Dr. Wittawin Susutti
wittawin.sus@kmutt.ac.th

CSS227 WEB PROGRAMMING

LECTURE 08 – NOSQL

What's NoSQL?

- A No SQL database provides a mechanism for storage and retrieval of data that employs **less constrained consistency models** than traditional relational database → เก็บข้อมูลแบบ flexible

- No SQL systems are also referred to as "**Not only SQL**".

- Next generation databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontally scalable**.
- Often more characteristics apply as: **schema-free**, **easy replication support**, **simple API**, **eventually consistent**, a **huge data amount**, and more.

schema-free : ไม่ต้องออกแบบ schema → ไม่มี ER Diagram

สำคัญ : NOSQL สำคัญใน ยุค BigData

Characteristics of NoSQL

- NoSQL **avoids**:
  - Overhead of ACID transactions → ไม่รองรับ ACID
  - Complexity of SQL query → Qry ที่ซับซ้อน
  - Burden of up-front schema design → ออกแบบ DB
  - Transactions → ไม่รองรับ Transaction
- **Provides**:
  - Easy and frequent changes to DB
  - Fast development
  - Large data volumes
  - Schema less

Data Volume มาก ↑
ง่ายและเปลี่ยนตลอดเวลา
เร็ว !!!

Schema-less datamodel

In relational Databases:

- You can not add a record which **does not fit the schema**
- You need to **add NULLs** to **unused items** in a row
- We should **consider the datatypes**.
- You **can not add multiple items in a field**

Schema-less datamodel

In NoSQL Databases:

- There is **no schema** to consider
- There is **no unused cell** → ไม่มี unused cell
- There is **no datatype** → ไม่มี datatype
- We gather all items in a document → เก็บในรูปแบบ document

ยกตัวอย่าง : graph ง่ายๆ

What are the Types of NoSQL Databases?

4 แบบหลัก

- **Document databases**
  - Store data in documents like JSON (JavaScript Object Notation) objects → เก็บใน JSON File
  - Each document contains pairs of fields and values. → ประกอบด้วย Field และ Value
    → MongoDB เป็น document
- **Key-value databases**
  - A simpler type of database where each item contains keys and values. → เก็บใน key และ value
  - A value can typically only be retrieved by referencing its value → ใช้ ref จาก key
- **Wide-column stores**
  - Store data in tables, rows, and dynamic columns. → ใกล้เคียงกับ relational DB แต่เป็น Dynamic table
    ↓ fixed ↓ changable #
  - Each row is not required to have the same columns.
- **Graph databases**
  - Store data in nodes and edges. → เก็บใน node และ edge
  - Nodes typically store information about people, places, and things while edges store information about the relationships between the nodes.

Features of NoSQL Databases

Elastic scaling

- Traditional approach: scaling-up
  - Buying bigger servers as database load increases → จ่ายเพิ่มเพื่อเก็บเพิ่ม (cost ↑)
- New approach: scaling-out
  - Distributing database data across multiple hosts → database เดียว host หลายๆ (cost ↓)
    - Graph databases (unfortunately): difficult or impossible at all

↓
cloud service

Data distribution

- Sharding → แบ่ง data เป็นกลุ่มย่อยๆ
  - Ways how database data is split into separate groups
- Replication → copy data เป็นกลุ่มย่อยๆ
  - Maintaining several data copies (performance, recovery)

Features of NoSQL Databases

Automated processes

- Traditional approach
  - Expensive and highly trained database administrators
- New approach: automatic recovery, distribution, tuning, ...

Relaxed consistency

- Traditional approach
  - Strong consistency (ACID properties and transactions)
- New approach
  - Eventual consistency only (BASE properties)
  - I.e., we have to make trade-offs because of the data distribution

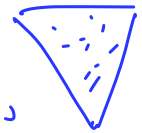
Features of NoSQL Databases

Schemalessness

- Relational databases
  - Database schema present and **strictly enforced** → เข้มงวดในการจัดเก็บ
- NoSQL databases
  - **Relaxed schema** or **completely missing**
  - Consequences: **higher flexibility** → ยืดหยุ่นมากขึ้น
    - Dealing with **non-uniform data**
    - **Structural changes** cause no overhead
  - However: there is (usually) an **implicit schema** → มี schema แต่ไม่ชัดเจน
    - We must know the data structure at the application level anyway

Advantages

- **Scaling**
  - Horizontal distribution of data among hosts → กระจายแบบแนวกว้าง



- **Volume**
  - High volumes of data that cannot be handled by RDBMS → รองรับการรับข้อมูลมากกว่า
- **Administrators**
  - No longer needed because of the automated maintenance → ไม่จำเป็นต้อง expert
- **Economics**
  - Usage of cheap commodity servers, lower overall costs → ใช้ cheap commu cost ลดลง
- **Flexibility**
  - Relaxed or missing data schema, easier design changes → ยืดหยุ่นกว่า ฯ
    ง่ายเมื่อ design เปลี่ยน

