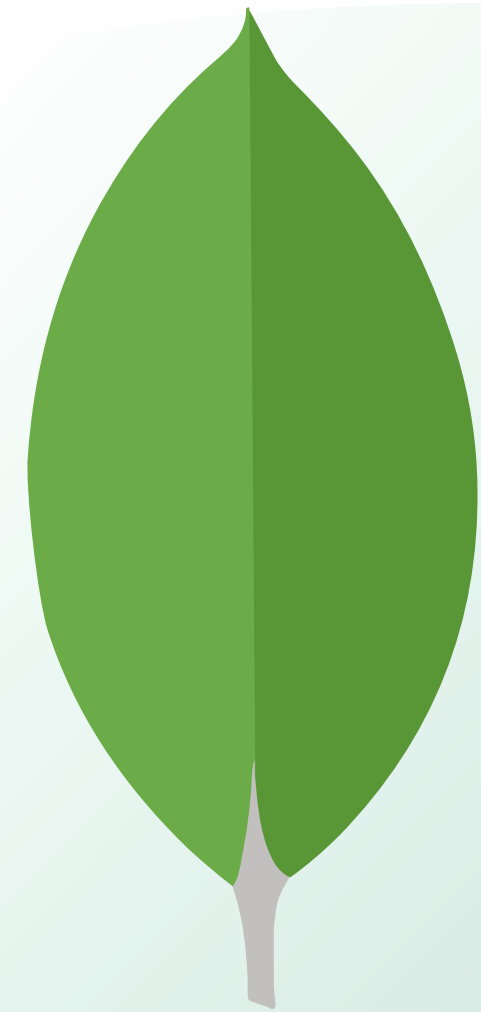


# MEET MONGODB

---

- MongoDB is a **document-oriented** database.
- It is a **NoSQL database**, meaning it does not use a traditional relational database structure.
- MongoDB is **schema-less**, meaning that you do not need to define the structure of your data in advance.
- MongoDB is **highly scalable** and can be used to store large amounts of data.



