Introduction to NoSQL Database Management Systems

A. Big Data and NoSQL

- **Big Data** is used to label large volumes of data that push the limits of conventional software. This data is usually unstructured or semi-structured and may originate from a wide variety of sources: social media postings, emails, electronic archives with multimedia content, etc.
- Big data generally refers to a set of data that displays the characteristics of high-volume, high-velocity, and high-variety (the 3 Vs) information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making.
- **Volume –** the quantity of data to be stored, is a key characteristic of big data. The storage capacities associated with big data are extremely large.
- **Velocity –** another key characteristic of big data. This refers to the rate at which new data enters the system as well as the rate at which the data must be processed.
- **Variety –** In a big data context, it refers to the vast array of formats and structures in which the data may be captured.
- **NoSQL database management systems (NoSQL DBMS) –** A new generation of database management systems that is not based on the traditional relational database model (SQL) which met the two criteria:
  1. The data is not stored in tables. (Like SQL)
  2. The database language is not SQL.
- NoSQL is also sometimes interpreted as 'Not only SQL' to express that other technologies (besides relational DBMS) are often used for storing Big Data and in massively distributed web applications. NoSQL technologies are especially necessary if the web service requires high availability (i.e., data in social media like Facebook).
- NoSQL excels in its ease of use, scalability, resilience, and availability characteristics. Instead of joining tables of normalized data, NoSQL stores unstructured or semi-structured data, often in key-value pairs or JSON documents.

B. NoSQL Data Model

- **Key-value (KV) database:**
  - It is the simplest of the NoSQL data models.
  - It stores data as a collection of key-value pairs.
  - The key acts as an identifier for the value. The value can be anything such as text, an XML document, or an image.
  - The database does not attempt to understand the contents of the value
  - The database simply stores whatever value is provided for the key.
  - It is the job of the applications that use the data to understand the meaning of the data in the value component.
  - There are no foreign keys. Relationships cannot be tracked among keys at all.
  - Greatly simplifies the work that the DBMS must perform, making KV databases extremely fast and scalable for basic processing.

Key	Value
10010	"Lname Banaag Fname Michelle Initial A Phone 09358891234 Balance 0"

Table 1. *Key-value pair*

- **Column-oriented databases:**
  - can refer to traditional, relational database technologies that use column-centric storage instead of row-centric storage
  - More efficient for optimizing read operations to store the data in relational tables, not per row, but per column.
  - All columns in one row are rarely needed at once, but there are groups of columns that are often read together.
  - To optimize access, it is useful to structure the data in such groups of columns—column families—as storage units.

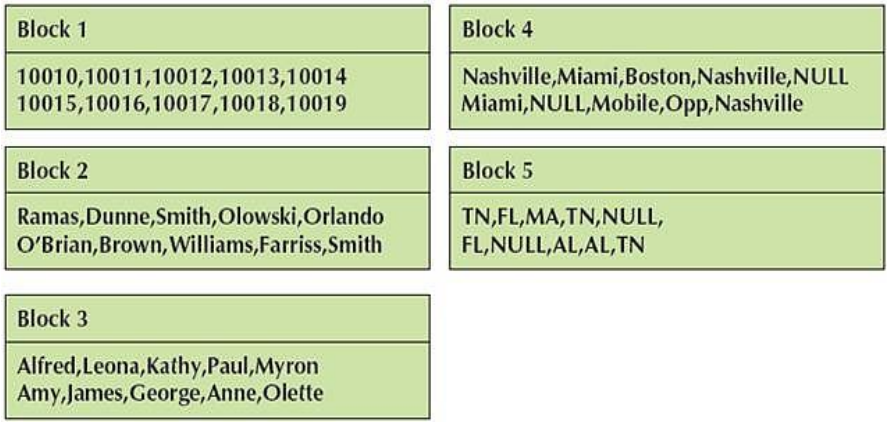

Figure 1. Column-oriented database

- **Graph databases:**
  - Graph databases are based on graph theory and represent data through nodes, edges, and properties.
  - A node is similar to an instance of an entity in the relational model. Edges are the relationships between nodes. Both nodes and edges can have properties, which are attributes that describe the corresponding node or edge.
  - Graph databases excel at tracking data that are highly interrelated, such as social media data (Facebook, Instagram, Twitter).

