NoSQL and MongoDB

Session 4

Database for Big Data Analytics

Siti Mariyah, Ph.D.
Poilteknik Statistika STIS

Email: sitimariyah@stis.ac.id
Github: https://github.com/diahnuri

Session 4 – NoSQL and MongoDB

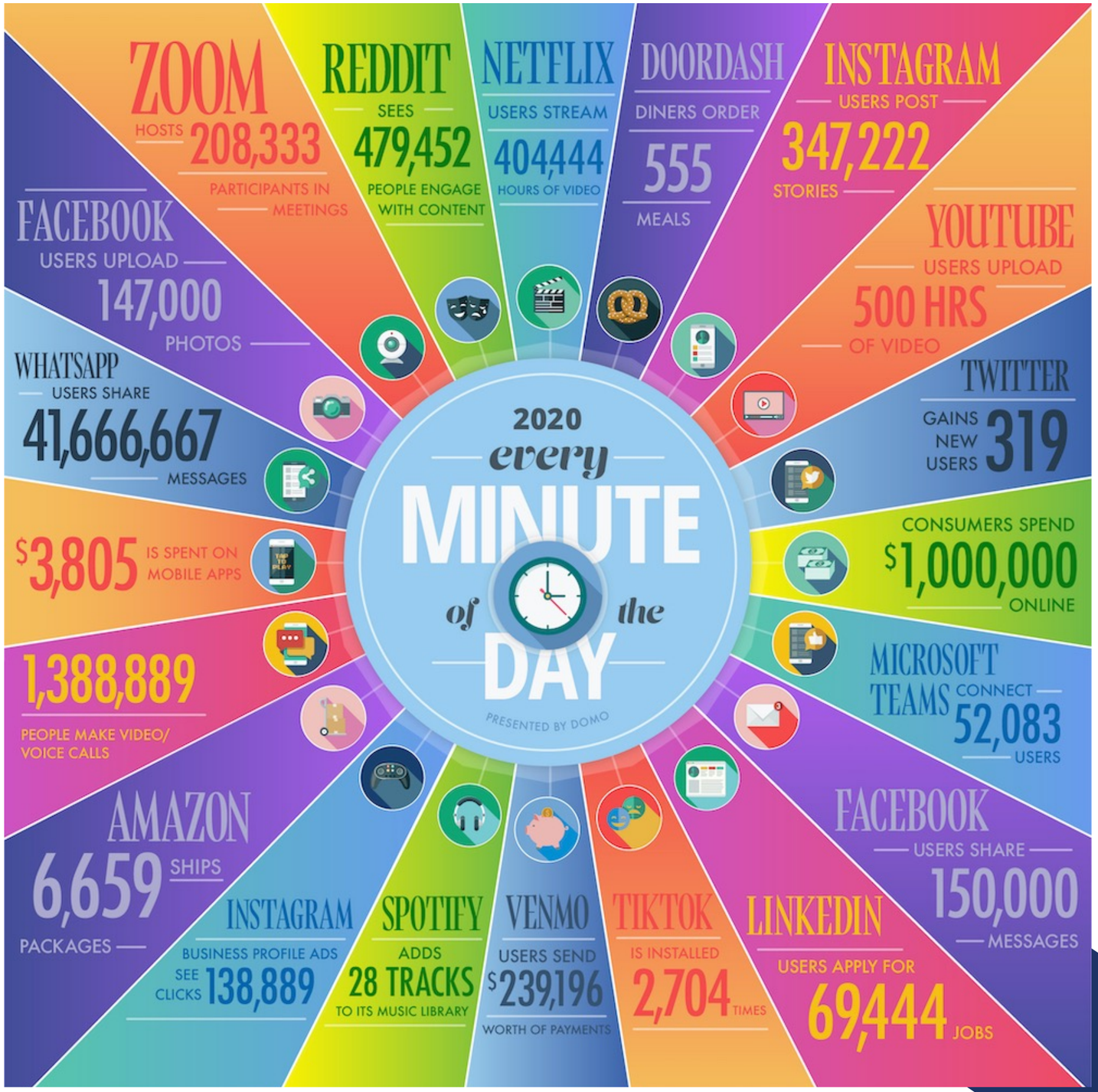
- Goal: Be able to understand the concept of NoSQL and to apply NoSQL in MongoBD
- Content:
  - Database for Big Data Technology
  - Introduction of NoSQL
  - NoSQL Data Model
  - JSON
  - Application of NoSQL in MongoDB
- Hands-on:
  - Install MongoDB
  - Understand JSON data format
  - Create database in MongoDB
  - Data manipulation in MongoDB

Web 2.0

- Web 2.0, the era of social media (Facebook, Twitter, etc.), and e-commerce (Amazon), with its massive data requirements.
- The amount of data, data structures, application scale, and how applications are developed have all changed dramatically.
- This requires machines with greater capabilities.

