NOSQL

Aggregate data models

- A data model is the model through which we perceive and manipulate our data.

- For people using a database, the data model describe how we interact with the data in the database.

- Data model : the model by which the database organizes data.

- The dominant data model of the last couple of decades is the relational data model, which is best visualized as a set of tables.

- Each table has rows, with each row representing some entity of interest.

- We describe this entity through columns, each having a single value(atomic value).

- A column may refer to another row in the different table, which constitutes a relationship between those entities. (Foreign Key)

Aggregates

- The relational model takes the information that we want to store and divides it into tuples (rows).

- A tuple is a limited data structure: It captures a set of values, so we cannot nest one tuple within another to get nested records, nor can we put a list of values or tuples within another.

- Aggregate orientation takes a different approach.

- We often want to operate on data in units that have a more complex structure than a set of tuples.

- key-value, document, and column-family databases all make use of this more complex record

- However, there is no common term for this complex record; here (we) use the term " **aggregate**."

- **aggregate is a collection of related objects that we wish** to treat as a unit.

- In particular, it is a unit for data manipulation and management of consistency.

- Aggregates are also often easier for application programmers to work with, since they often manipulate data through aggregate structures.

NoSQL

NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways.

It is designed for distributed data stores where very large scale of data storing needs.

For example Google or Facebook which collects terabits of data every day for their users.

These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

RDBMS

- Structured and organized data
- Structured query language (SQL)
- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition

  Language
- Tight Consistency

NoSQL

- Stands for Not Only SQL
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- All NoSQL offerings relax one or more of the ACID properties (will talk about the CAP theorem)

NoSQL Categories

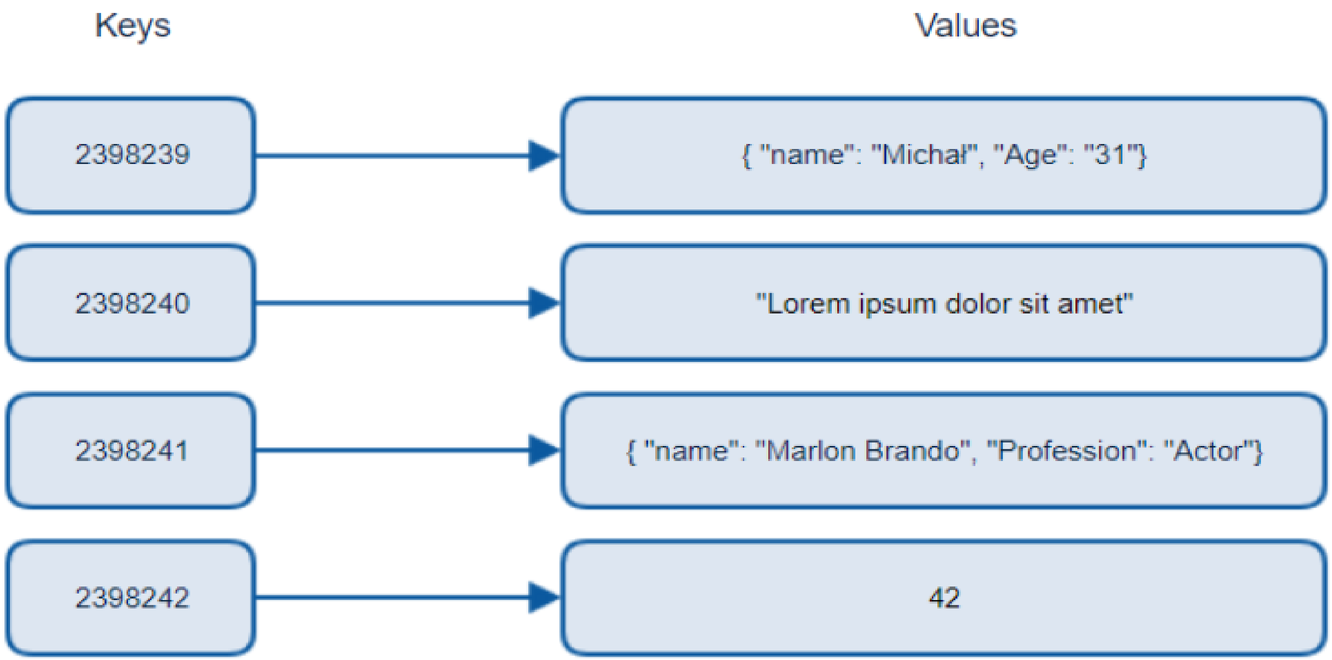
- There are four general types (most common categories) of NoSQL databases.
- Each of these categories has its own specific attributes and limitations.
- There is not a single solutions which is better than all the others, however there are some databases that are better to solve specific problems.

