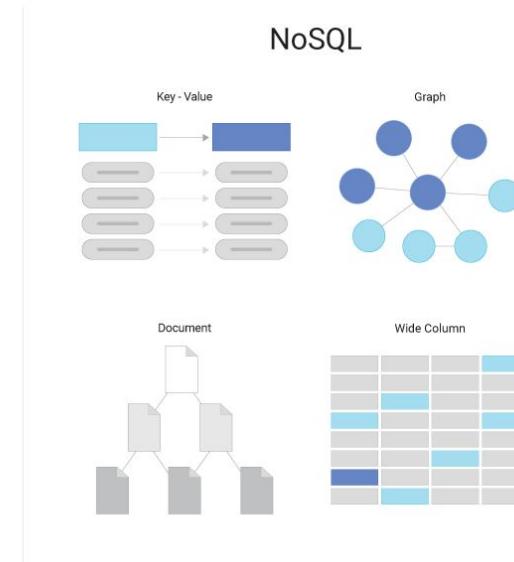


MongoDB

Goal: Build the mental model & write simple MongoDB queries

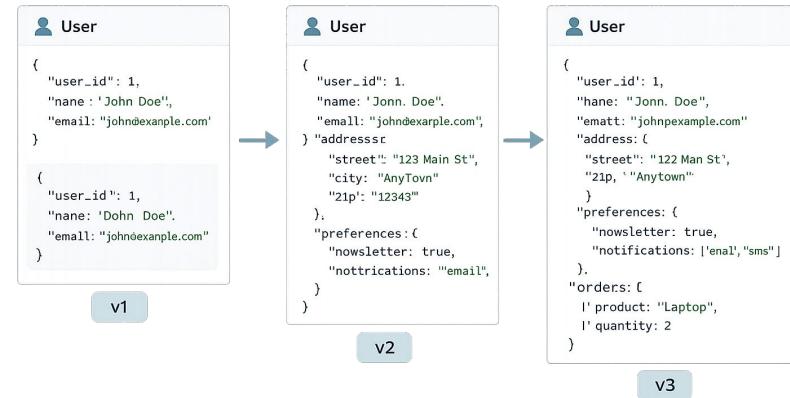
What is NoSQL

- NoSQL is a category of databases that do not rely on relational tables.
- Different NoSQL models exist for different needs, such as document and key value.
- MongoDB is a document database, which means it stores data as documents.



What is MongoDB

- MongoDB is a NoSQL database that stores data as documents.
- A document looks like JSON and can include nested objects and arrays.
- When the shape of the data changes in a system (eg: when your product evolves), MongoDB can be a good choice.

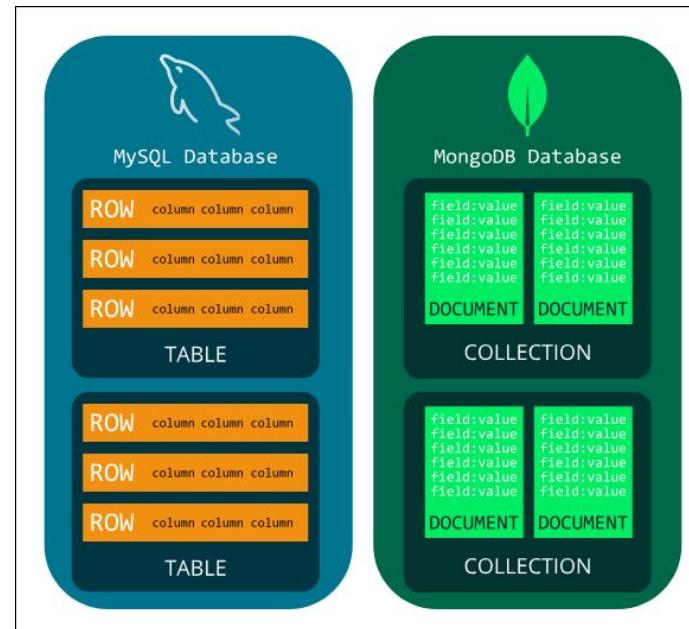


Where MongoDB Fits

- User profiles and settings often change over time and can be stored as flexible documents.
- Activity and event data can arrive in high volume and often gains new fields as features are added.
- Content and product metadata often benefits from nested structures, such as tags, variants, and attributes.
- Systems at companies like Google, Netflix, and Stripe often deal with profiles, catalogs, and event streams at scale, where NoSQL storage models are a common fit.