DATA NEVER SLEEP 8.0

https://www.domo.com/learn/infographic/data-never-sleeps-8



Database for Big Data Technology

- To handle the immense volume, velocity, and variety of big data, organizations use specialized database technologies designed for distributed processing and flexible data models.
- While relational databases are suitable for structured data, big data often requires NoSQL databases, data warehouses, and data lakes to manage diverse data types at scale.

Data Warehouses

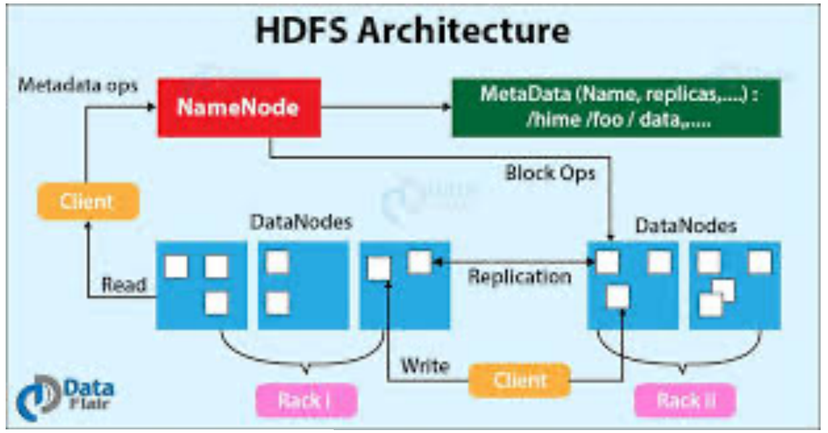
- These tools are built for analytical processing and managing massive datasets in a centralized location.

- Data Warehouse aggregate data from multiple sources and prepare it for business intelligence and analytics
- Amazon Redshift: A cloud-based, managed data warehouse that uses columnar storage and parallel processing for fast query performance.
- Azure Synapse Analytics: A cloud-native analytics service that integrates big data and data warehousing capabilities.
- Snowflake: A cloud-based data platform that provides a single solution for data warehousing, data lakes, and data engineering.

Data Lakes

- Designed to store massive amounts of raw, multi-structured data without a fixed schema, data lakes are ideal for advanced analytics, machine learning, and AI.

- Hadoop Distributed File System (HDFS): A core component of the Apache Hadoop framework that splits and stores large files across a cluster of commodity servers.

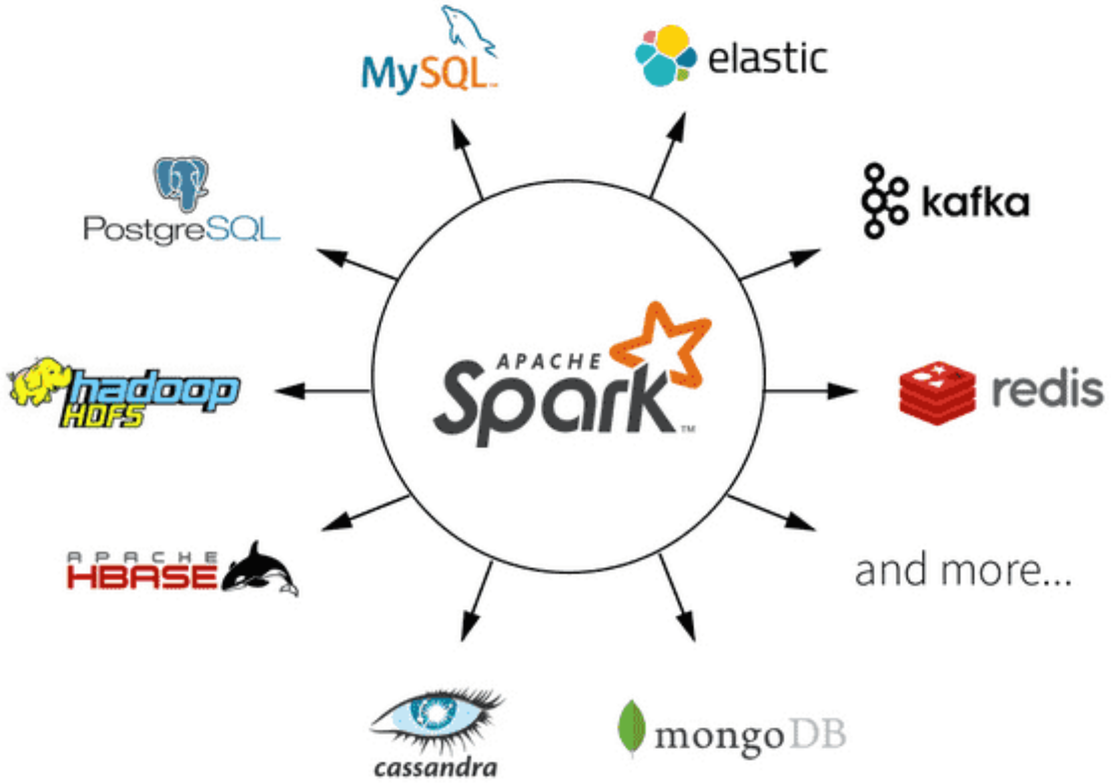


- Delta Lake: An open-source storage layer that sits on a data lake, adding reliability, schema enforcement, and ACID transactions.