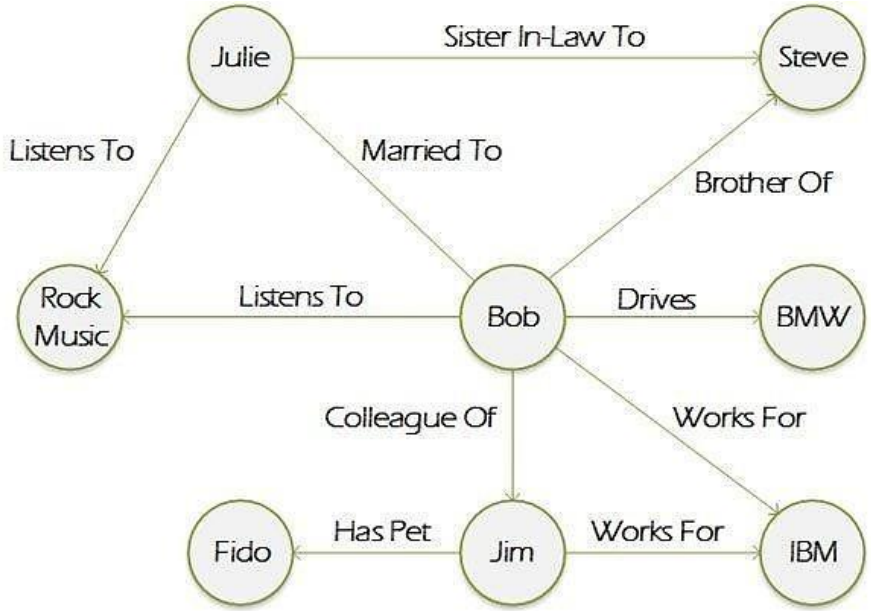

Figure 2. *Graph database*

- **Document-Oriented databases**
  - Document-oriented databases are conceptually similar to key-value databases, and they can almost be considered a subtype of KV databases.
  - A document database is a NoSQL database that stores data in tagged documents in key-value pairs. Unlike a KV database, where the value component can contain any type of data, a document database always stores a document in the value component.
  - The document can be in any encoded format, such as XML, JSON (JavaScript Object Notation), or BSON (Binary JSON).
  - Tags are named portions of a document.
  - Although all documents have tags, not all documents are required to have the same tags, so each document can have its own structure.
  - Tags in a document database are extremely important because they are the basis for most of the additional capabilities that document databases have over KV databases.

```
{
  id: 1,
   Name: "Justin York",
   Age: 27,
   Gender: "Male",
   Address:
   {
          City: "Makati"
          Country: "Philippines
   },
   Contact:
   {
          Mobile: "+639358491124"
          Email: “justineyork@gmail.com”
   }
}
```

Example of a Customer document written in JSON format

C. Document-Oriented Database

- **MongoDB** is a cross-platform, open-source, document-oriented database that provides high performance, high availability, automatic, and easy scalability. It is highly optimized for JSON. It stores data in flexible JSON documents, which means the columns may vary from document to document, and the data structure may be reformed over time.

SQL	MongoDB
Database	Database
Tables	Collections
Columns	Fields
Rows	Documents

Table 2. *SQL vs. MongoDB*

- **Database** – MongoDB groups collections into databases. A single instance of MongoDB can host several databases, each of which can be thought of as completely independent.
- **Collections** – can be thought of as a table with a dynamic schema.
- **Document** – is the basic unit of data for MongoDB, roughly equivalent to a row in a relational database management system (but much more expressive).
- Every document has a special key, "_id", that is unique across the document's collection.

D. Basic Operations in Document-Oriented Database

If there is no existing database, the given command below will automatically **create a new database**. In *MongoDB,* you don't need to create a database manually because *MongoDB* will create it automatically when you save the value into the defined collection for the first time.

```
> use FirstMongoDB
switched to db FirstMongoDB
```

To **check the currently selected database**, use the **db** command:

```
> db
FirstMongoDB
```

To **query the database list**, use the **show dbs** command:

```
> show dbs
admin    0.000GB
config   0.000GB
db       0.000GB
db_test  0.000GB
local    0.000GB
nosql    0.000GB
>
```

FirstMongoDB is currently not on the list because it does not contain any documents. Use this syntax to **create a document** inside the *FirstMongoDB* database.

```
> db.student.insertMany([ {name: "John Smith"}, {name: "John Cedrick"} ])
```

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5fab4e024d65454f319d8760"),
        ObjectId("5fab4e024d65454f319d8761")
    ]
}
```

Now if we **check the database list** using the **show dbs** command:

```
> show dbs
FirstMongoDB  0.000GB
admin         0.000GB
config        0.000GB
db            0.000GB
db_test       0.000GB
local         0.000GB
nosql         0.000GB
>
```

To **query the newly created documents**, use this syntax:

```
> db.student.find()
{ "_id" : ObjectId("5fab4e024d65454f319d8760"),
  "name" : "John Smith" }
{ "_id" : ObjectId("5fab4e024d65454f319d8761"),
  "name" : "John Cedrick" }
```

To **query the database collection lists**, use this syntax:

```
> show collections
student
```

Note that after creating a **document** from the previous query, the *student collection* is automatically created.

To **manually create a collection** in the database, use this syntax:

```
> db.createCollection("Teacher")
{ "ok" : 1 }
```

To **remove or drop a document in the database**, use this syntax:

```
> db.student.remove({name: "John Cedrick"})
WriteResult({ "nRemoved" : 1 })
```

To **remove or drop a collection** in the database, use this syntax:

```
> db.student.drop()
true
```

To **delete the currently selected database**, use this syntax:

```
> db.dropDatabase()
{ "dropped" : "FirstMongoDB", "ok" : 1 }
```

REFERENCES

Coronel, C. and Morris, S. (2018). *Database systems design, implementation, & management* (13th ed.). Cengage Learning.

Chodorow, K., Brazil, E., and Bradshaw, S. (2019). *MongoDB: The definitive guide* (3rd ed.). O'Reilly Media, Inc.