CST 204 – Database Management Systems

Module 5

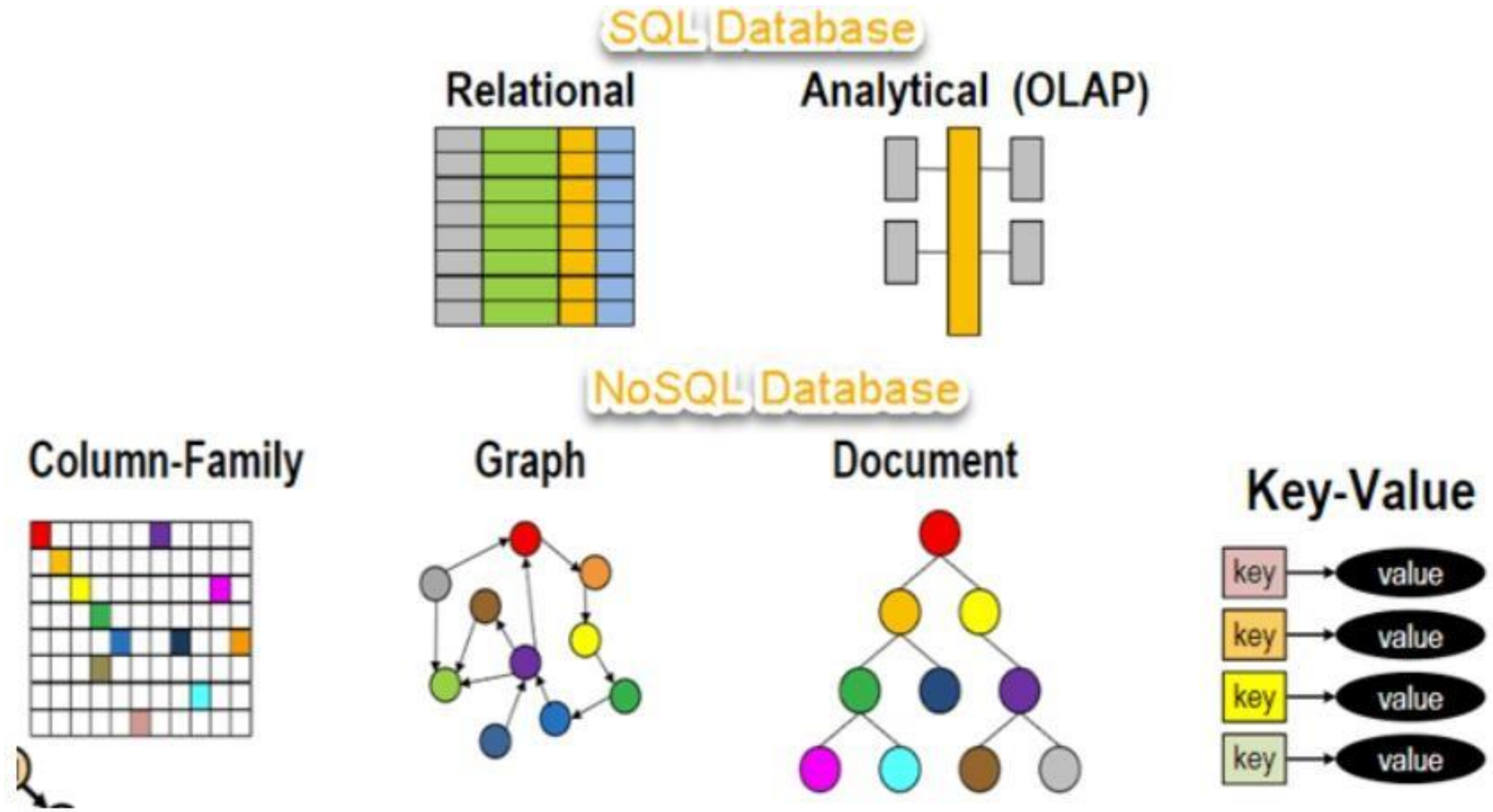
- Transaction Processing Concepts - overview of concurrency control, Transaction Model, Significance of concurrency Control & Recovery, Transaction States, System Log, Desirable Properties of transactions.
- Serial schedules, Concurrent and Serializable Schedules, Conflict equivalence and conflict serializability, Recoverable and cascade-less schedules, Locking, Two-phase locking and its variations. Log-based recovery, Deferred database modification, check-pointing.
- Introduction to NoSQL Databases, Main characteristics of Key-value DB (examples from: Redis), Document DB (examples from: MongoDB)
- Main characteristics of Column - Family DB (examples from: Cassandra) and Graph DB (examples from : ArangoDB)

Introduction to NoSQL Databases

What is NoSQL?

- NoSQL database stands for “Not Only SQL” or “Not SQL.”
- It is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

What is NoSQL?



Why NoSQL?

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could “scale up” our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as “scaling out.”
- NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

Features of NoSQL

- Non-relational
  - NoSQL databases never follow the relational model
  - Never provide tables with flat fixed-column records
  - Work with self-contained aggregates or BLOBs
  - Doesn’t require object-relational mapping and data normalization
  - No complex features like query languages, query planners,referential integrity joins, ACID
- Schema-free
  - NoSQL databases are either schema-free or have relaxed schemas
  - Do not require any sort of definition of the schema of the data
  - Offers heterogeneous structures of data in the same domain

Features of NoSQL

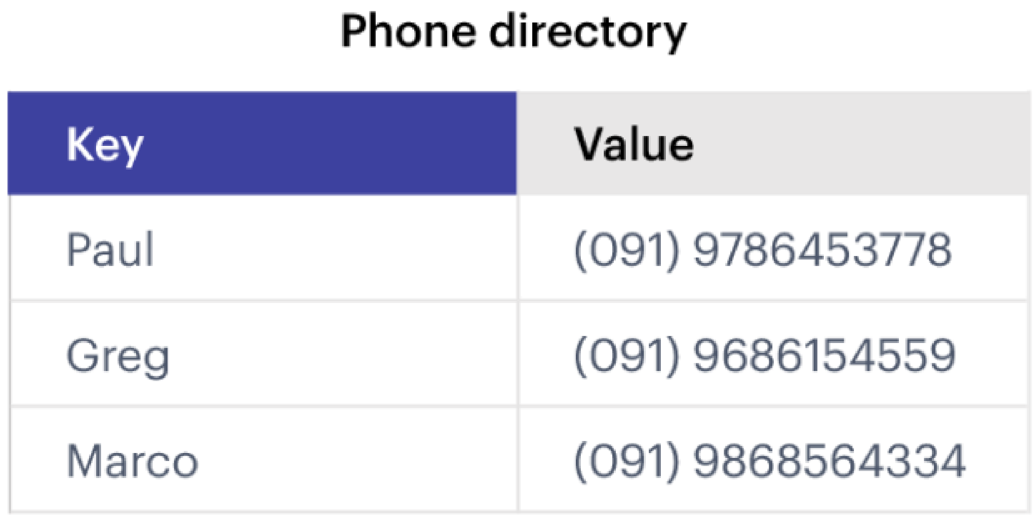
- Simple API
  - Offers easy to use interfaces for storage and querying data provided
  - APIs allow low-level data manipulation & selection methods
  - Text-based protocols mostly used with HTTP REST with JSON
  - Mostly used no standard based NoSQL query language
  - Web-enabled databases running as internet-facing services
- Distributed
  - Multiple NoSQL databases can be executed in a distributed fashion
  - Offers auto-scaling and fail-over capabilities
  - Often ACID concept can be sacrificed for scalability and throughput
  - Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
  - Only providing eventual consistency
  - Shared Nothing Architecture. This enables less coordination and higher distribution.