DELTA LAKE

Processing Frameworks

- While not databases themselves, these frameworks are essential big data tools that often interact with the storage layer.
- Apache Spark: A fast, in-memory processing framework that can run on clusters like Hadoop. It supports batch and stream processing, interactive queries, and machine learning.
- Apache Kafka: A distributed event streaming platform used for building real-time data pipelines and streaming analytics.
- Apache Hive: An SQL-based data warehousing system built on top of Hadoop for data summarization and querying.



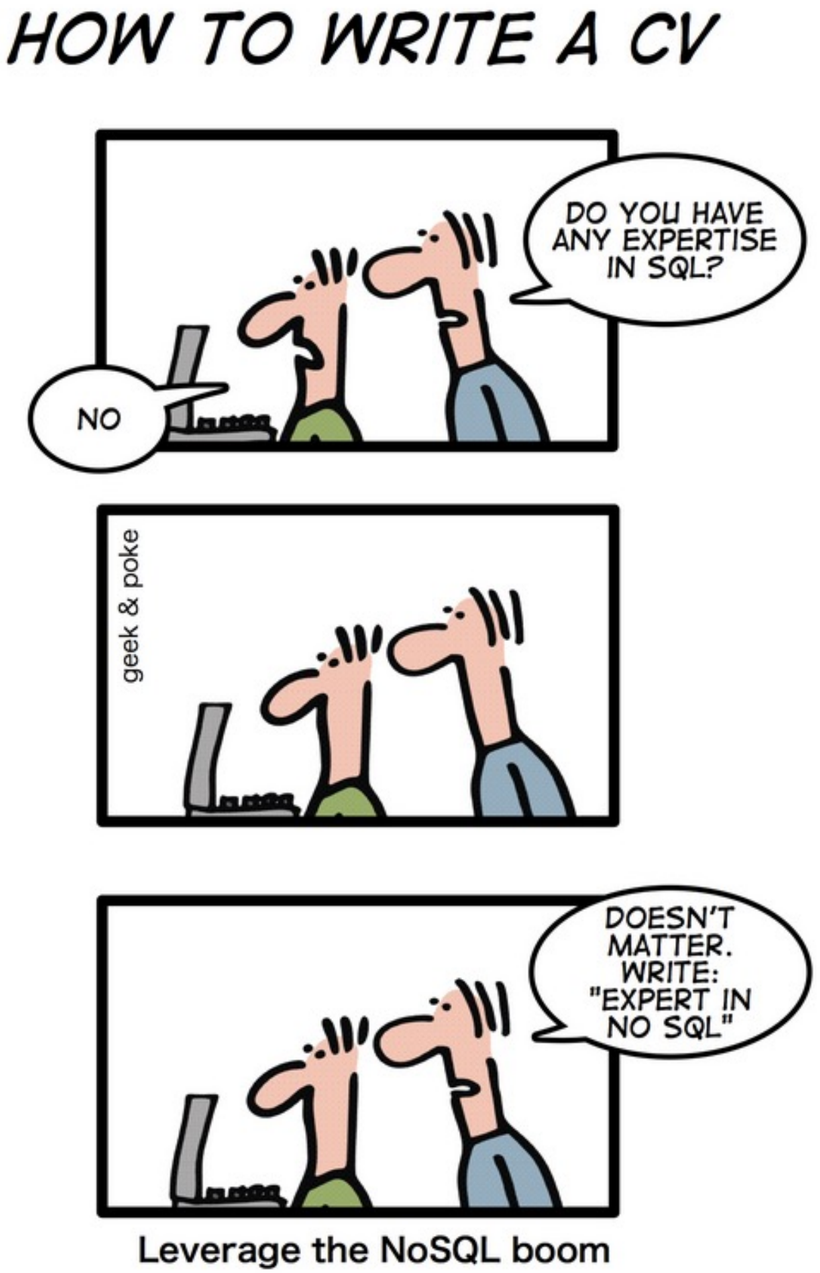
Introduction to NoSQL

- NOSQL
  - Not only SQL
  - SQL systems offer many features (not everyone will use) and restrictive
- Most NOSQL systems are distributed databases or distributed storage systems
  - Focus on semi-structured data storage, high performance, availability, data replication, and scalability
- NOSQL systems focus on storage of “big data”

- Typical applications that use NOSQL
  - Social media
  - Web links
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email