Challenges

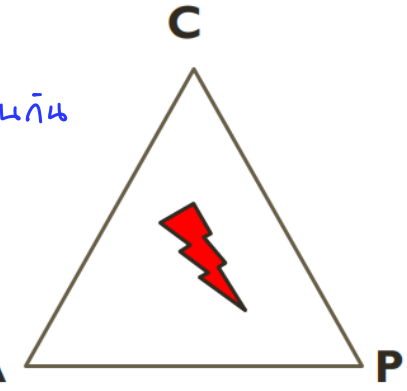
- **Maturity**
  - Often still in pre-production phase with key features missing
- **Support**
  - Mostly open-source, limited sources of credibility
- **Administration**
  - Sometimes relatively difficult to install and maintain
- **Analytics**
  - Missing support for business intelligence and ad-hoc querying
- **Expertise**
  - Still low number of NoSQL experts available in the market

CAP Theorem

GIVEN:

database ในหลาย node โดยในแต่ละ node เก็บ data ไม่เหมือนกัน

- Many nodes
- Nodes contain replicas of partitions of the data

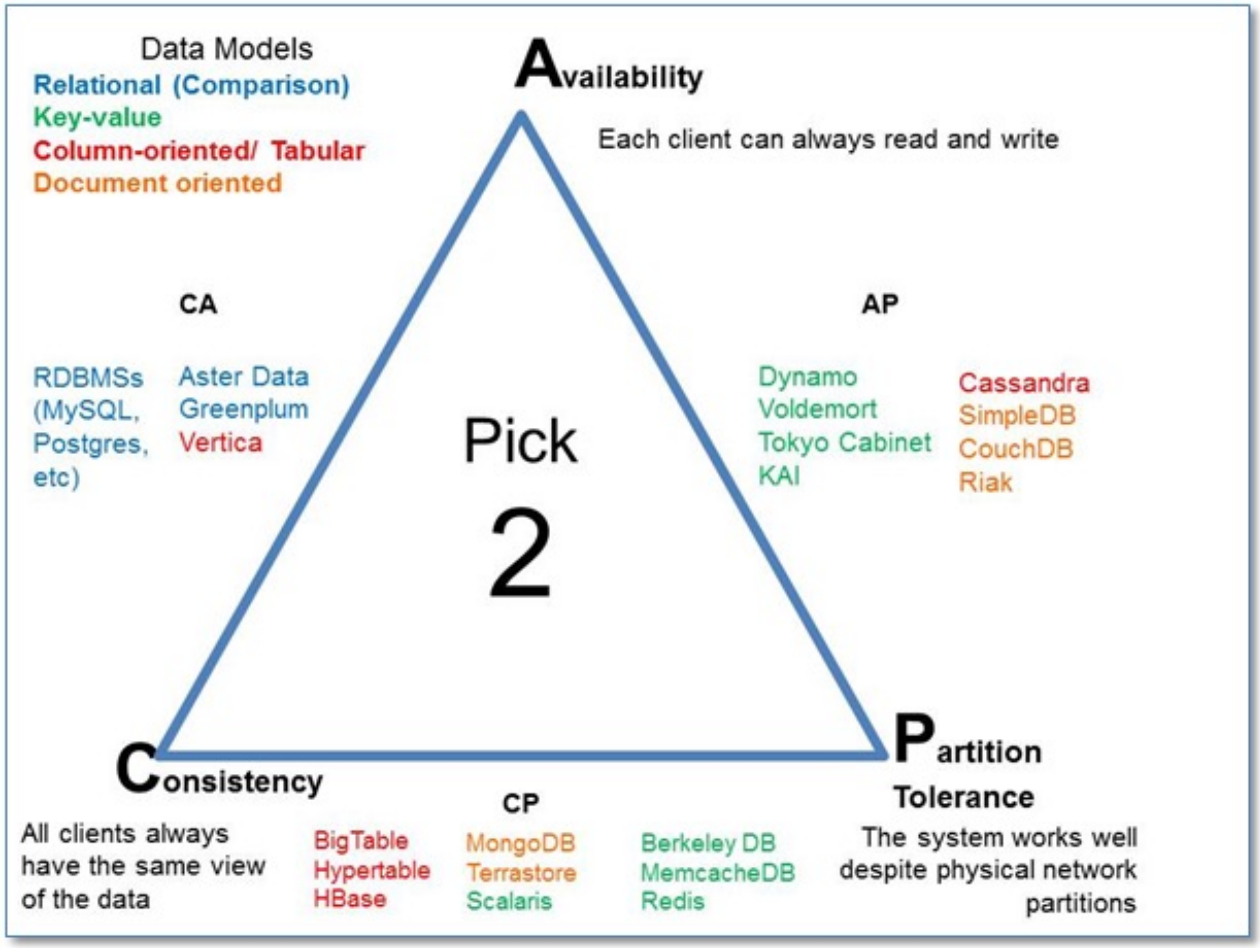


- **Consistency** : ความถูกต้อง
  - All replicas contain the same version of data ← All replica ต้องมี version เหมือนกัน
  - Client always has the same view of the data (no matter what node)
- **Availability**
  - System remains operational on failing nodes ← เข้าระบบได้แม้ node fail
  - All clients can always read and write → read & write ตลอดเวลา
- **Partition tolerance**
  - multiple entry points
  - System remains operational on system split (communication malfunction)
  - System works well across physical network partitions

Satisfying **all three** at the same time is **impossible**!