Types of NoSQL Databases

- NoSQL Databases are mainly categorized into four types:
  - Key-value Pair Based
  - Column-oriented
  - Graphs based
  - Document-oriented

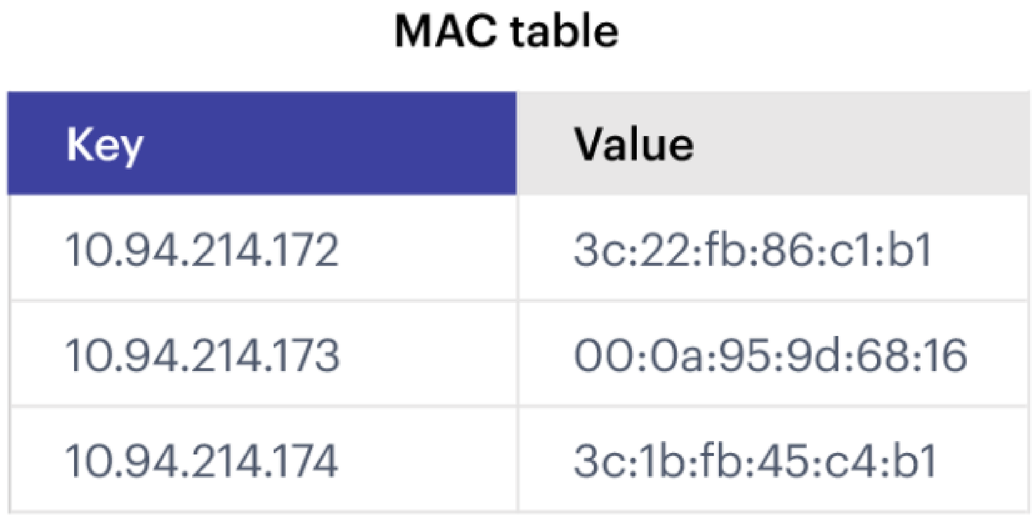
What is a Key-Value Database?

- A key-value database (sometimes called a key-value store) uses a simple key-value method to store data.
- These databases contain a simple string (the key) that is always unique and an arbitrary large data field (the value).
- They are easy to design and implement.



Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334



MAC table

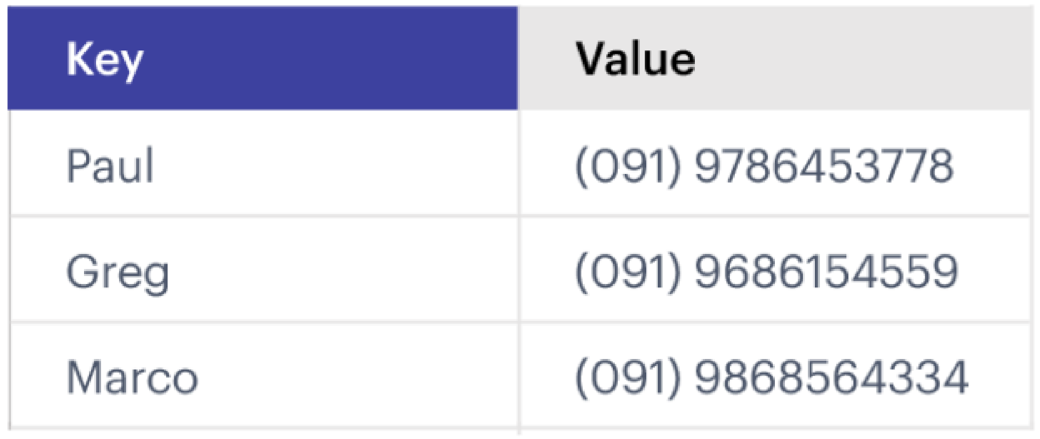
Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

What is a Key-Value Database?

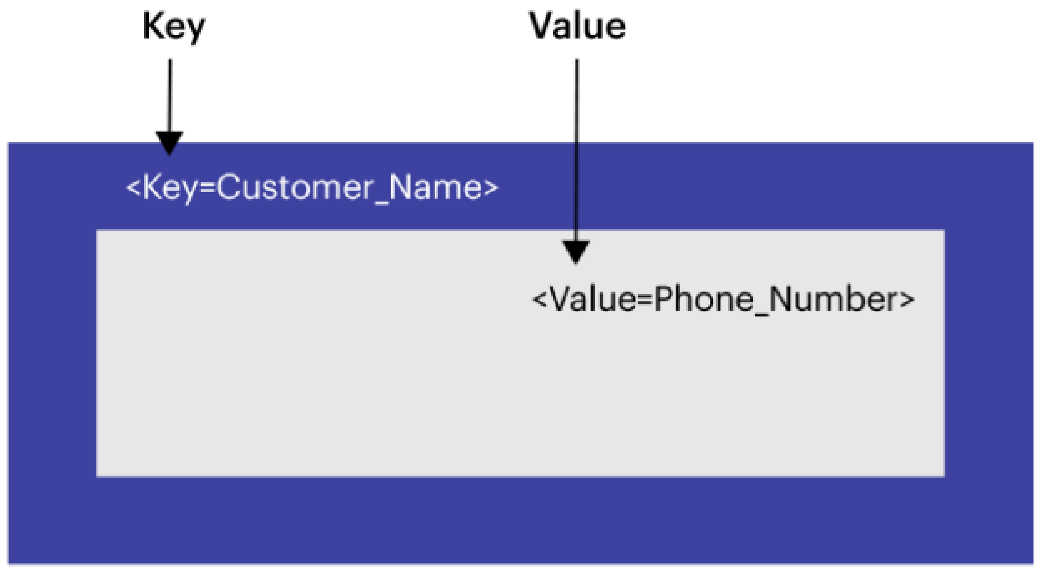
- As the name suggests, this type of NoSQL database implements a hash table to store unique keys along with the pointers to the corresponding data values.
- The values can be of scalar data types such as integers or complex structures such as JSON, lists, BLOB, and so on.
- A value can be stored as an integer, a string, JSON, or an array—with a key used to reference that value.
- It typically offers excellent performance and can be optimized to fit an organization’s needs.
- Key-value stores have no query language but they do provide a way to add and remove key-value pairs.
- Values cannot be queried or searched upon. Only the key can be queried.