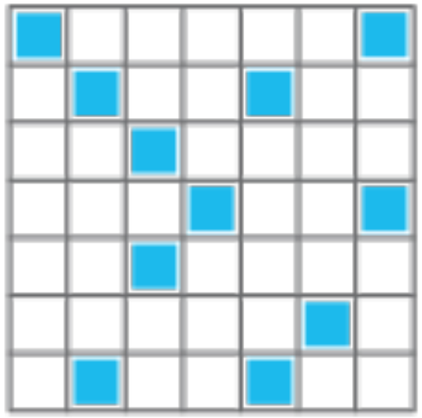
Introduction

- BigTable
  - Google’s proprietary NOSQL system
  - Column-based or wide column store
- DynamoDB (Amazon)
  - Key-value data store
- Cassandra (Facebook)
  - Uses concepts from both key-value store and column-based systems



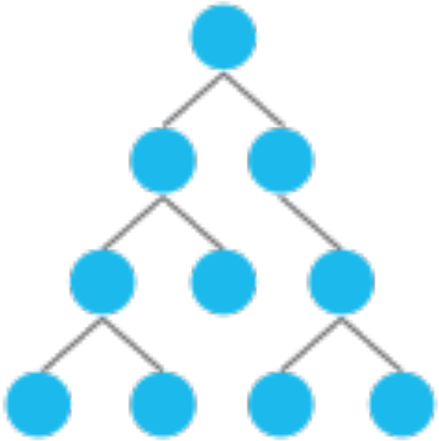
https://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

NoSQL Data Model



Column-Family

Graph

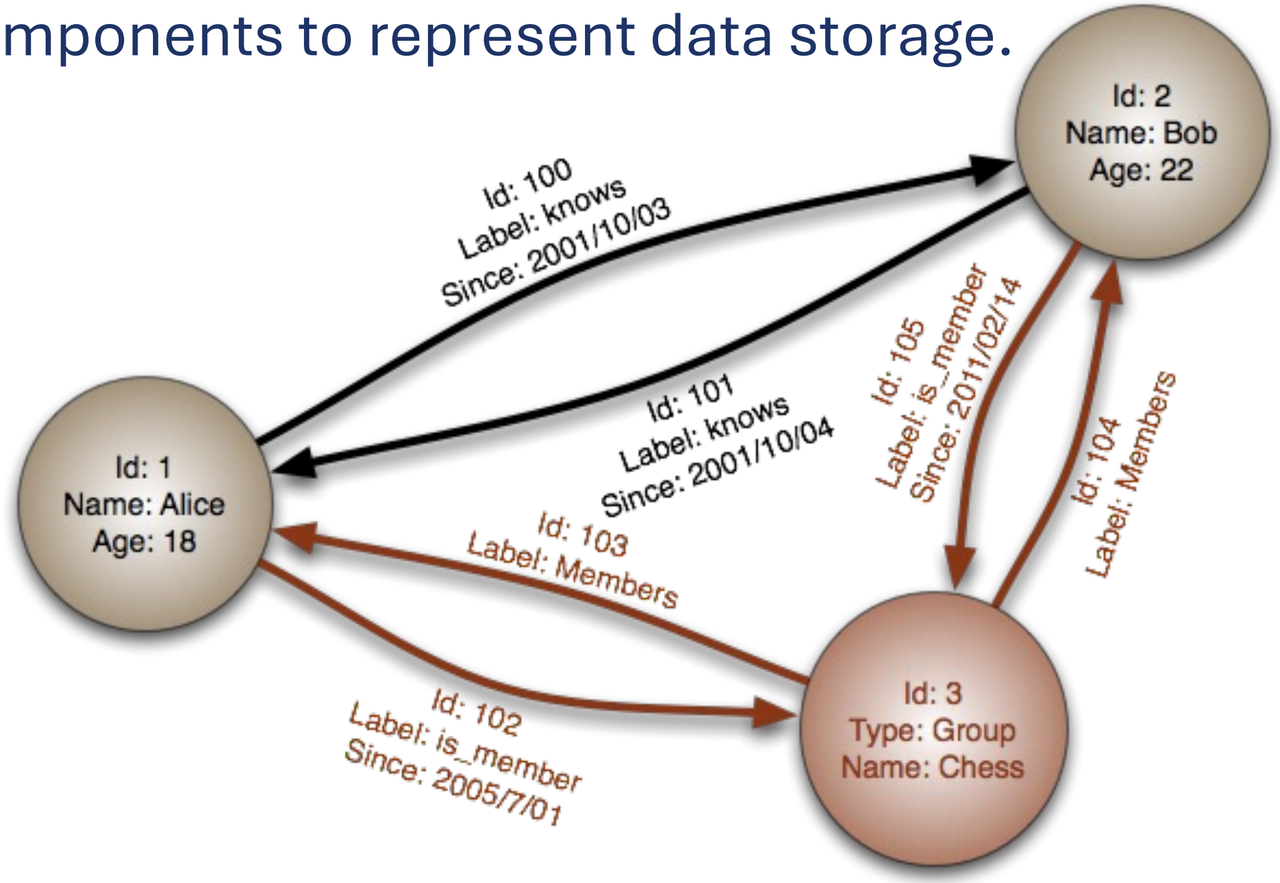


Document

Key-Value

Graph Database

A database that uses a graph data structure that has node, edge and property components to represent data storage.