ต้องเลือก 1, 2 ใน 3 ที่จะอยู่ด้วยกันไม่ได้

CAP theorem


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick 2
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

NoSQL BASE

- NoSQL relies upon a softer model known as the BASE model.
- **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency).
  - **Basically Available:** → การันตี availability เสมอ → เข้าถึง data ได้
    - Guarantees the availability of the data .
    - There will be a response to any request (can be failure too).
  - **Soft state**: → state ของ system เปลี่ยนได้ตลอดเวลา
    - The state of the system could change over time.
  - **Eventual consistency:** → Final ข้อมูลถูกต้อง ระหว่างเวลาในไม่สน
    - The system will eventually become consistent once it stops receiving input.
- NoSQL databases give up the A, C and/or D requirements, and in return they improve scalability. → ไม่สนใจ A,C,D , scaling ง่ายขึ้น

SQL vs NoSQL

Parameter	SQL	NOSQL
Definition	SQL databases are primarily called RDBMS or **Relational Databases**	NoSQL databases are primarily called as **Non-relational or distributed database**
Design for	Traditional RDBMS uses SQL syntax and queries to analyze and get the data for further insights.	These databases were developed in response to the demands presented for the development of the modern application.
Query Language	Structured query language (**SQL**)	**No declarative query language**
Type	SQL databases are **table-based databases**	NoSQL databases can be **document based**, key-value pairs, graph databases
Schema	SQL databases have a **predefined schema**	NoSQL databases use **dynamic schema** for unstructured data.
Examples	Oracle, Postgres, and MS-SQL.	MongoDB, Redis, Neo4j, Cassandra, Hbase.
Best suited for	An ideal choice for the **complex query intensive environment**.	It is **not good fit complex queries**.
Best Used for	RDBMS database is the right option for **solving ACID problems**.	NoSQL is a best used for solving **data availability problems**
Importance	It should be used when **data validity** is super important	Use when it's more important to have **fast data than correct data**
ACID vs. BASE	**ACID** (Atomicity, Consistency, Isolation, and Durability) is a standard for RDBMS	**BASE** (Basically Available, Soft state, Eventually Consistent) is a model of many NoSQL systems

Conclusion

The end of relational databases?

- Certainly no → ไม่!!! ยังถูกใช้งานอยู่มาก relational db
  - They are still suitable for most projects
  - Familiarity, stability, feature set, available support, ...
- However, we should also consider different database models and systems
  - Polyglot persistence = usage of different data stores in different circumstances

MongoDB

- A document-oriented, NoSQL database
- No Data Definition Language
- Document Identifiers (_id) will be created for each document, field name reserved by system
- Uses BSON format
  - Based on JSON –B stands for Binary

Functionality of MongoDB

- Dynamic schema
  - No DDL
- Document-based database
- Query language via an API
- Atomic writes and fully-consistent reads
- Master-slave replication
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

Why use MongoDB?

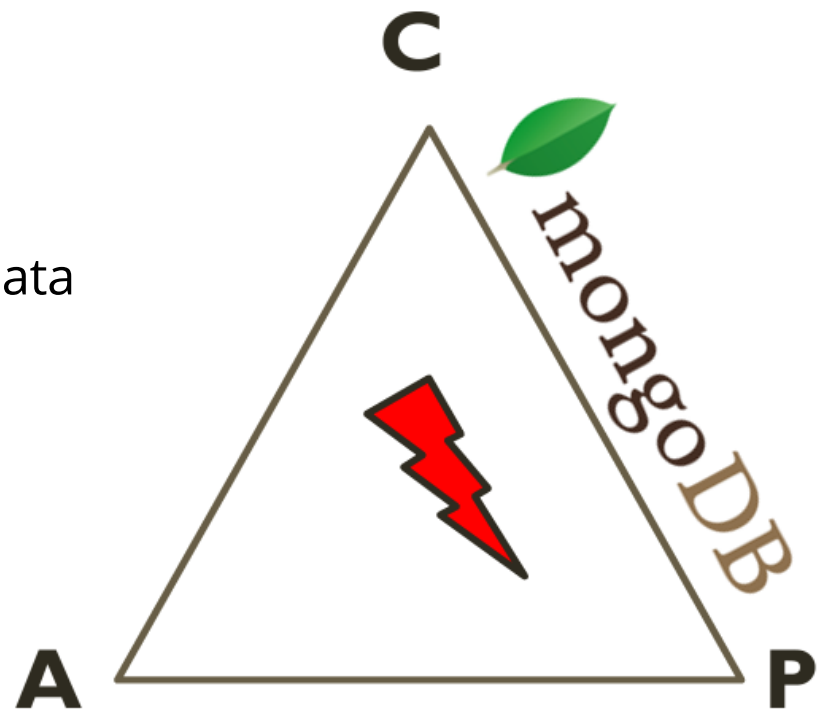
- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