What is a Key-Value Database?

Phone directory



Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334



A simple example of key-value data store.

When to use a key-value database

- When your application needs to handle lots of small continuous reads and writes, that may be volatile. Key-value databases offer fast in-memory access.
- When storing basic information, such as customer details; storing webpages with the URL as the key and the webpage as the value; storing shopping-cart contents, product categories, e-commerce product details
- For applications that don’t require frequent updates or need to support complex queries.

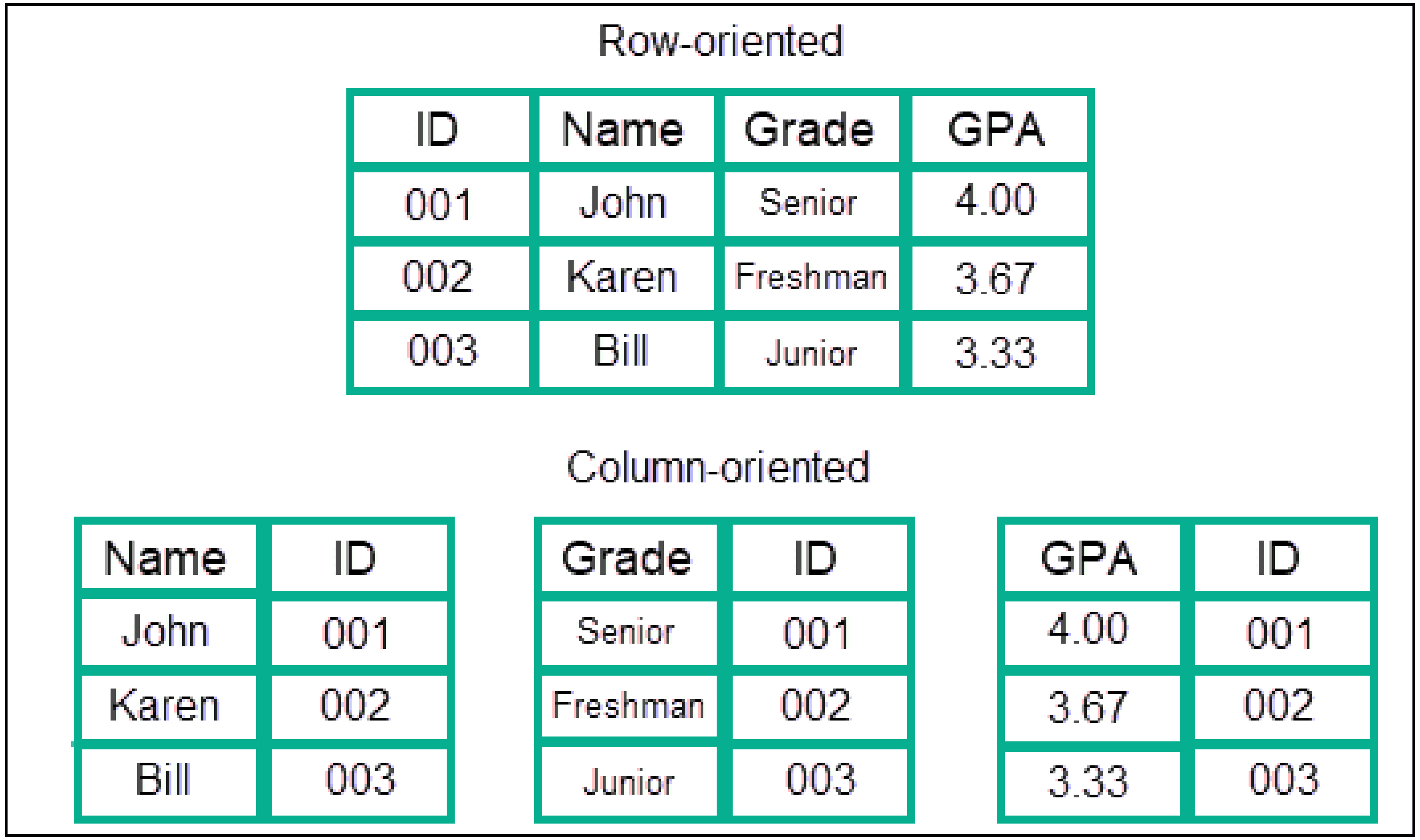
Use cases for key-value databases

- Session management on a large scale.
- Using cache to accelerate application responses.
- Storing personal data on specific users.
- Product recommendations, storing personalized lists of items for individual customers.
- Managing each player’s session in massive multiplayer online games.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases.

Column-oriented

- While a relational database stores data in rows and reads data row by row, a column store is organized as a set of columns.
- When you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.
- Columns are often of the same type and benefit from more efficient compression, making reads even faster.
- Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). Use cases include analytics.