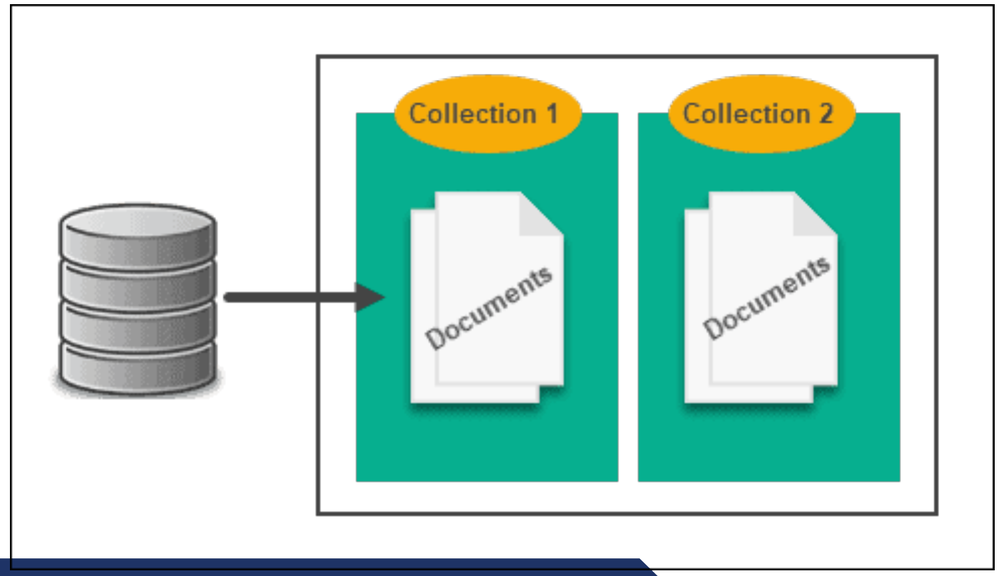


RedisGraph

neo4j

Document Store

- Documents are stored in some standard format or encoding (e.g., XML, JSON, PDF, or Office Document)
- Commonly referred to as Binary Large Objects (BLOBs)
- Documents can be indexed
- Allows document storage to outperform traditional file systems



mongoDB®

CouchDB
relax

Key Value Store

- Keys are mapped to (possibly) more complex values (e.g., lists).
- Keys can be stored in hash tables and can be easily distributed.
- Storage typically supports CRUD operations and no joins or aggregate functions.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Columnar Database (Column-Family)

- Based on the Google BigTable paper
- A combination of RDBMS and Key-Value storage
- Similar to an RDBMS but capable of handling semi-structured data

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

VERTICA

APACHE HBASE

When Do We Use NoSQL?

- The development pace with NoSQL databases can be significantly faster than with SQL databases.
- Different data structures are easier to handle and develop with NoSQL databases.
- The volume of data in many applications cannot be handled affordably by SQL databases.

