very responsive. The load time is minimal, and queries are performed fast, which is expected given that they are designed to hold big data and be practical for analytics.

Columnar Databases: Diasdvantages

- Online transactional processing. These databases are not very efficient with online transactional processing as much as they are for online analytical processing.
- Incremental data loading. Typically column-oriented databases are used for analysis and are quick to retrieve data, even when processing complex queries, as it is kept close together in columns.
- Row-specific queries. Like the potential downfalls mentioned above, it all boils down to the same issue, which is using the right type of database for the right purposes.

Document Databases

- What are documents?
  - A document is a record in a document database. A document typically stores information about one object and any of its related metadata.
  - Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML.
- Below is a JSON document that stores information about a user named Tom.

Reference: mongodb.com

Document Databases

```
{
     "_id": 1,
     "first_name": "Tom",
     "email": "tom@example.com",
     "cell": "765-555-5555",
     "likes": [
        "fashion",
        "spas",
        "shopping"
     ],
     "businesses": [
        {
           "name": "Entertainment 1080",
           "partner": "Jean",
           "status": "Bankrupt",
           "date_founded": {
              "$date": "2012-05-19T04:00:00Z"
           }
        },
        {
           "name": "Swag for Tweens",
           "date_founded": {
              "$date": "2012-11-01T04:00:00Z"
           }
        }
     ]
  }
```

Collections

- A collection is a group of documents. Collections typically store documents that have similar contents.
- Not all documents in a collection are required to have the same fields, because document databases have a flexible schema.
- Note that some document databases provide schema validation, so the schema can optionally be locked down when needed.

Collections

- Continuing with the example above, the document with information about Tom could be stored in a collection named users.
- More documents could be added to the users collection in order to store information about other users.
- For example, the document below that stores information about Donna could be added to the users collection.

Collections

```
{
    "_id": 2,
    "first_name": "Donna",
    "email": "donna@example.com",
    "spouse": "Joe",
    "likes": [
       "spas",
       "shopping",
       "live tweeting"
    ],
    "businesses": [
       {
          "name": "Castle Realty",
          "status": "Thriving",
          "date_founded": {
             "$date": "2013-11-21T04:00:00Z"
          }
       }
    ]
 }
```

CRUD Operations

- Document databases typically have an API or query language that allows developers to execute the CRUD (create, read, update, and delete) operations.
  - Create: Documents can be created in the database. Each document has a unique identifier.
  - Read: Documents can be read from the database. The API or query language allows developers to query for documents using their unique identifiers or field values. Indexes can be added to the database in order to increase read performance.
  - Update: Existing documents can be updated — either in whole or in part.
  - Delete: Documents can be deleted from the database.

Key Features

- Document model:
  - Data is stored in documents (unlike other databases that store data in structures like tables or graphs). Documents map to objects in most popular programming languages, which allows developers to rapidly develop their applications.
- Flexible schema:
  - Document databases have a flexible schema, meaning that not all documents in a collection need to have the same fields. Note that some document databases support schema validation, so the schema can be optionally locked down.

Key Features

- Distributed and resilient:
  - Document databases are distributed, which allows for horizontal scaling (typically cheaper than vertical scaling) and data distribution. Document databases provide resiliency through replication.
- Querying through an API or query language:
  - Document databases have an API or query language that allows developers to execute the CRUD operations on the database. Developers have the ability to query for documents based on unique identifiers or field values.

How different from RDBMS?

- The intuitiveness of the data model:
  - Documents map to the objects in code, so they are much more natural to work with. There is no need to decompose data across tables, run expensive joins, or integrate a separate Object Relational Mapping (ORM) layer.
  - Data that is accessed together is stored together, so developers have less code to write and end users get higher performance.

How different from RDBMS?

- The ubiquity of JSON documents:
  - JSON has become an established standard for data interchange and storage. JSON documents are lightweight, language-independent, and human-readable.
  - Documents are a superset of all other data models so developers can structure data in the way their applications need — rich objects, key-value pairs, tables, geospatial and time-series data, or the nodes and edges of a graph.

How different from RDBMS?

- The flexibility of the schema:
  - A document's schema is dynamic and self-describing, so developers don't need to first pre-define it in the database. Fields can vary from document to document.
  - Developers can modify the structure at any time, avoiding disruptive schema migrations.
  - Some document databases offer schema validation so you can optionally enforce rules governing document structures.

Document vs. Others

- Key-value pairs can be modeled with fields and values in a document. Any field in a document can be indexed, providing developers with additional flexibility in how to query the data.
- Relational data can be modeled differently (and some would argue more intuitively) by keeping related data together in a single document using embedded documents and arrays.
  - Related data can also be stored in separate documents, and database references can be used to connect the related data.

Document vs. Others

- Documents map to objects in most popular programming languages.
- Graph nodes and/or edges can be modeled as documents. Edges can also be modeled through database references. Graph queries can be run using operations like $graphLookup.
- Geospatial data can be modeled as arrays in documents.

Strengths

- The document model is ubiquitous, intuitive, and enables rapid software development.
- The flexible schema allows for the data model to change as an application's requirements change.
- Document databases have rich APIs and query languages that allow developers to easily interact with their data.
- Document databases are distributed (allowing for horizontal scaling as well as global data distribution) and resilient.

Weakness

- A common weakness that people cite about document databases is that many do not support multi-document ACID transactions.
- We estimate that 80%-90% of applications that leverage the document model will not need to use multi-document transactions.

Use Cases

- Document databases are general-purpose databases that serve a variety of use cases for both transactional and analytical applications:
  - Single view or data hub
  - Customer data management and personalization
  - Internet of Things (IoT) and time-series data
  - Product catalogs and content management
  - Payment processing
  - Mobile apps
  - Mainframe offload
  - Operational analytics
  - Real-time analytics

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com