# MEET MONGODB

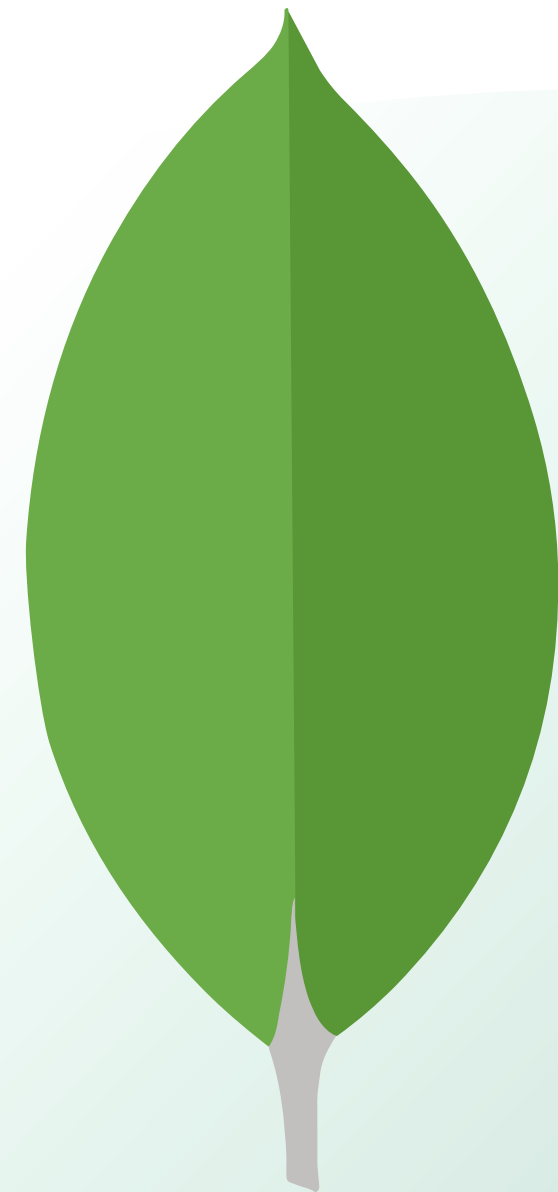
---

## MongoDB can manage

- Structured data
- Semi structured data
- Unstructured data

## NoSQL Databases

- No table
- No row
- No complex join



# MEET MONGODB

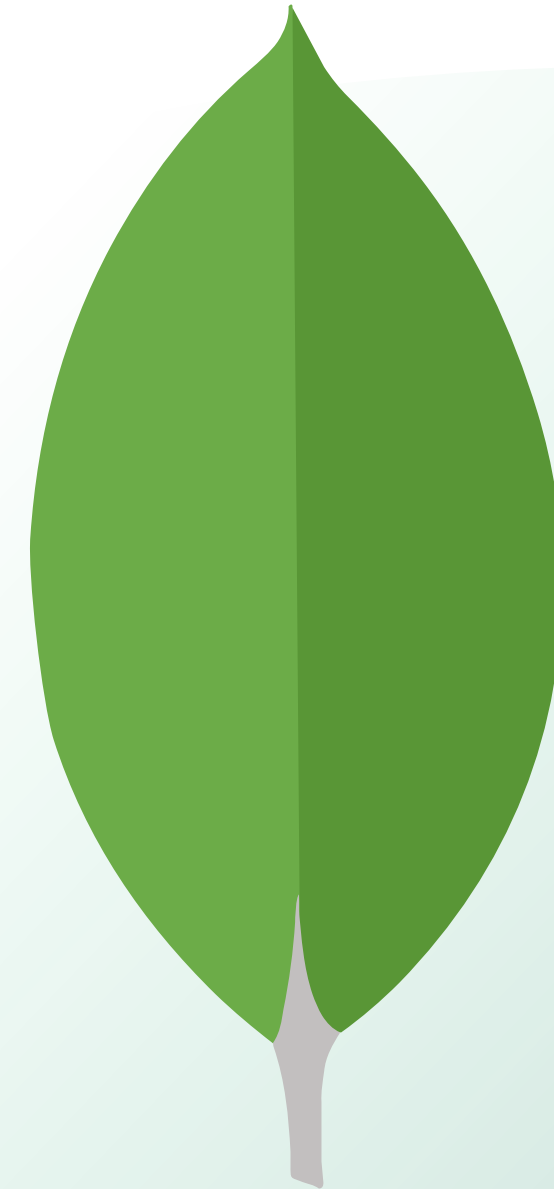
---

## Advantages

- Schema less
- single object
- No complex joins
- Deep query-ability
- Tuning
- Ease of scale-out
- Uses internal memory for storing

## Where to Use MongoDB

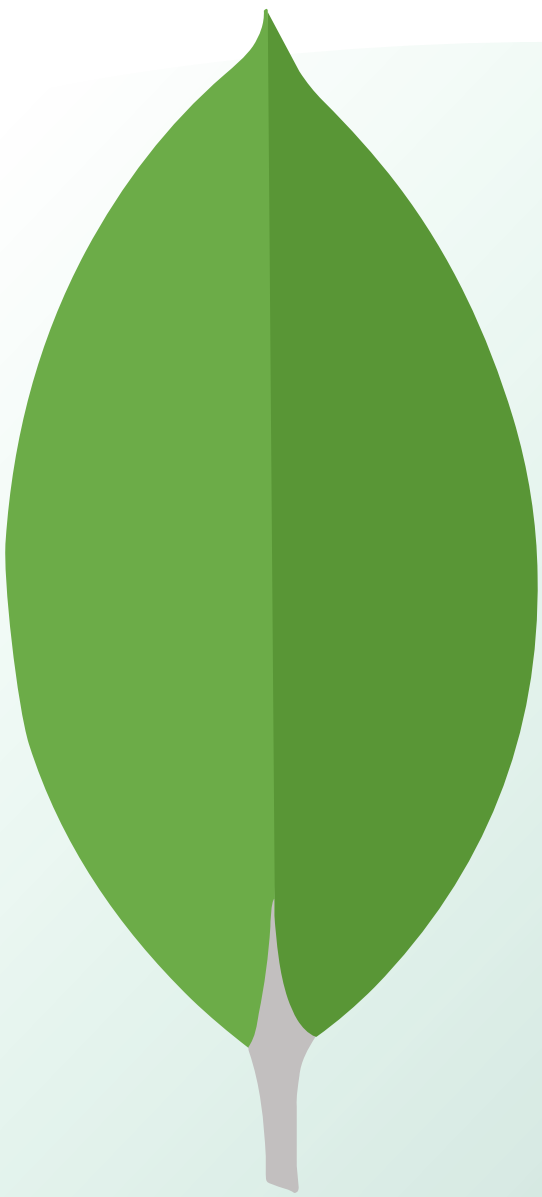
- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub



# BASIC TERMINOLOGY

---

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Default key _id provided by MongoDB itself)



# EMBEDDED DATA MODEL

---

- In this model, you can have (embed) all the related data in a single document
- it is also known as de-normalized data model.

```
{
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details:{
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
  },
  Contact: {
    e-mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
  },
  Address: {
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana"
  }
}
```

# NORMALIZED DATA MODEL

---

In this model, you can refer the sub documents in the original document.

```
{
  _id: <ObjectId101>,
  Emp_ID: "10025AE336",
}
```

```
{
  _id: <ObjectId104>,
  empDocID: " ObjectId101",
  city: "Hyderabad",
  Area: "Madapur",
  State: "Telangana"
}
```

```
{
  _id: <ObjectId103>,
  empDocID: " ObjectId101",
  e-mail: "radhika_sharma.123@gmail.com",
  phone: "9848022338",
}
```

```
{
  _id: <ObjectId102>,
  empDocID: " ObjectId101",
  First_Name: "Radhika",
  Last_Name: "Sharma",
  Date_Of_Birth: "1995-09-26",
}
```