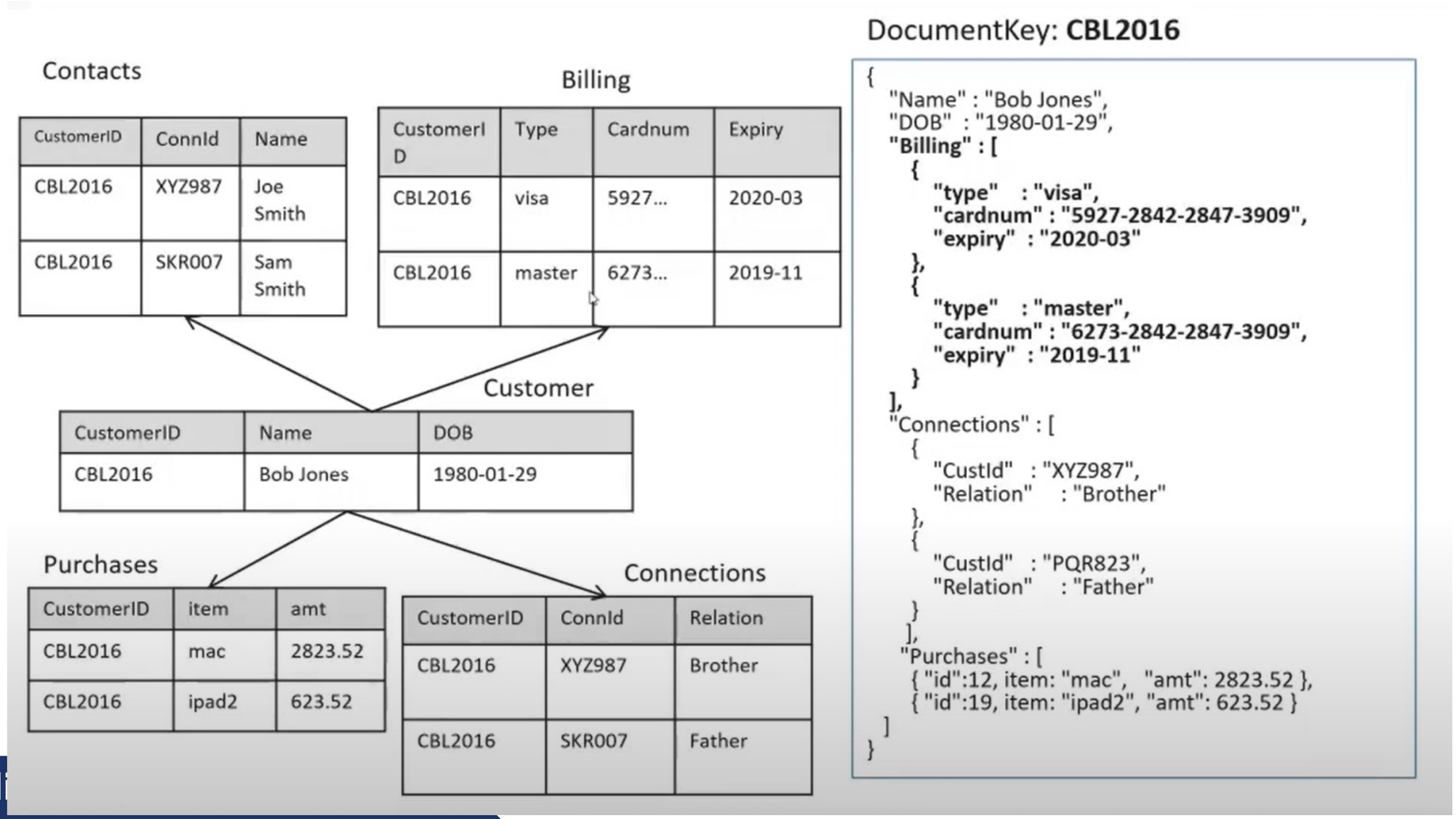
JSON

- JSON (JavaScript Object Notation) is a lightweight data interchange format.
- The JSON format is not only easy to read and write by humans, but also easy to parse and generate by machines.
- JSON data is format-independent but uses the conventions of the C family of programming languages, such as C, C++, Java, JavaScript, Perl, Python, and others.
- It is widely used to transfer data from servers to applications, web services, APIs, and NoSQL database formats.

JSON Example

```
{
  "book": [
    {
      "id":"01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id":"07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }]}
```

Database Format in JSON

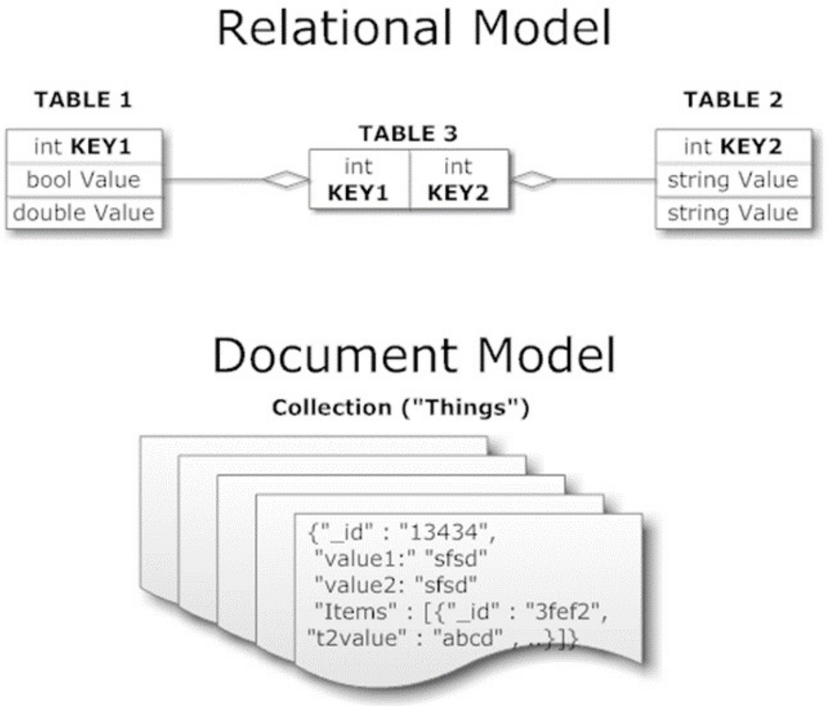


Introduction to MongoDB

MongoDB

- MongoDB is a document database designed for ease of development and scaling
- Leading open-source document database and NoSQL database.
- MongoDB is written in C++.