  - Key-value stores
  - Column-oriented
  - Document oriented
  - Graph database

Key-value stores

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data (Based on Amazon's Dynamo paper).
- Key value stores allow developer to **store schema-less data**.
- In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON etc.
- For example a key-value pair might consist of a key like "Name" that is associated with a value like "Robin".
- **Use Cases :**
  - Key-Values stores would work well for shopping cart contents.
  - Customized ad delivery to users based on their data profile
- Example of Key-value store DataBase : Redis, DynamoDB, Riak. etc.

- The application is developed on queries that are based on keys.



Keys	Values
2398239	{ "name": "Michal", "Age": "31"}
2398240	"Lorem ipsum dolor sit amet"
2398241	{ "name": "Marlon Brando", "Profession": "Actor"}
2398242	42

Implements a **hash table** to store unique keys along with the pointers to the corresponding data values.

Have **no query language** but they do provide a way to **add** and **remove** key-value pairs. Search only based on Key

Column-oriented databases

- Most databases have a row as a unit of storage which, in particular, helps write performance.

- However, there are many scenarios where writes are rare, but we often need to read a few columns of many rows at once for some analytics on those few columns.

- In this situation, it’s **better to store groups of columns** for all rows as the basic storage unit—which is why these databases are called column stores.

- Example of Column-oriented databases : BigTable, Hbase, Cassandra etc.

To get a particular customer's name from Figure 2.5 we could do something like get('1234', 'name').