Column-oriented



Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

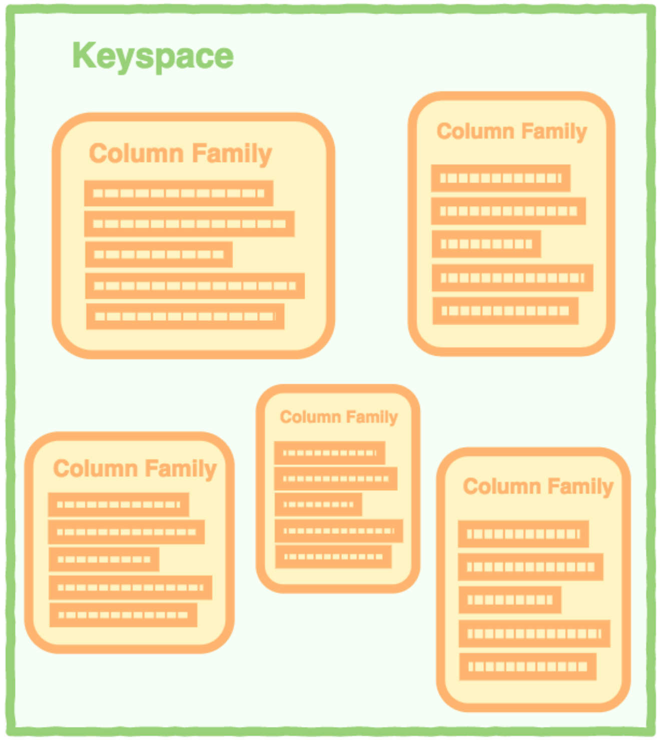
Name	ID
John	001
Karen	002
Bill	003

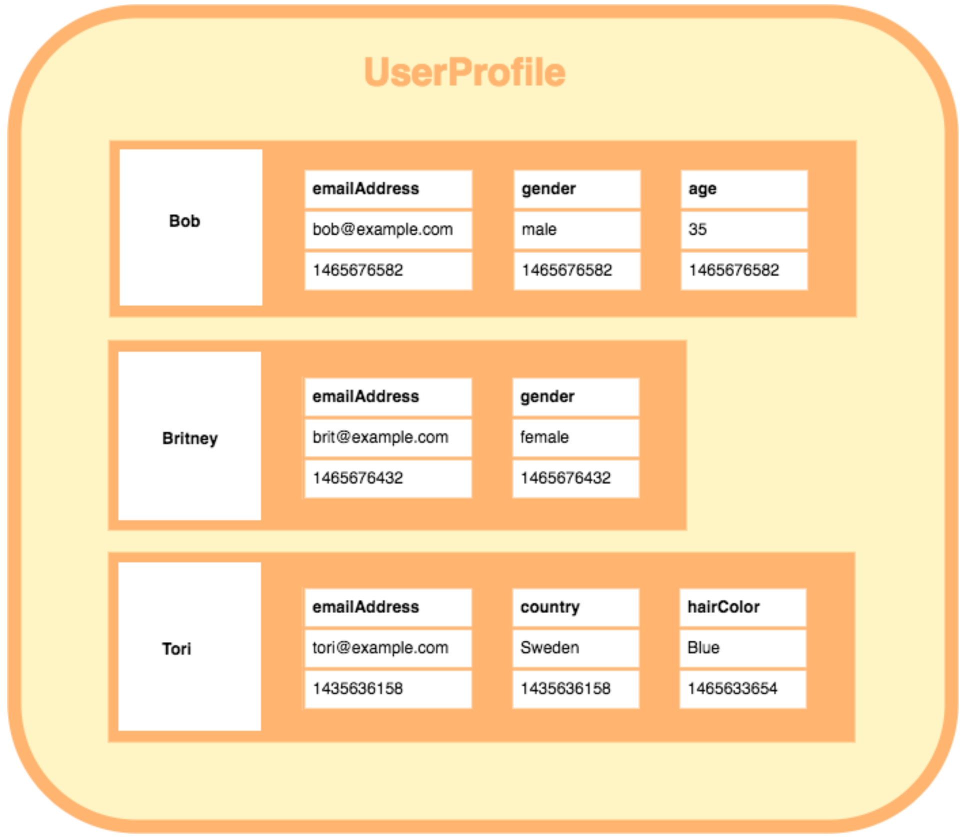
Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

Column-oriented

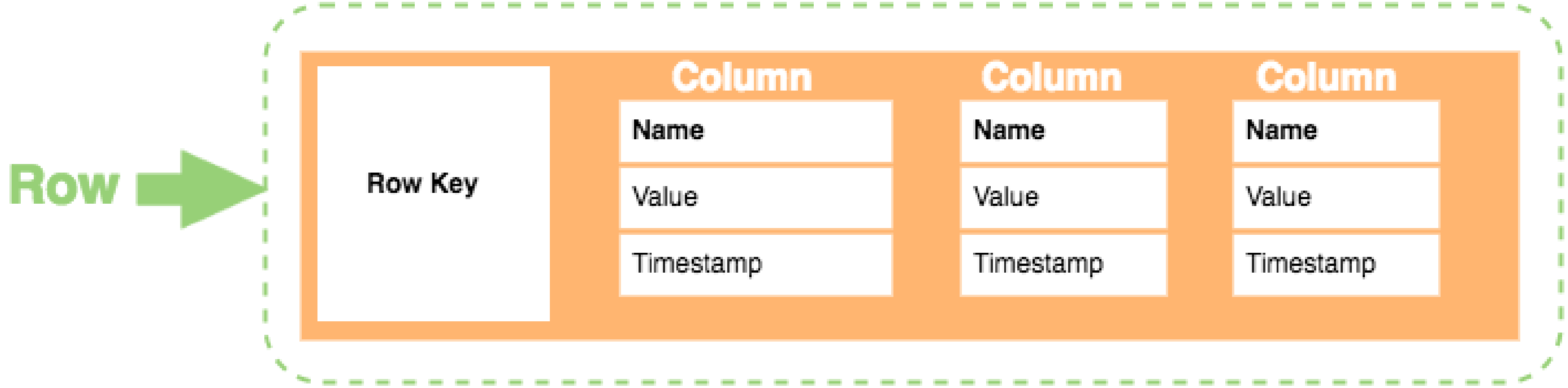
- Column databases use the concept of keyspace, which is sort of like a schema in relational models.
- This keyspace contains all the column families, which then contain rows, which then contain columns.