Advantages of MongoDB

- **Schema less** − one collection holds different documents.
- Structure of a single object is clear.
- No complex joins.
- Tuning.
- **Ease of scale-out.**
- Faster access of data.
- Simple queries.
- Functionality provided applicable to most web applications.
- Easy and fast integration of data.
- No ERD diagram.
- Not well suited for heavy and complex transactions systems.

MongoDB: CAP approach

- Focus on **Consistency** and **Partition tolerance**
- Consistency
  - all replicas contain the same version of the data
- Availability
  - system remains operational on failing nodes
- Partition tolerance
  - multiple entry points system remains operational on system split



MongoDB concepts

Database:

- A container of MongoDB collections

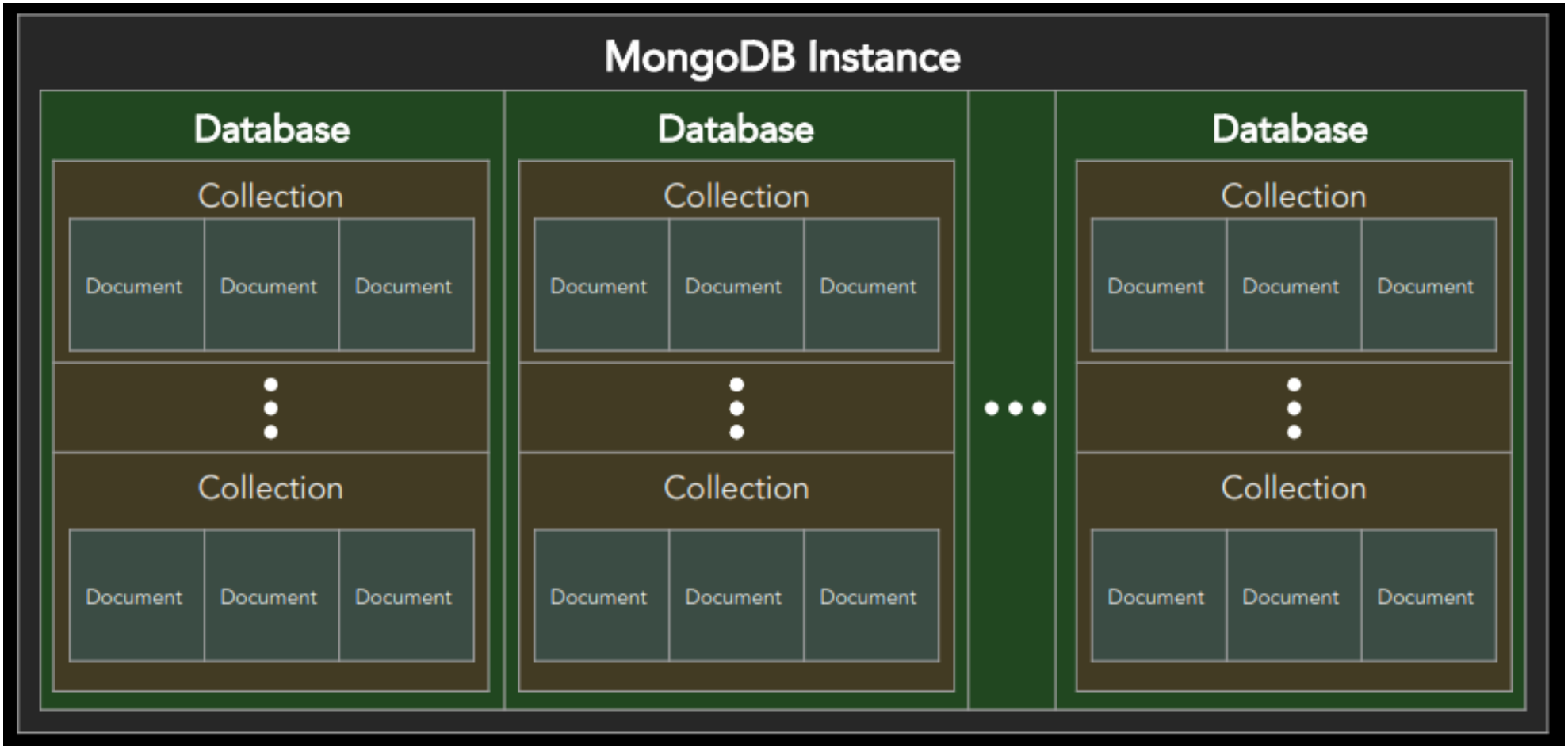
Collection:

- A group of MongoDB documents.