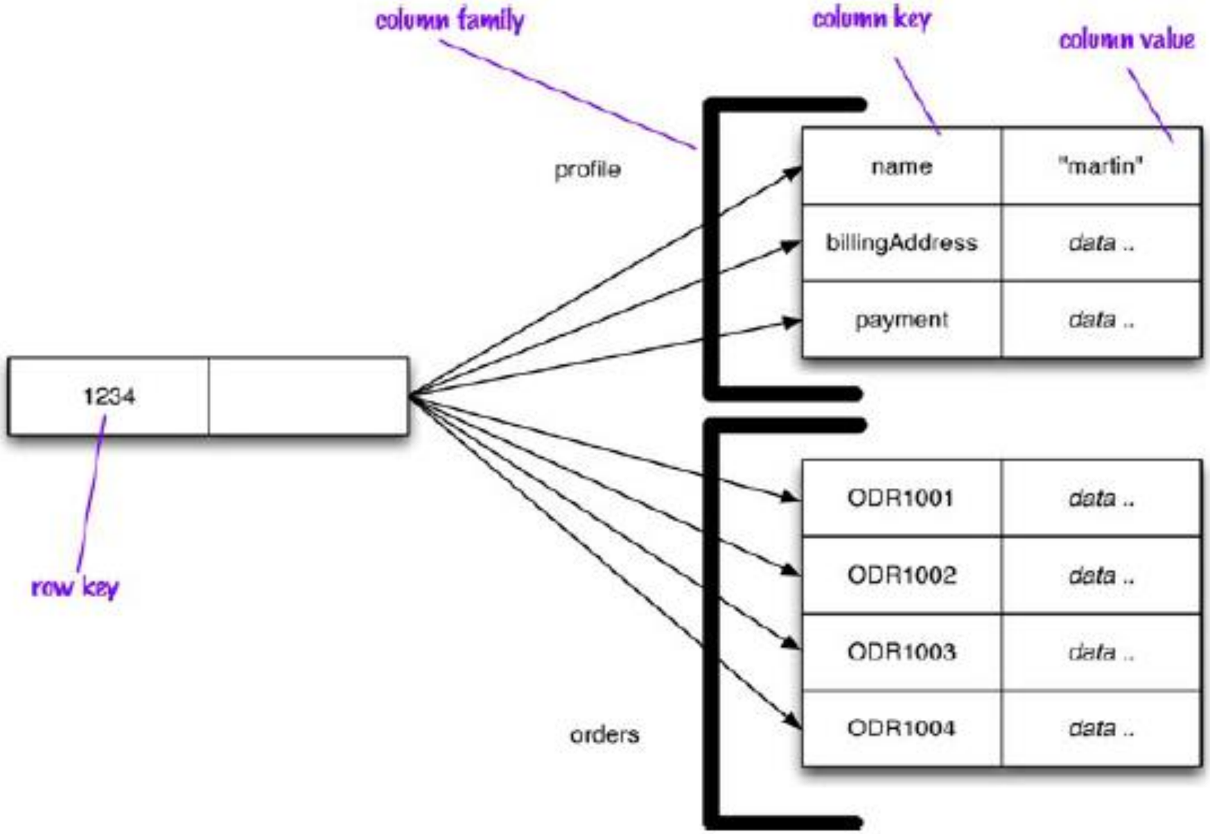
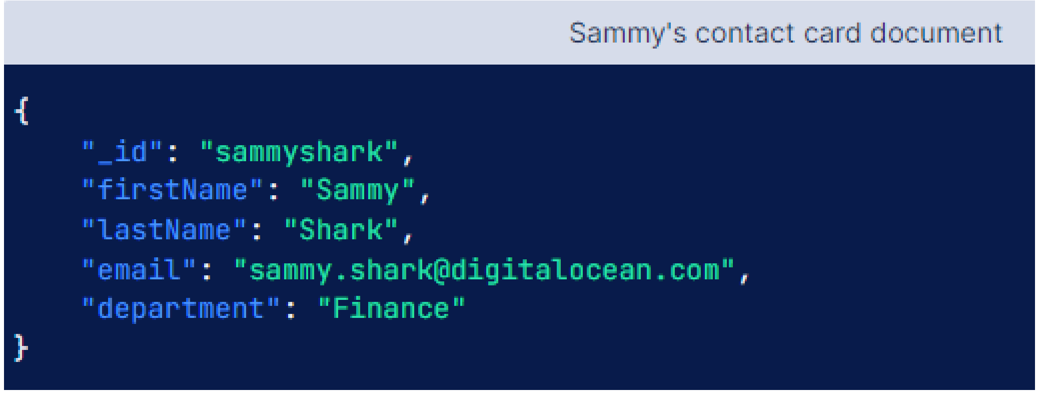