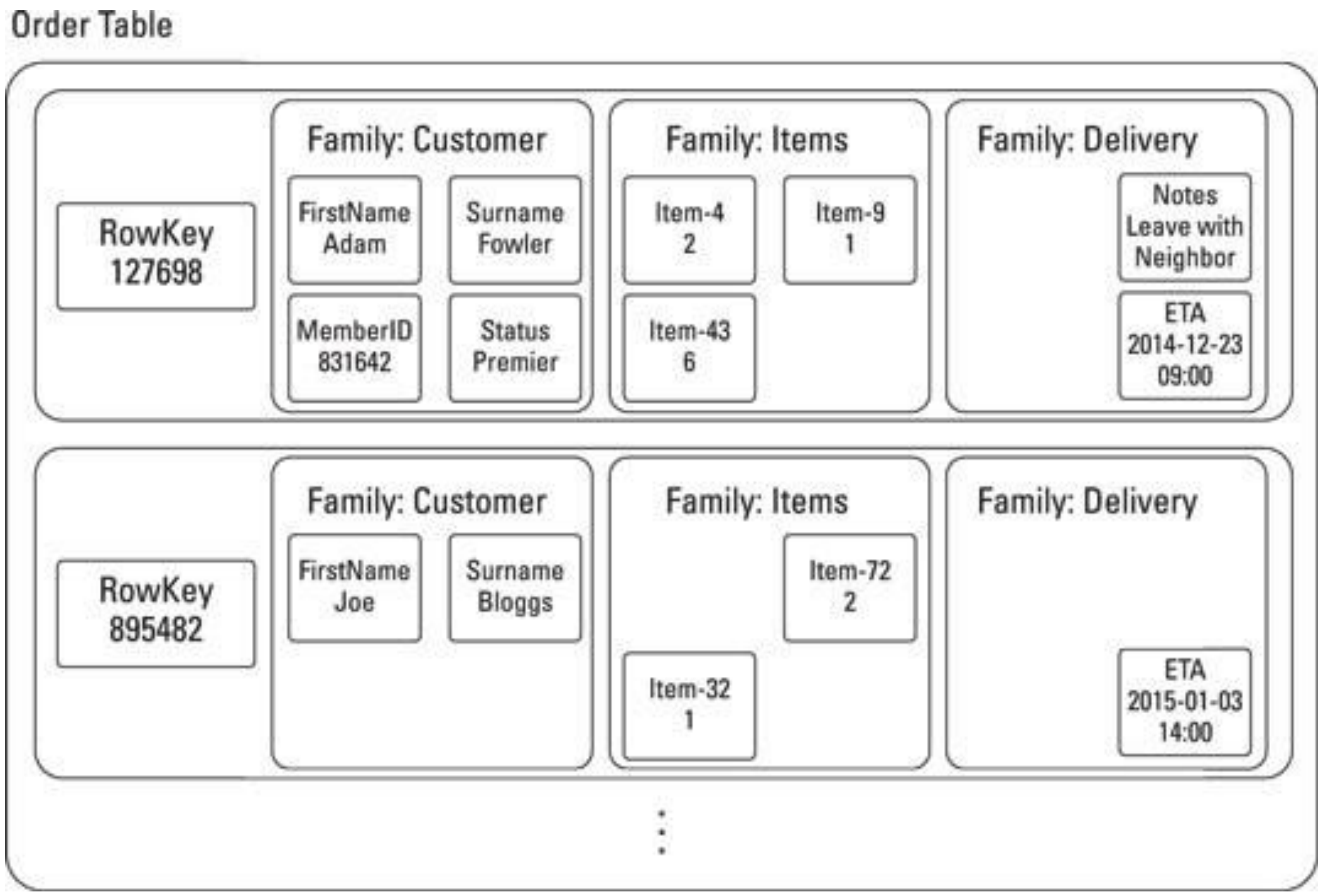
Column-oriented

- If we take a specific row as an example:



- The Row Key is exactly that: the specific identifier of that row and is always unique.
- The column contains the name, value, and timestamp, so that’s straightforward. The name/value pair is also straight forward, and the timestamp is the date and time the data was entered into the database.
- Some examples of column-store databases include Casandra, CosmoDB, Bigtable, and HBase.

Column-oriented



Column-oriented - Use cases

- Developers mainly use column databases in:
  - Content management systems
  - Blogging platforms
  - Systems that maintain counters
  - Services that have expiring usage
  - Systems that require heavy write requests (like log aggregators)

Benefits of Column Databases

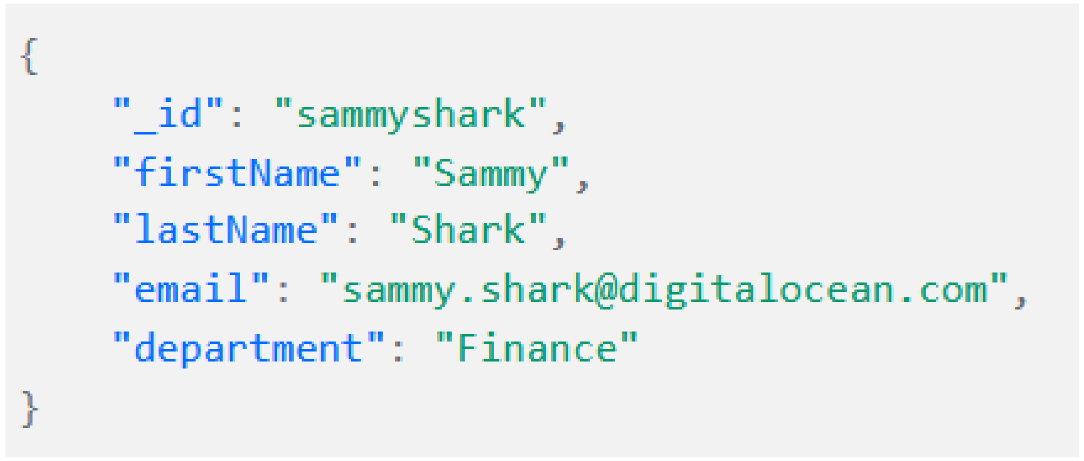
- There are several benefits that go along with columnar databases:
  - Column stores are excellent at compression and therefore are efficient in terms of storage.
    - You can reduce disk resources while holding massive amounts of information in a single column
  - Since a majority of the information is stored in a column, aggregation queries are quite fast, which is important for projects that require large amounts of queries in a small amount of time.
  - Scalability is excellent with column-store databases.
    - They can be expanded nearly infinitely, and are often spread across large clusters of machines, even numbering in thousands.
    - That also means that they are great for Massive Parallel Processing

Benefits of Column Databases

- Load times are similarly excellent, as you can easily load a billion-row table in a few seconds.
  - You can load and query nearly instantly.
- Large amounts of flexibility as columns do not necessarily have to look like each other.
  - You can add new and different columns without disrupting the whole database.

What are document-oriented databases?