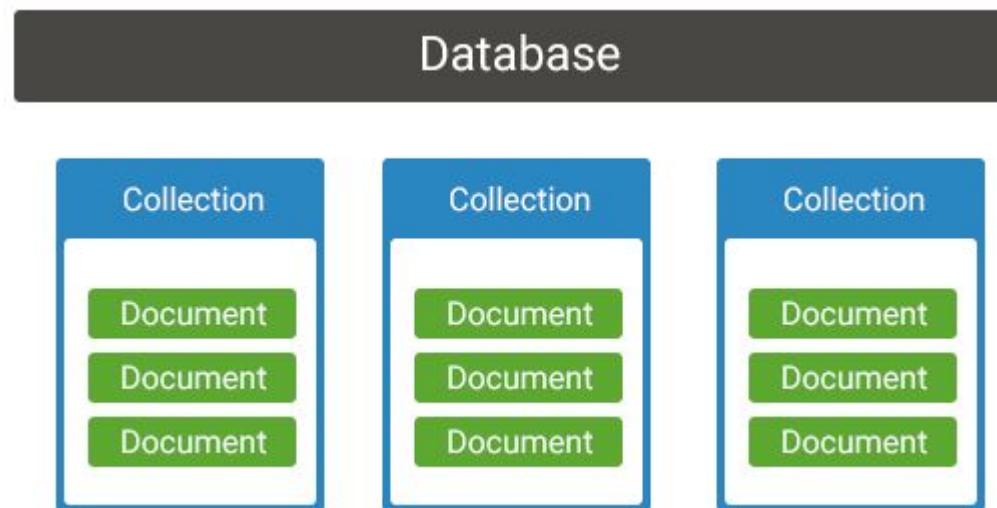
MongoDB and MySQL Vocabulary

- A MySQL table is similar to a MongoDB **collection**.
- A MySQL row is similar to a MongoDB **document**.
- A MySQL column is similar to a **field** inside a document.
- Note: both systems store databases, but they organize data differently.



Documents and Collections in MongoDB

- A **document** is one record stored as key value pairs.
- A **collection** is a group of documents of a similar type.
- Every document has an `_id` field that uniquely identifies it.



Nested Data in a Document

- An employee document can store address as an object and skills as an array.
- This makes it natural to fetch related details together in one read.
- The goal is to model data in the shape your application uses.

```
1 {  
2   "name": "Asha",  
3   "age": 24,  
4   "department": "Engineering",  
5   "skills": ["Java", "MySQL"],  
6   "address": {  
7     "city": "Bangalore",  
8     "pincode": 560001  
9   }  
10 }
```

Tools and Connection

MongoDB Server

- MongoDB Server is the database engine that stores data and runs queries.
- When you connect from Compass or Shell, you are connecting to the MongoDB Server.
- [Download link](#)

MongoDB Compass

- MongoDB Compass is a graphical tool to browse databases, collections, and documents.
- It lets you run queries and aggregation pipelines using MQL and view results immediately.
- [Download link](#)

The screenshot shows the MongoDB Compass interface. On the left, the 'Local' sidebar displays connection information: HOST localhost:27017, CLUSTER Standalone, and EDITION MongoDB 4.4.1 Community. Below this are sections for 'School' (highlighted) and 'admin', 'company'. The main panel shows the 'School.Students' collection. The 'Documents' tab is selected, showing one document with the following data:

```
_id: ObjectId("5f848a2d97e467a50c3a1eb9")
name: "Mike"
age: "15"
class: "10A"
```

Mongo Shell

- Mongo Shell is a command line tool for MongoDB.
- It is useful for running commands quickly and repeating them reliably.
- [Download link](#) (Note: this is an optional tool for our course)

MongoDB Query Language

- MongoDB Query Language, or MQL, is the syntax used to work with documents.
- MQL is used for inserting, finding, updating, deleting, and aggregating data.
- You can write MQL inside MongoDB Compass and inside Mongo Shell.