Figure 2.5. Representing customer information in a column-family structure

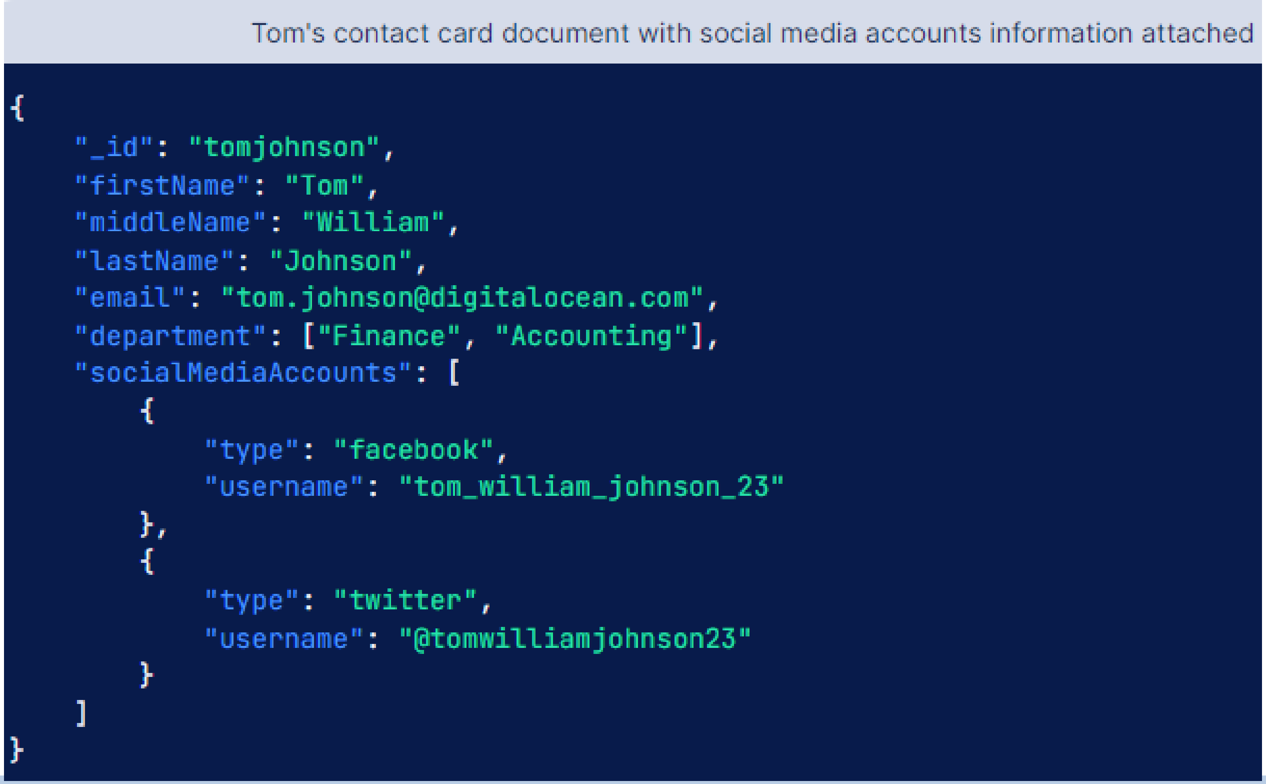
Document Oriented databases

- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- Example of Document Oriented databases : MongoDB, CouchDB etc.