Document:

- A JSON-like object that represents one instance of a collection
- Also used more generally to refer to any set of key-value pairs.

MongoDB: Hierarchical Objects



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)

Relational database:

Name	School	Employer	Occupation
Lori	null	Self	Entrepreneur
Malia	Harvard	null	null

Document-oriented DB:

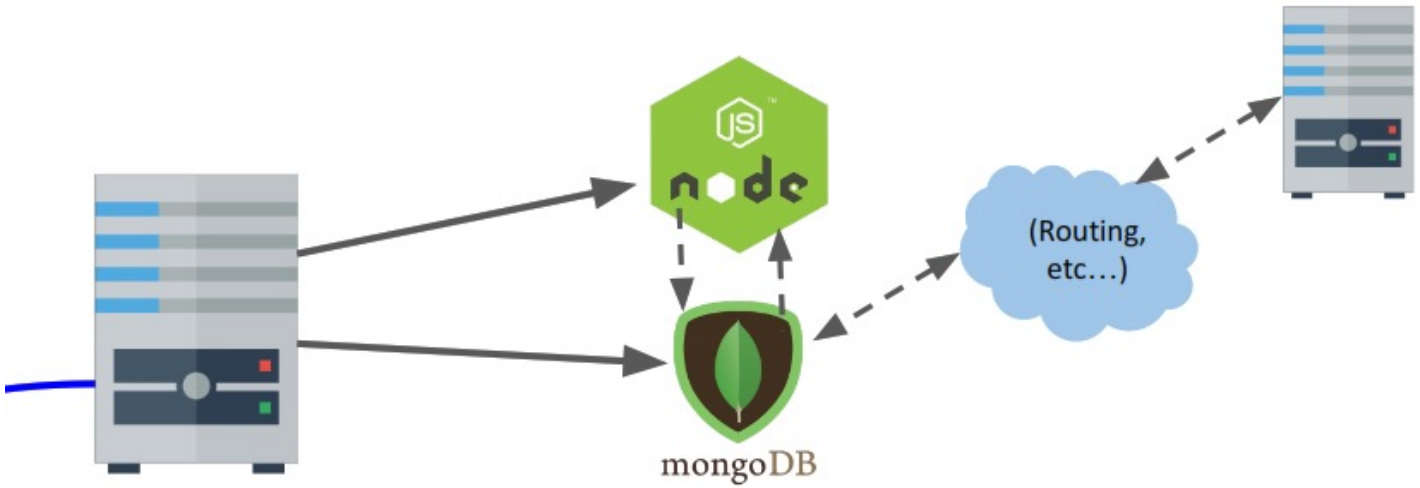
```
{
  name: "Lori",
  employer: "Self",
  occupation: "Entrepreneur"
}
{
  name: "Malia",
  school: "Harvard"
}
```

Choices made for Design of MongoDB

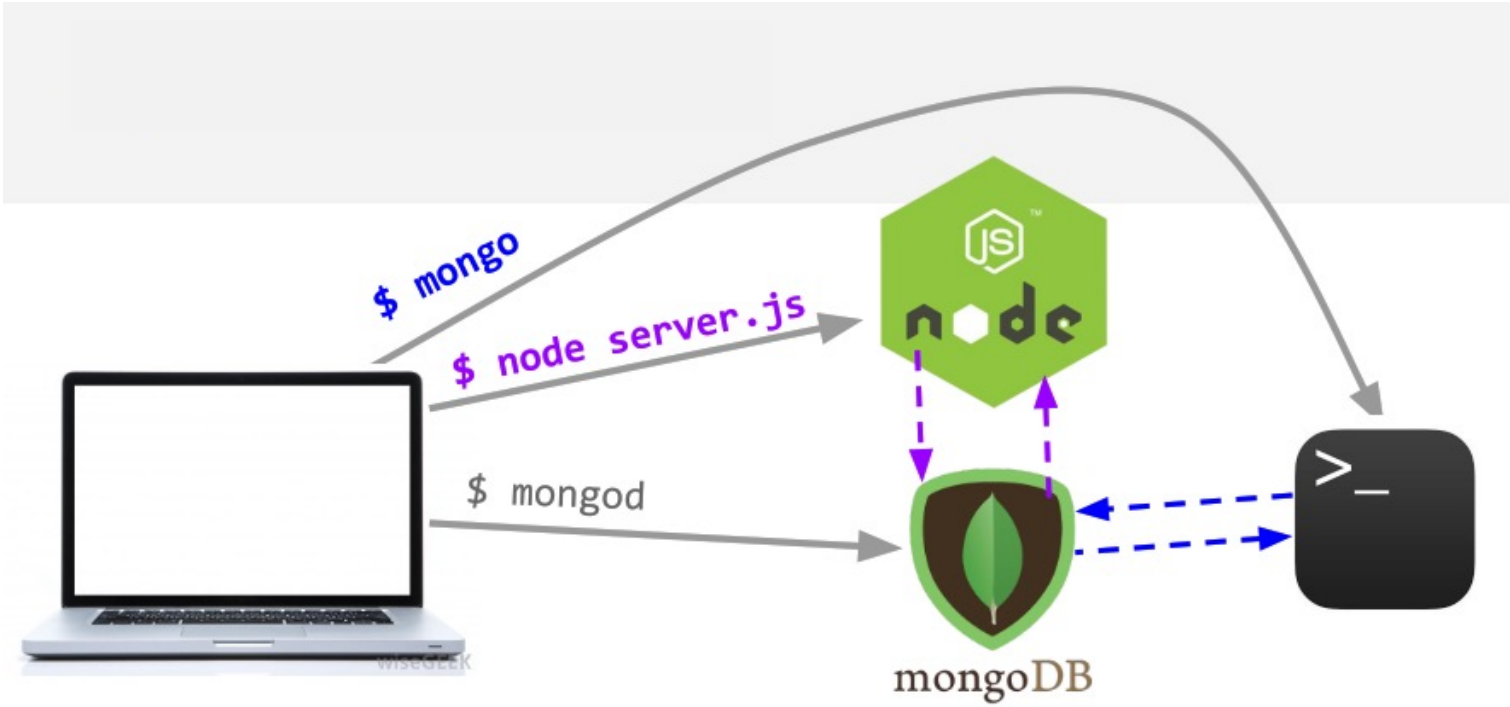
- Scale horizontally over commodity hardware
  - Lots of relatively inexpensive servers
- Keep the functionality that works well in RDBMSs
  - Ad-hoc queries
  - Fully featured indexes
  - Secondary indexes