Connecting to MongoDB

- Install MongoDB Server and ensure the service is running.
- Open MongoDB Compass and connect using your connection string.
- After connecting, you can create a database and start inserting documents.

CRUD Operations

CRUD

- CRUD describes the four actions you perform on stored data.
- **Create** adds new data,
- **Read** fetches data,
- **Update** changes data, and
- **Delete** removes data.

CRUD in MongoDB

- Create is done with **insert** operations.
- Read is done with **find** operations.
- Update is done with **update** operations.
- Delete is done with **delete** operations.

Data Model We'll Use

- We will use one database, **company_db**.
- We will use one collection, **employees**.
- Employees will have the following fields: **name, age, department, skills, and address**.

Create

- Create means inserting a new document into a collection.

```
1 db.employees.insertOne({  
2   name: "Asha",  
3   age: 24,  
4   department: "Engineering",  
5   skills: ["Java", "MySQL"],  
6   address: { city: "Bangalore", pincode: 560001 }  
7 });
```

Read

- Read means finding documents that match a filter.

```
1 db.employees.find({ department: "Engineering" });
```

- This returns all employee documents where the department field matches Engineering.

Update

- Update means changing fields in matching documents.
- A common pattern is to add an item to an array field such as skills. Alternatively, an update may simply update the age of an employee.

```
1 db.employees.updateOne(  
2   { name: "Asha" },  
3   { $addToSet: { skills: "MongoDB" } }  
4 );
```

```
1 db.employees.updateOne(  
2   { name: "Asha" },  
3   { $set: { age: 30 } }  
4 );
```

- `$addToSet` adds the value only if it is not already present in the array.
 - `$addToSet` is an update operator, which is a special keyword used inside an update command to modify document(s).

Delete

- Delete means removing documents that match a filter.

```
1 db.employees.deleteMany({ age: { $lt: 25 } });
```

- This removes all employee documents where age is less than 25.
- In order to delete a single document matching a condition, you may use deleteOne(...)

Querying with Filters and Projection

Querying

- Querying means selecting documents that match conditions.
- A filter defines the condition that documents must match.
- An [equality filter](#) matches an exact value in a field.
- Comparison operators such as \$gt and \$lt match a range of values.

```
1 db.employees.find({ department: "Engineering" });
2
3 db.employees.find({ age: { $gt: 25 } });
4
5 db.employees.find({ age: { $lt: 25 } });
```

Combining Conditions

- You can combine multiple conditions in a single filter object.
- The below query matches employees in Engineering who are older than 25.

```
1 db.employees.find({  
2   department: "Engineering",  
3   age: { $gt: 25 }  
4 });
```

Projection

- Projection controls which fields come back in the result.
- This helps you fetch only what you need, such as name and department.
- The query below returns only name and department and hides `_id`.

```
1 db.employees.find(  
2   { department: "Engineering" },  
3   { name: 1, department: 1, _id: 0 }  
4 );
```

Aggregation Framework

Aggregation

- **Aggregation** summarizes many documents into fewer results.
- It answers questions like counts and averages by group.
- In MySQL, we use COUNT and AVG with GROUP BY.
- In MongoDB, we use an aggregation pipeline to produce similar summaries.
 - An aggregation pipeline is a sequence of stages.
 - Each stage transforms the data and passes results to the next stage.