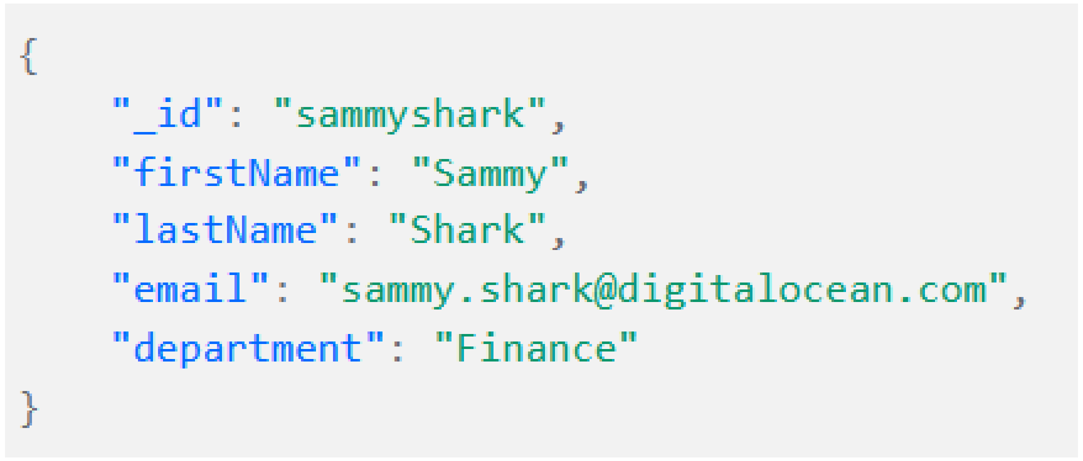
- Is a modernized way of storing data as JSON rather than basic columns/rows — i.e. storing data in its native form.
- This storage system lets you retrieve, store, and manage document-oriented information
- It's a very popular category of modern NoSQL databases, used by the likes of MongoDB, Cosmos DB, DocumentDB, SimpleDB, PostgreSQL, OrientDB, Elasticsearch and RavenDB.



```
{
    "_id": "sammyshark",
    "firstName": "Sammy",
    "lastName": "Shark",
    "email": "sammy.shark@digitalocean.com",
    "department": "Finance"
}
```

- This is an example of a document that might appear in a document database like MongoDB.
- This sample document represents a company contact card, describing an employee called Sammy:

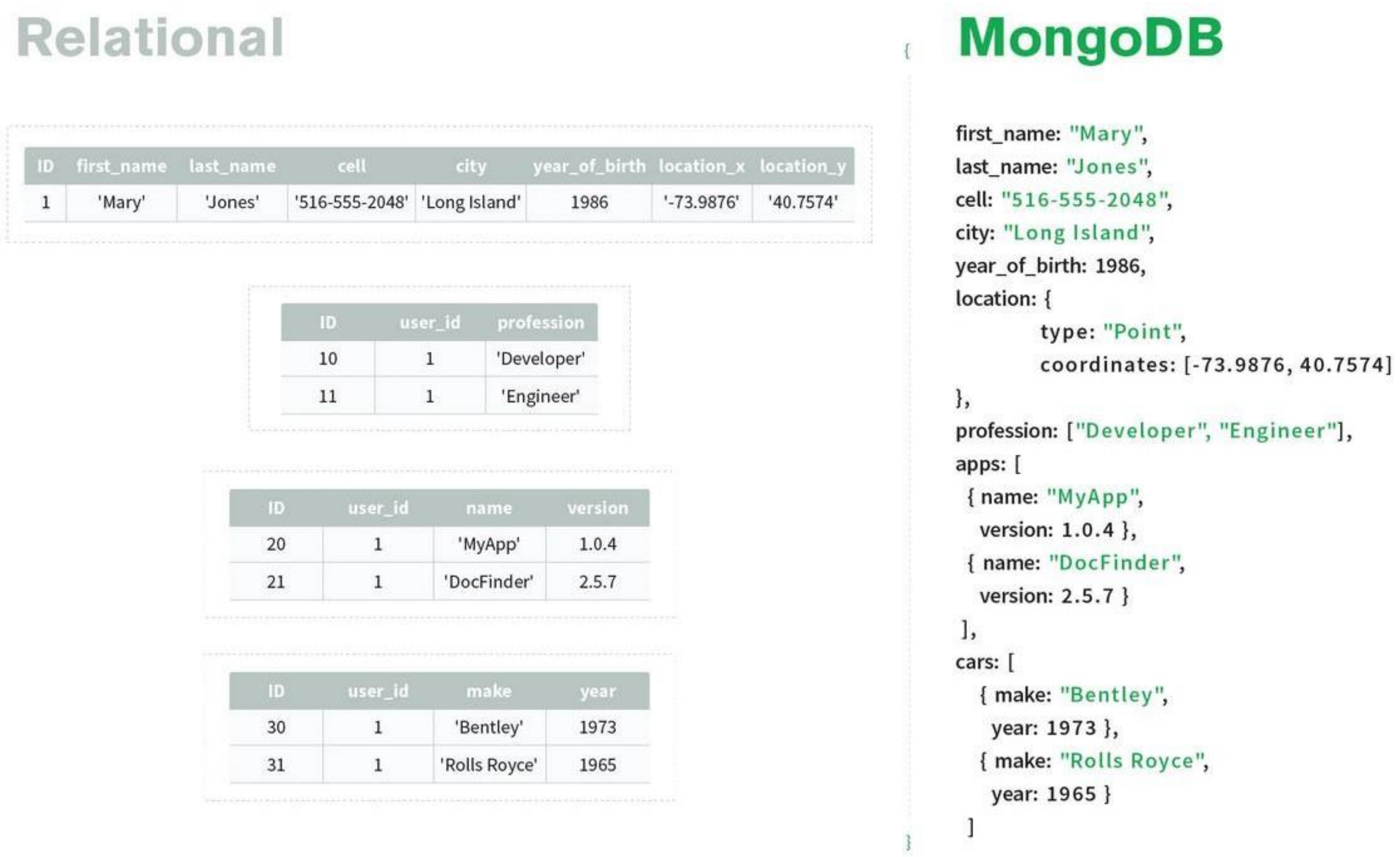
What are document-oriented databases?



```
{
    "_id": "sammyshark",
    "firstName": "Sammy",
    "lastName": "Shark",
    "email": "sammy.shark@digitalocean.com",
    "department": "Finance"
}
```

- Notice that the document is written as a JSON object.
- JSON is a human-readable data format that has become quite popular in recent years.
- While many different formats can be used to represent data within a document database, such as XML or YAML, JSON is one of the most common choices.
- For example, MongoDB adopted JSON as the primary data format to define and manage data.

Relational – Document Database



Relational

ID	first_name	last_name	cell	city	year_of_birth	location_x	location_y
1	'Mary'	'Jones'	'516-555-2048'	'Long Island'	1986	'-73.9876'	'40.7574'

ID	user_id	profession
10	1	'Developer'
11	1	'Engineer'

ID	user_id	name	version
20	1	'MyApp'	1.0.4
21	1	'DocFinder'	2.5.7

ID	user_id	make	year
30	1	'Bentley'	1973
31	1	'Rolls Royce'	1965

MongoDB

```
{
first_name: "Mary",
last_name: "Jones",
cell: "516-555-2048",
city: "Long Island",
year_of_birth: 1986,
location: {
        type: "Point",
        coordinates: [-73.9876, 40.7574]
},
profession: ["Developer", "Engineer"],
apps: [
 { name: "MyApp",
   version: 1.0.4 },
 { name: "DocFinder",
   version: 2.5.7 }
 ],
cars: [
  { make: "Bentley",
    year: 1973 },
  { make: "Rolls Royce",
    year: 1965 }
 ]
}
```

Benefits of Document Databases

- A few of the most important benefits are:
  - **Flexibility and adaptability:** with a high level of control over the data structure, document databases enable experimentation and adaptation to new emerging requirements.
  - New fields can be added right away and existing ones can be changed any time.
  - It's up to the developer to decide whether old documents must be amended or the change can be implemented only going forward.
  - **Ability to manage structured and unstructured data:** Document databases can be used to handle structured data as well, but they're also quite useful for storing unstructured data where necessary.
  - **Scalability by design**: Conversely, document databases are designed as distributed systems that instead allow you to scale horizontally (meaning that you split a single database up across multiple servers).