MongoDB

- MongoDB is another software program running on the computer.
- It is also known as the MongoDB server.
- There are MongoDB libraries we can use in NodeJS to communicate with the MongoDB Server.
- The MongoDB Server manages might be local to the server computer or it could be stored on cloud storage.



System overview



Example

- Website has the following requirements.
  - Every post has the unique title, description and url.
  - Every post can have one or more tags.
  - Every post has the name of its publisher and total number of likes.
  - Every post has comments given by users along with their name, message, data-time and likes.
  - On each post, there can be zero or more comments.

CRUD operations

- **Create**
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- **Read**
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- **Update**
  - db.collection.update(<query>, <update>, <options>)
- **Delete**
  - db.collection.remove(<query>, <justOne>)
- Collection specifies the collection or the ‘table’ to store the document

Create Operations

Db.collection specifies the collection or the ‘table’ to store the document

- db.collection_name.insert(<document>)
  - Omit the _id field to have MongoDB generate a unique key
  - Example: db.parts.insert({{type: “screwdriver”, quantity: 15 })
  - db.parts.insert({_id: 10, type: “hammer”, quantity: 1 })
- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.save(<document>)
  - Updates an existing record or creates a new record

Read Operations

db.collection.find(<query>, <projection>).cursor modified

- Provides functionality similar to the SELECT command
  - <query> where condition , <projection> fields in result set
- Example: varPartsCursor= db.parts.find({parts: “hammer”}).limit(5)
- Has cursors to handle a result set
- Can modify the query to impose limits, skips, and sort orders.
- Can specify to return the ‘top’ number of records from the result set

db.collection.findOne(<query>, <projection>)

Update Operations

db.collection_name.insert(<document>)

- Omit the _id field to have MongoDB generate a unique key
- Example: db.parts.insert({{type: “screwdriver”, quantity: 15 })
- db.parts.insert({_id: 10, type: “hammer”, quantity: 1 })

- db.collection_name.save(<document>)
  - Updates an existing record or creates a new record
- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.findAndModify(<query>, <sort>, <update>,<new>, <fields>,<upsert>)
  - Modify existing record(s) –retrieve old or new version of the record

Delete Operations

db.collection_name.remove(<query>, <justone>)

- Delete all records from a collection or matching a criterion
- <justone> -specifies to delete only 1 record matching the criterion
- Example: db.parts.remove(type: /^h/ }) -remove all parts starting with h
- Db.parts.remove() –delete all documents in the parts collections