Stages We'll Cover

- **\$match** filters documents before grouping.
- **\$group** groups documents and computes summary values.
- **\$project** shapes the final output fields.
- Table we'll use:

name	age	department	skills	address
Asha	24	Engineering	["Java", "MySQL"]	{ city: "Bengaluru", pincode: 560001 }
Rohan	28	Engineering	["Node.js", "MongoDB"]	{ city: "Pune", pincode: 411001 }
Meera	22	HR	["Hiring", "Onboarding"]	{ city: "Chennai", pincode: 600001 }
Kabir	30	Finance	["Excel", "Reporting"]	{ city: "Mumbai", pincode: 400001 }
Zoya	26	Sales	["Lead Gen", "CRM"]	{ city: "Hyderabad", pincode: 500001 }

Aggregation: \$match

- This single pipeline matches employees by skills

```
1 db.employees.aggregate([
2   { $match: { skills: "MongoDB" } }
3 ]);
```

Output:

name	age	department	skills	address
Rohan	28	Engineering	["Node.js", "MongoDB"]	{ city: "Pune", pincode: 411001 }

Aggregation: \$match + \$group

- The pipeline below first selects employees whose skills include "MongoDB".
- Then it groups the matching employees by department.
- For each department, it outputs the number of matching employees as total.
- How?
 - `_id: "$department"` creates one bucket per department (Engineering, Sales, etc)
 - `total: { $sum: 1 }` adds 1 for each matching employee in that bucket, so total becomes the count per department

Pipeline:

```
1 db.employees.aggregate([  
2   { $match: { skills: "MongoDB" } },  
3   { $group: { _id: "$department", total: { $sum: 1 } } }  
4 ]);
```

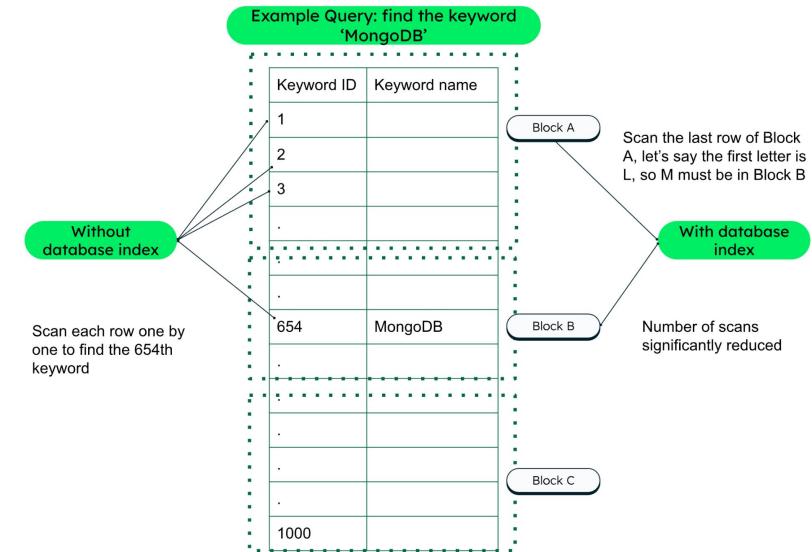
Output:

department	total
Engineering	1

Indexing Basics

What is an Index

- An index is a data structure that helps MongoDB search faster on a field.
- Indexes matter most when collections grow large and queries repeat frequently.
- Trade Offs
 - Indexes improve read performance for common queries.
 - They consume storage and can make writes (inserts, updates) slower because the index must be maintained
- When do we create an index?
 - Fields that you filter on frequently (Eg: filter by department)
 - Fields that you sort on frequently (Eg: sort by name)
 - Note: Avoid indexing every field unless you have a clear reason



Creating an Index

- This creates an index on department, which helps queries that filter by department.

```
1 db.employees.createIndex({ department: 1 });
```

- The impact is faster reads for matching queries, with a small cost on insert and update operations.

Your turn!

Activity

- Create database **library_db** and collection **members**
- Insert 5 member documents with **fields**:
 - name, age, membershipType, borrowedBooks (array), address (object)
- **Update** one member by adding a new book to borrowedBooks
- **Delete** members where age < 18
- **Fetch** only name and membershipType using projection
- **Aggregation**: count members per membershipType