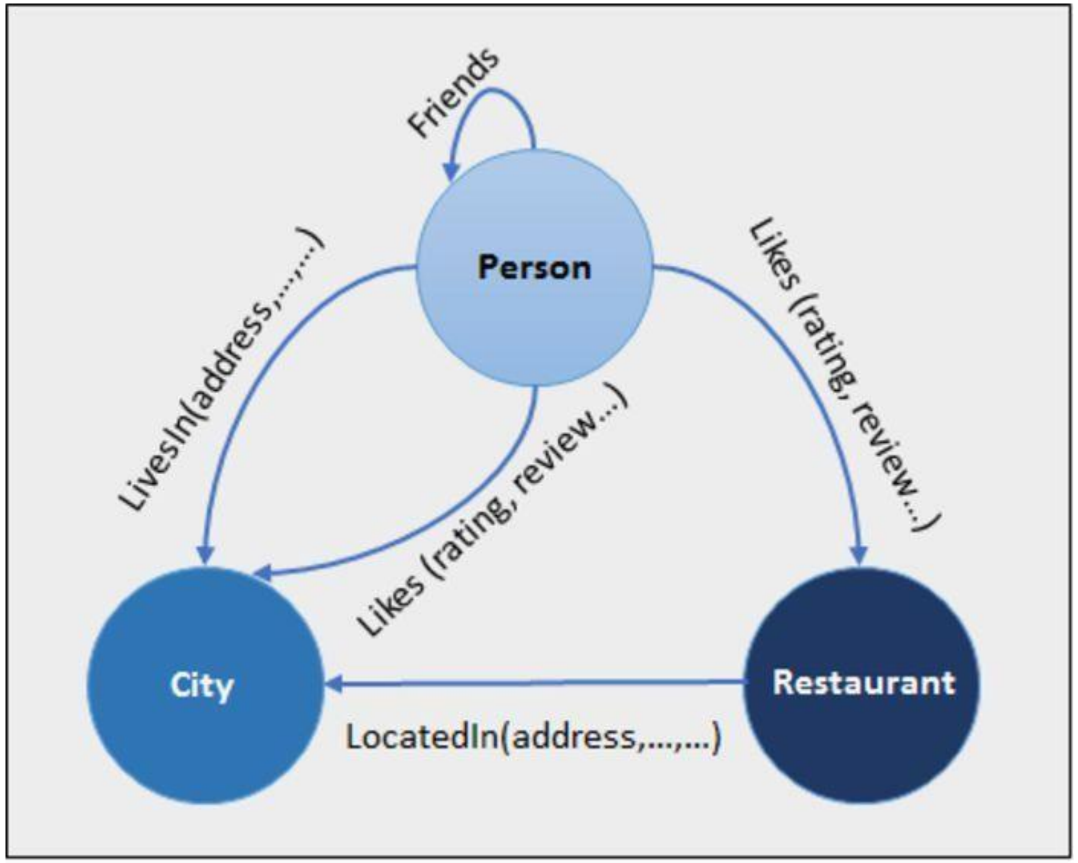
Graph-Based NoSQL

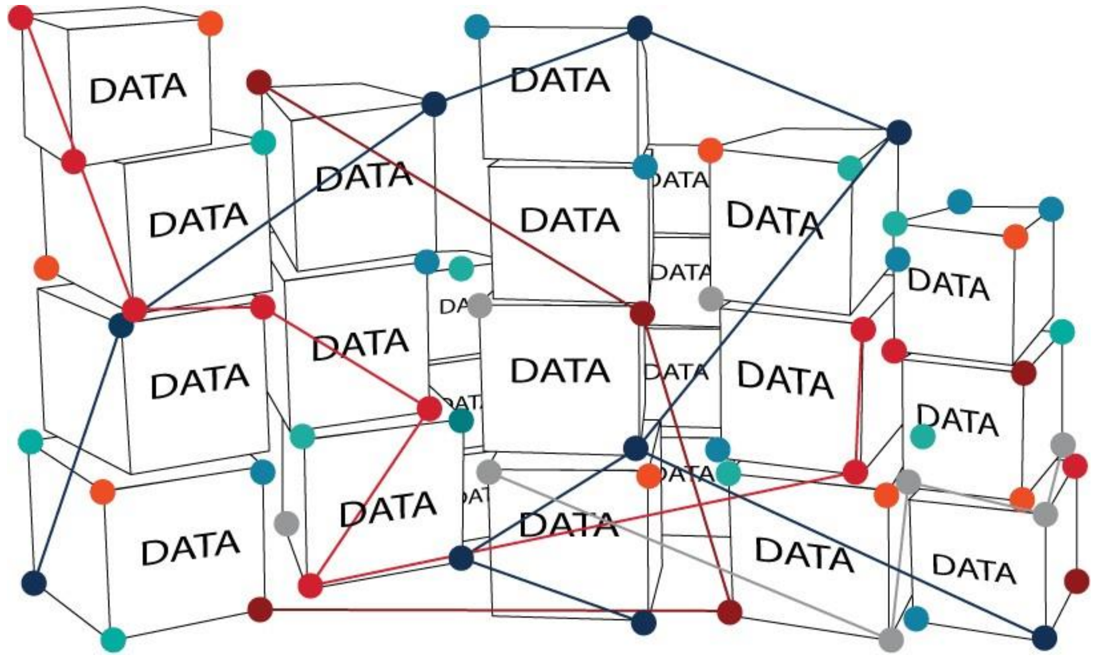
- Graph databases are generally straightforward in how they’re structured though. They primarily are composed of two components:
- The Node
  - This is the actual piece of data itself.
  - It can be the number of viewers of a youtube video, the number of people who have read a tweet, or it could even be basic information such as people’s names, addresses, and so forth.
- The Edge
  - This explains the actual relationship between two nodes.
  - Interestingly enough, edges can also have their own pieces of information, such as the nature of the relation between two nodes. Similarly, edges might also have directions describing the flow of said data.