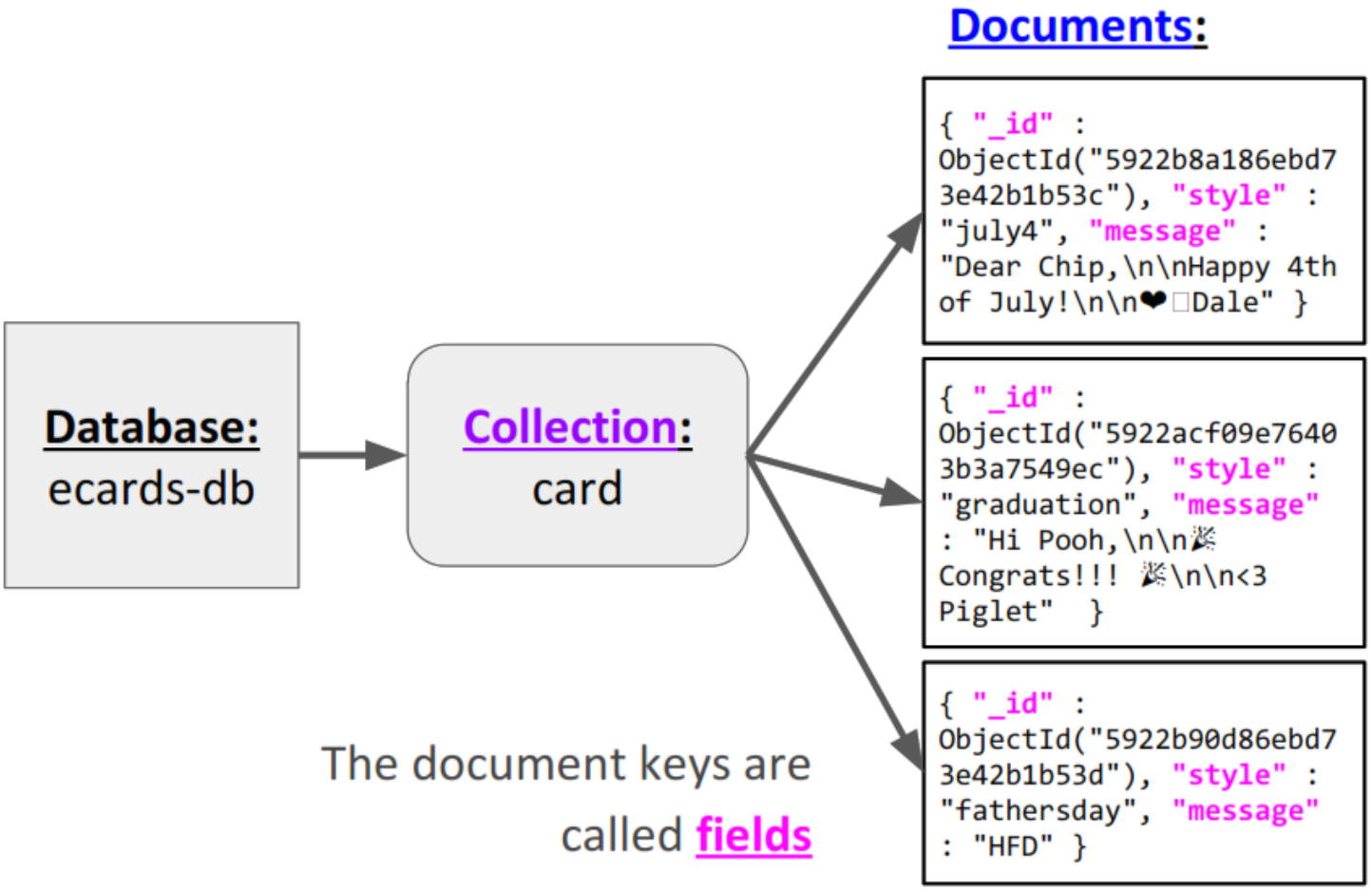
MongoDB schema

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

MongoDB Example Queries

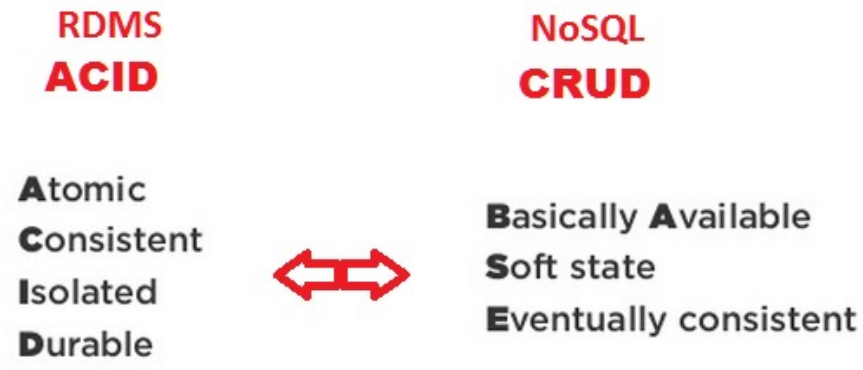
- db.cds.count()
- db.cds.find({ artist : ‘Taylor Swift’ })
- db.cds.find(
  { “tracks.artist” : ‘Jaques Brel’, “tracks.name” : “Mathilde” })
- db.cds.runCommand(“text”, { search: ‘Paris’ })

MongoDB example



Mongo Transactions

- No ACID guarantees
  - Clients may see writes before they are committed
  - If a query updates multiple documents, might read some updates and not others
- BASE instead
  - High availability
  - Eventual consistency