Sammy's contact card document

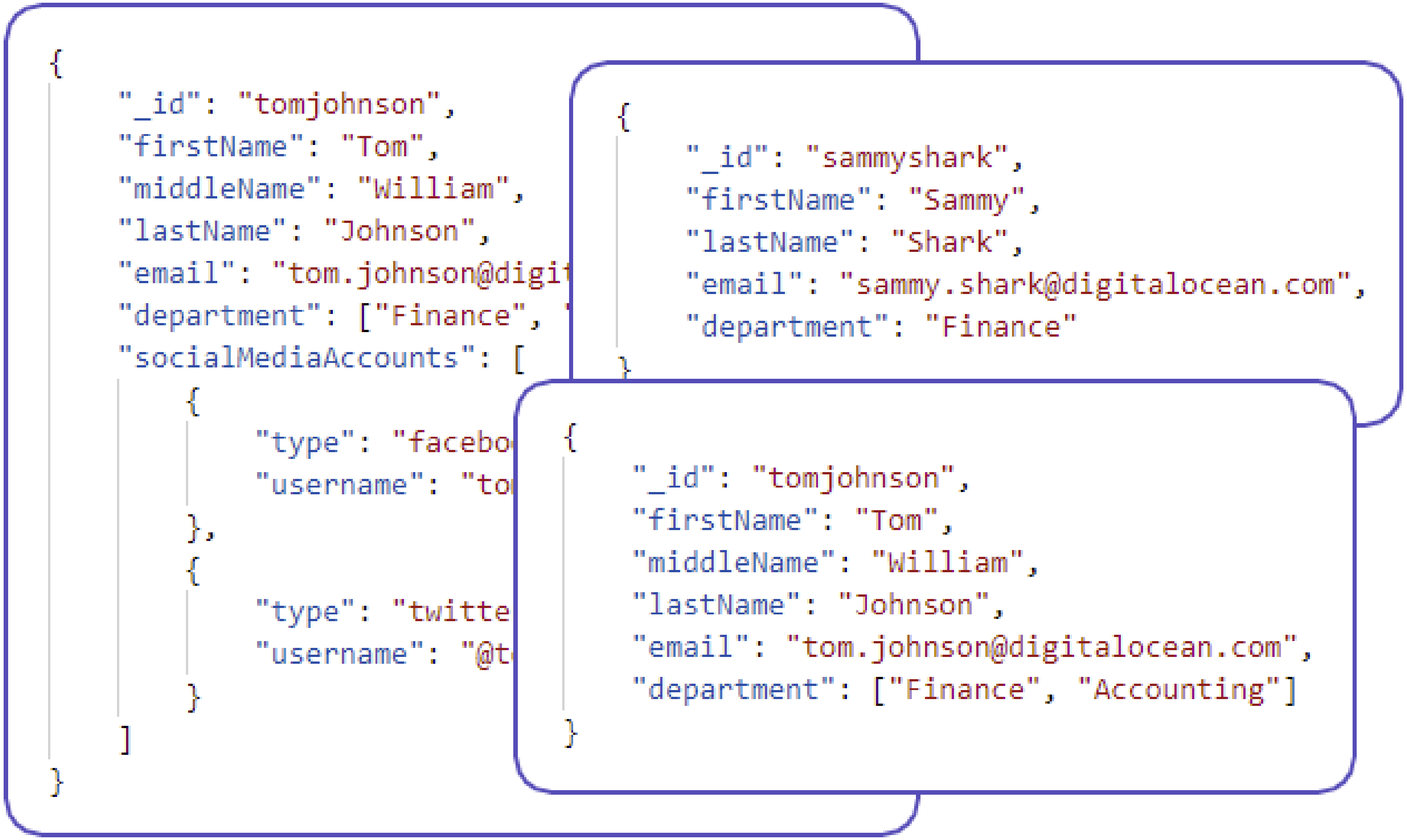
```
{
    "_id": "sammyshark",
    "firstName": "Sammy",
    "lastName": "Shark",
    "email": "sammy.shark@digitalocean.com",
    "department": "Finance"
}
```



Tom's contact card document with social media accounts information attached

```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"],
    "socialMediaAccounts": [
        {
            "type": "facebook",
            "username": "tom_william_johnson_23"
        },
        {
            "type": "twitter",
            "username": "@tomwilliamjohnson23"
        }
    ]
}
```

Collection



```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digit
    "department": ["Finance", '
    "socialMediaAccounts": [
        {
            "type": "facebo
            "username": "to
        },
        {
            "type": "twitte
            "username": "@t
        }
    ]
}
```

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

```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"]
}
```

Graph databases

- A graph database stores data in a graph.
- A graph database is a collection of nodes and edges
- Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.
- Every node and edge are defined by a unique identifier.
- Each node knows its adjacent nodes.
- Example of Graph databases : OrientDB, Neo4J, Titan. etc.

- Graph databases are an odd fish in the NoSQL pond.

- Graph databases are motivated by a different frustration with relational databases and thus have an **opposite model—small records with complex interconnections,** something like Figure below.

- We refer to a graph data structure of nodes connected by edges.

- In Figure we have a web of information whose nodes are very small (nothing more than a name) but there is a rich structure of interconnections between them.

- With this structure, we can ask questions such as “ find the books in the Databases category that are written by someone whom a friend of mine likes.”