Translating NoSQL Knowledge to Graphs

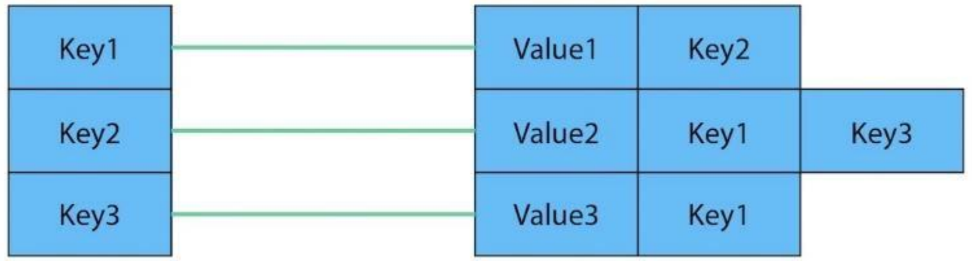
- With the advent of the NoSQL movement, businesses of all sizes have a variety of modern options from which to build solutions relevant to their use cases.
  - Calculating average income? Ask **a relational database**.
  - Building a shopping cart? Use a **key-value Store**.
  - Storing structured product information? Store as a **document**.
  - Describing how a user got from point A to point B? Follow a **graph**.
- Examples of Graph Databases
  - Neo4j, ArangoDB

Graph-Based NoSQL

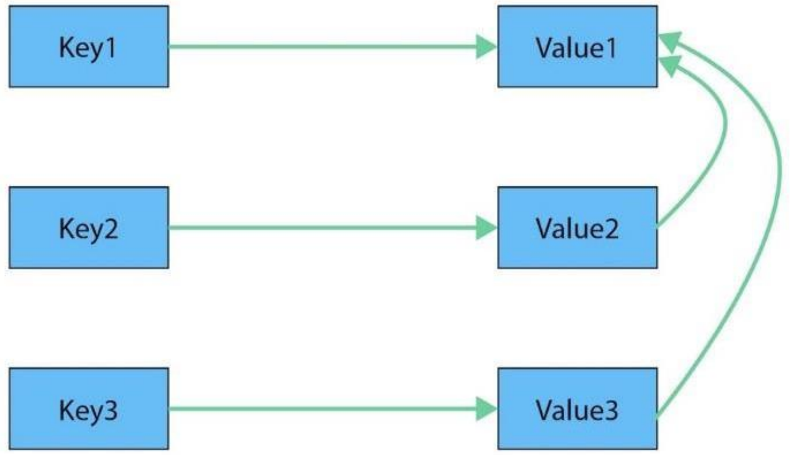

Friends
Person
LivesIn(address,...,...)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)



Key-Value vs. Graph: Data Model Differences

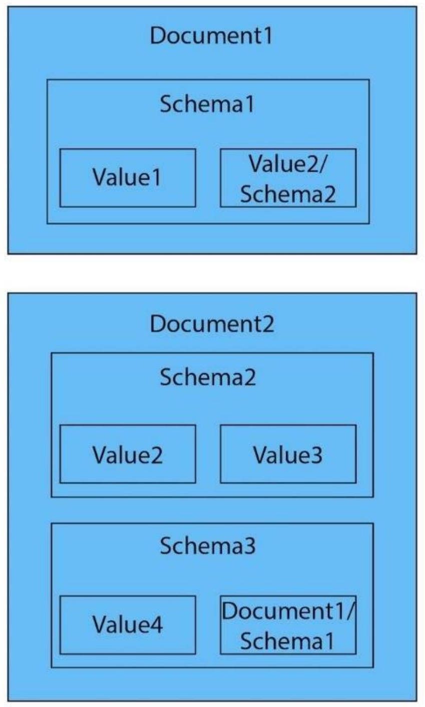


Key-Value Model

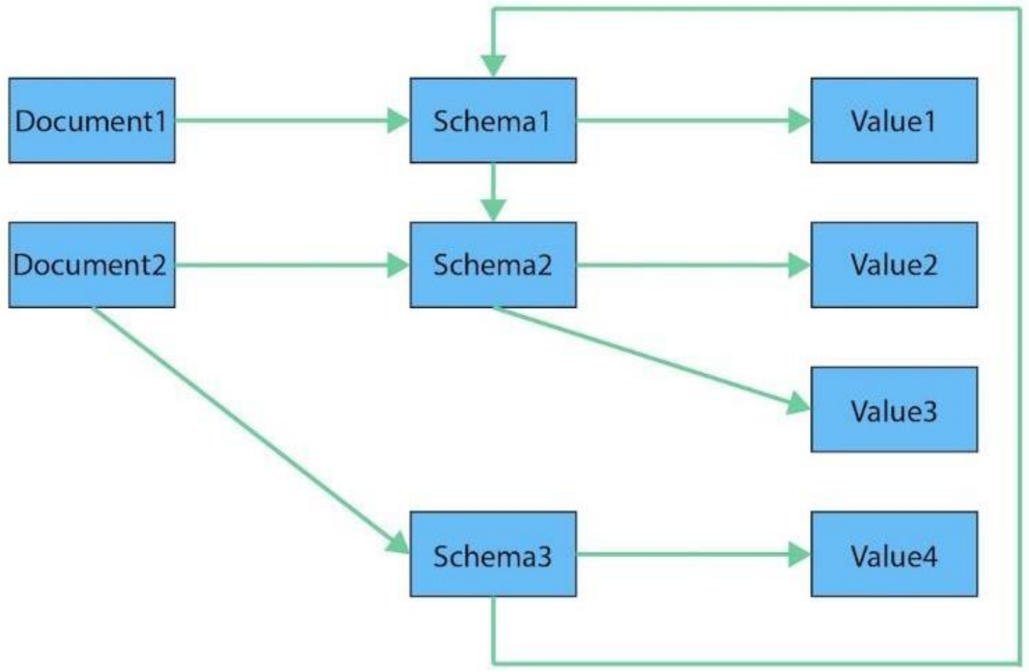


Key-Value as Graph

Document vs. Graph: Data Model Differences



Document Model



Document as Graph

References

- https://www.guru99.com/nosql-tutorial.html
- https://redis.com/nosql/key-value-databases/
- https://www.mongodb.com/scale/types-of-nosql-databases
- https://www.kdnuggets.com/2021/02/understanding-nosql-database- types-column-oriented-databases.html
- https://ravendb.net/articles/nosql-document-oriented-databases-detailed- overview