Document-based Database




```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

Working with data doesn't need to be hard

Our guiding principle is to help developers solve their data challenges.
Here's what you can do with MongoDB.

Build Faster

Ship and iterate 3–5x faster with our flexible document data model and a unified query interface for any use case.

Scale Further

Whether it's your first customer or 20 million users around the world, meet your performance SLAs in any environment.

Sleep Better

Easily ensure high availability, protect data integrity, and meet the security and compliance standards for your mission-critical workloads.

https://www.mongodb.com/

Used by Many Organization

https://www.mongodb.com/

Features

High Performance

Rich Query Language

High Availabilty

Horizontal Scalability

Model Data

- Document-Based
- Documents in BSON format, consisting of field and value pairs
- (max 16 MB)
- Each document is stored in a collection
- Collections
- have indexes
- Like relational database tables.
- Documents do not have to have a uniform structure

CRUD

How to Connect to A Cluster?

Install MongoDB database tools on Mac

- brew tap mongodb/brew
- brew install mongodb-database-tools

Cluster Identity

- Database User Username: Cluster38685
- Database User Password: hzMLwoEHaToVQP1Y
- The connection string: mongodb+srv://cluster38685.gi0czx5.mongodb.net
- Connecting to:

mongosh "mongodb+srv://cluster38685.**gi0czx5.mongodb.net**/mpd" –username Cluster38685

Password: hzMLwoEHaToVQP1Y

Create Database and Collection

- Switch to (or create) a new database called "mpd"

  use mpd

- Create empty collections (optional, auto-created on insert)

  db.createCollection("subscribers");
  db.createCollection("towers");
  db.createCollection("cdr_events");
  db.createCollection("tourism_events");

Import Data to A Cluster

```
mongoimport –uri "mongodb+srv://cluster38685:
hzMLwoEHaToVQP1Y@cluster38685.giOczx5.mongodb.net/mpd" –
collection cdr_events –type json –file cdr_events.jsonl –jsonArray
```

Import Data to A Cluster

```
mongoimport –uri
"mongodb+srv://cluster38685:hzMLwoEHaToVQP1Y@cluster38685.gi0czx5.mongodb.net/mpd" –collection cdr_events –file "/Users/sitimariyah/Documents/Regional Hub on Big Data and Data Science for the Asia and the Pacific/MPD Short Course Database for Big Data Analytics/mpd_db_course_starter_kit/mongo/cdr_events.jsonl"
```

Document Insert

- To insert data into a MongoDB collection, we can use the MongoDB insert() or save() commands.
- The save() command will be discussed in the document update subsection.

```
db.subscribers.insertOne({
  subscriber_id: 1,
  home_regency: "Denpasar",
  home_tower_id: 101,
  sim_type: "prepaid"
})
```

```
db.towers.insertMany([
  { tower_id: 101, regency: "Denpasar",
latitude: -8.65, longitude: 115.22 },
  { tower_id: 102, regency: "Gianyar", latitude: -
8.55, longitude: 115.33 }
])
```

Document INSERT

Insert a call detail record (CDR)

```
db.cdr_events.insertOne({
  event_id: 5001,
  subscriber_id: 1,
  tower_id: 101,
  event_type: "call",
  event_ts: Date("2025-09-
12T10:30:00Z")
});
```

Insert a tourism event

```
db.tourism_events.insertOne({
  event_id: "E001",
  name: "Ubud Arts Fair",
  regency: "Gianyar",
  start_date: ISODate("2025-09-16T00:00:00Z"),
  end_date: ISODate("2025-09-24T23:59:59Z")
});
```

Verify INSERT

```
// Show all collections
show collections

// Find one subscriber
db.subscribers.findOne()

// Preview towers
db.towers.find().pretty()
```

READ (Find Documents)

- MongoDB uses the find() method to query collections.
  - db.collection.findOne() → returns the first match.
  - db.collection.find(query, projection) → multiple matches.
  - projection controls which fields to return.

FIND Document

Find all prepaid subscribers in Denpasar

```
db.subscribers.find(
  { home_regency: "Denpasar",
sim_type: "prepaid" }
)
```

Return only subscriber_id and sim_type (projection)

```
db.subscribers.find(
  { home_regency: "Denpasar" },
  { subscriber_id: 1, sim_type: 1, _id: 0 }
)
```

FIND Document

Find all events for subscriber_id = 1

```
db.cdr_events.find(
  { subscriber_id: 1 }
)
```

Find events between two dates

```
db.cdr_events.find({
  event_ts: {
    $gte: ISODate("2025-09-10T00:00:00Z"),
    $lte: ISODate("2025-09-12T23:59:59Z")
  }
})
```

FIND Document

Find top 5 busiest towers (using aggregates)

```
db.cdr_events.aggregate([
  { $group: { _id: "$tower_id",
total_events: { $count: {} } } },
  { $sort: { total_events: -1 } },
  { $limit: 5 }
])
```

Find tourism events happening in Gianyar