MongoDB Schema Design

- With MongoDB schema design, there is:
  - No formal process
  - No algorithms
  - No rules
- When you are designing your MongoDB schema design, the only thing that matters is that you design a schema that will work well for ***your*** application.
- Two different apps that use the same exact data might have very different schemas if the applications are used differently.
- When designing a schema, we want to take into consideration the following:
  - Store the data
  - Provide good query performance
  - Require reasonable amount of hardware

Embedding vs. Referencing

Embedding	Referencing
Advantages	
• You can retrieve all relevant information in a single query. • Avoid implementing joins in application code or using $lookup. • Update related information as a single atomic operation. • However, if you need a transaction across multiple operations, you can use the transaction operator.	• By splitting up data, you will have smaller documents. • Infrequently accessed information not needed on every query. • Reduce the amount of duplication of data. • However, it's important to note that data duplication should not be avoided if it results in a better schema.
Limitations	
• Large documents mean more overhead if most fields are not relevant. • There is a 16-MB document size limit in MongoDB.	• In order to retrieve all the data in the referenced documents, a minimum of two queries or $lookup required to retrieve all the information.

Type of Relationships

One-to-One

```
{
   "_id": "ObjectId('AAA')",
   "name": "Joe Karlsson",
   "company": "MongoDB",
   "twitter": "@JoeKarlsson1",
   "twitch": "joe_karlsson",
   "tiktok": "joekarlsson",
   "website": "joekarlsson.com"
}
```

One-to-Few

```
{
   "_id": "ObjectId('AAA')",
   "name": "Joe Karlsson",
   "company": "MongoDB",
   "twitter": "@JoeKarlsson1",
   "twitch": "joe_karlsson",
   "tiktok": "joekarlsson",
   "website": "joekarlsson.com",
   "addresses": [
     { "street": "123 Sesame St",
       "city": "Anytown", "cc": "USA" },
     { "street": "123 Avenue Q",
       "city": "New York", "cc": "USA" }
   ]
}
```

One-to-Many

```
{
   "name": "left-handed smoke shifter",
   "manufacturer": "Acme Corp",
   "catalog_number": "1234",
   "parts": ["ObjectID('AAAA')",
             "ObjectID('BBBB')",
             "ObjectID('CCCC')"]
}
```

```
{
   "_id" : "ObjectID('AAAA')",
   "partno" : "123-aff-456",
   "name" : "#4 grommet",
   "qty": "94",
   "cost": "0.94",
   "price":" 3.99"
}
```

One-to-Squillions

```
{
   "_id": ObjectID("AAAB"),
   "name": "goofy.example.com",
   "ipaddr": "127.66.66.66"
}
```

```
{
   "time": ISODate("2014-03-28T09:42:41.382Z"),
   "message": "cpu is on fire!",
   "host": ObjectID("AAAB")
}
```

Many-to-Many

```
{
   "_id": ObjectID("AAF1"),
   "name": "Kate Monster",
   "tasks": [ObjectID("ADF9"),
             ObjectID("AE02"),
             ObjectID("AE73")]
}
```

```
{
   "_id": ObjectID("ADF9"),
   "description": "Write blog post about
                   MongoDB schema design",
   "due_date": ISODate("2014-04-01"),
   "owners": [ObjectID("AAF1"),
              ObjectID("BB3G")]
}
```

General Rules for MongoDB Schema Design

One-to-One

- Prefer key value pairs within the document

One-to-Few

- Prefer embedding

One-to-Many

- Prefer embedding

One-to-Squillions

- Prefer Referencing

Many-to-Many

- Prefer Referencing

Rule 1: Favor embedding unless there is a compelling reason not to.

Rule 2: Needing to access an object on its own is a compelling reason not to embed it.

Rule 3: Avoid joins and lookups if possible, but don't be afraid if they can provide a better schema design.

Rule 4: Arrays should not grow without bound.

- If there are more than a couple of hundred documents on the *many* side, don't embed them
- If there are more than a few thousand documents on the *many* side, don't use an array of ObjectID references.
- High-cardinality arrays are a compelling reason not to embed.

Rule 5: As always, how you model your data depends **entirely** on your particular application's data access patterns.

NoSQL Databases

- **Pros**
  - Can store arbitrary objects
  - Easy to set up and use (little management)
  - Easy to change
  - Scales nicely

- **Cons**
  - No transactional guarantees
  - Data consistency is up to the user
  - Complex queries need to be coded explicitly

What Type of Database to Use

- How important are ACID guarantees
  - Newer versions of Mongo provide some of this
- How important is data consistency
  - SQL provides constraints and triggers
- How important is scalability
  - Newer versions of SQL database provide sharding, etc.
- How varied and complex are the queries
  - Complex, unanticipated NoSQL queries require code
- How complex is the data and how often does the schema change
  - Complex data is much easier in NoSQL
- Might want a combination of databases
  - Some for complex data
  - Some for users, authentication, etc. where ACID is useful