```
db.tourism_events.find(
  { regency: "Gianyar" }
)
```

FIND Document with PRETTY

Show all subscribers (pretty format)

```
db.subscribers.find().pretty()
```

Result:


```
{
  "subscriber_id": 1,
  "home_regency": "Denpasar",
  "home_tower_id": 101,
  "sim_type": "prepaid"
}
{
  "subscriber_id": 2,
  "home_regency": "Gianyar",
  "home_tower_id": 102,
  "sim_type": "postpaid"
}
```

FIND Document with PRETTY

Find prepaid subscribers in Denpasar

```
db.subscribers.find(
  { home_regency: "Denpasar", sim_type: "prepaid" }
).pretty()
```

FIND Document with PRETTY

Find CDR events for one subscriber

```
db.cdr_events.find(
  { subscriber_id: 1 }
).pretty()
```

Result:

```
{
  "event_id": 5001,
  "subscriber_id": 1,
  "tower_id": 101,
  "event_type": "call",
  "event_ts": ISODate("2025-09-12T10:30:00Z")
}
```

FIND Document with PRETTY

Find all towers with missing regency

```
db.towers.find(
  { regency: null }
).pretty()
```

Find tourism events in Gianyar

```
db.tourism_events.find(
  { regency: "Gianyar" }
).pretty()
```

FIND Document with AND or OR

Prepaid subscribers in Denpasar

```
db.subscribers.find(
  { home_regency: "Denpasar",
sim_type: "prepaid" }
).pretty()
```

```
home_regency = "Denpasar"
AND sim_type = "prepaid".
```

Subscribers in Denpasar or Gianyar

```
db.subscribers.find(
  { $or: [ { home_regency: "Denpasar"
}, { home_regency: "Gianyar" } ] }
).pretty()
```

Events of type “call” OR “sms”

```
db.cdr_events.find(
  { $or: [ { event_type: "call" }, {
event_type: "sms" } ] }
).pretty()
```

FIND Document with AND and OR

Prepaid subscribers in Denpasar OR Gianyar

```
db.subscribers.find(
  {
    sim_type: "prepaid",
    $or: [ { home_regency: "Denpasar" }, { home_regency: "Gianyar" } ]
  }
).pretty()
```

```
sim_type = "prepaid" AND (home_regency = "Denpasar" OR "Gianyar")
```

UPDATE Document

Update one subscriber’s SIM type

```
db.subscribers.updateOne(
  { subscriber_id: 1 },              // filter
  { $set: { sim_type: "postpaid" } } // update
)
```

Update many towers with missing regency

```
db.towers.updateMany(
  { regency: null },
  { $set: { regency: "Unknown" } }
)
```

UPDATE Document

Increment a field (e.g., add 1 to a counter)

```
db.cdr_events.updateOne(
  { event_id: 5001 },
  { $inc: { call_duration: 1 } }
)
```

Add a new field to all prepaid subscribers

```
db.subscribers.updateMany(
  { sim_type: "prepaid" },
  { $set: { promo_eligible: true } }
)
```

UPDATE Document

Add a new field to all prepaid subscribers

```
db.subscribers.updateMany(
  { sim_type: "prepaid" },
  { $set: { promo_eligible: true } }
)
```

Remove a field with $unset

```
db.subscribers.updateOne(
  { subscriber_id: 2 },
  { $unset: { promo_eligible: "" } }
)
```

DELETE Document

MongoDB provides:

- deleteOne() → remove the first matching document
- deleteMany() → remove all matching documents

Delete one subscriber by ID

```
db.subscribers.deleteOne(
  { subscriber_id: 999 }
)
```

DELETE Document

Delete all tourism events that ended before September 2025

```
db.tourism_events.deleteMany(
  { end_date: { $lt: ISODate("2025-09-01T00:00:00Z") } }
)
```

Delete all CDR events for a specific subscriber

```
db.cdr_events.deleteMany(
  { subscriber_id: 1 }
)
```

DELETE Document

Delete towers with Unknown regency

```
db.towers.deleteMany(
  { regency: "Unknown" }
)
```

Delete everything from a collection

```
db.cdr_events.deleteMany({})
```

Exercise

- Goals:

1. Understand how to Create, Read, Update, Delete (CRUD) documents in MongoDB
2. Practice queries on the MPD Collections: subscribers, towers, cdr_events, tourism_events

Exercise

1. Insert two new towers:
    1. Tower 201 in Gianyar
    2. Tower 202 in Denpasar
2. Find all subscribers in Denpasar
3. Find only subscriber_id and sim_type for prepaid subscribers
4. Find all call events between “2025-09-10” and “2025-09-12”
5. Find all events in Gianyar
6. Change subscriber 2001’s sim type from prepaid
7. Update all towers with null regency to “Unknown”
8. Delete all documents from cdr_events but keep collection structure.

Thank You

Email: sitimariyah@stis.ac.id
Github: https://github.com/diahnuri