# DATATYPES

Data Types	Description
<b>String</b>	String is the most commonly used datatype. It is used to store data. A string must be UTF 8 valid in <a href="#">mongodb</a> .
<b>Integer</b>	Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using.
<b>Boolean</b>	This datatype is used to store boolean values. It just shows YES/NO values.
<b>Double</b>	Double datatype stores floating point values.
<b>Min/Max Keys</b>	This datatype compare a value against the lowest and highest bson elements.
<b>Arrays</b>	This datatype is used to store a list or multiple values into a single key.
<b>Object</b>	Object datatype is used for embedded documents.
<b>Null</b>	It is used to store null values.
<b>Symbol</b>	It is generally used for languages that use a specific type.
<b>Date</b>	This datatype stores the current date or time in Unix time format. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it.



# SAMPLE DOCUMENT

- ❏ \_id is a 12 bytes hexadecimal number
- ❏ First 4 bytes = current timestamp
- ❏ Next 3 bytes = machine id
- ❏ Next 2 bytes = process id
- ❏ Last 3 bytes = incremental VALUE

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'value',
  tags: ['value', 'value', 'value'],
  likes: 100,
  comments: [
    {
      user: 'value',
      message: 'value',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'value',
      message: 'value',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```



# MONGODB ATLAS

- ❏ MongoDB Atlas is a cloud-based database service
- ❏ That makes it easy to deploy, manage, and scale MongoDB databases
- ❏ Atlas is a fully managed service, so you don't need to worry about the underlying infrastructure
- ❏ Atlas is available on a variety of cloud providers, including AWS, Azure, and Google Cloud Platform



# INSTALL MONGODB ENVIRONMENT

**Download & Install MongoDB Community Server**

**Download & Install MongoDB Compass**

**Connect Compass With Server**



# LOOK INSIDE

## MongoDB Compass

The screenshot displays the MongoDB Compass application window titled "MongoDB Compass - localhost:27017/school.students". The interface is divided into several sections:

- Left Sidebar:** Contains a tree view of the database structure. It shows "localhost:27017" with a search bar and a list of databases: "admin", "config", "local", and "school". The "school" database is expanded, showing a collection named "students".
- Top Bar:** Displays the current connection "localhost:27017" and the selected collection "Documents school.students".
- Main Panel:**
  - Collection Name:** "school.students" is prominently displayed, with "5 DOCUMENTS" and "1 INDEXES" shown to its right.
  - Tabs:** Below the collection name are tabs for "Documents", "Aggregations", "Schema", "Explain Plan", "Indexes", and "Validation". The "Documents" tab is active.
  - Query Bar:** Features a "Filter" button, a dropdown menu, and a text input field containing the query "{ field: 'value' }". It includes "Reset", "Find", and "More Options" buttons.
  - Actions:** Below the query bar are buttons for "ADD DATA" and "EXPORT COLLECTION".
  - Document List:** A list of documents is shown, with the first four visible:
    - Document 1: `{ "_id": ObjectId('645e34822f54fe1b177cfa87') }`
    - Document 2: `{ "_id": ObjectId('645e7efb9db46618e0515458'), "name": "Rabbil", "city": "Dhaka" }`
    - Document 3: `{ "_id": ObjectId('645e7f1a02144b9a5b5ac10e'), "name": "Rabbil", "city": "Dhaka" }`
    - Document 4: `{ "_id": ObjectId('645e7f2e23d560ac4ac3c0b0'), "name": "Rabbil" }`
- Bottom Bar:** A dark bar at the bottom left shows the command prompt ">\_MONGOSH".


# MONGODB

## Dedicated VS Code Extension






# LOOK INSIDE



## MongoDB VS Code Extension



### MongoDB for VS Code v1.0.1

MongoDB  [mongodb.com](https://mongodb.com) |  864,383 |  (30)


Connect to MongoDB and Atlas directly from your VS Code environment, navigate your databases and collections, inspect your

[Reload Required](#) [Disable](#) [Uninstall](#)  


This extension is enabled globally.

[DETAILS](#) [FEATURE CONTRIBUTIONS](#) [CHANGELOG](#) [RUNTIME STATUS](#)

## MongoDB for VS Code

 Test and Build passing


MongoDB for VS Code makes it easy to work with your data in MongoDB directly from your VS Code environment. MongoDB for VS Code is the perfect companion for [MongoDB Atlas](#), but you can also use it with your self-managed MongoDB instances.




MongoDB for VS Code

# Get Started

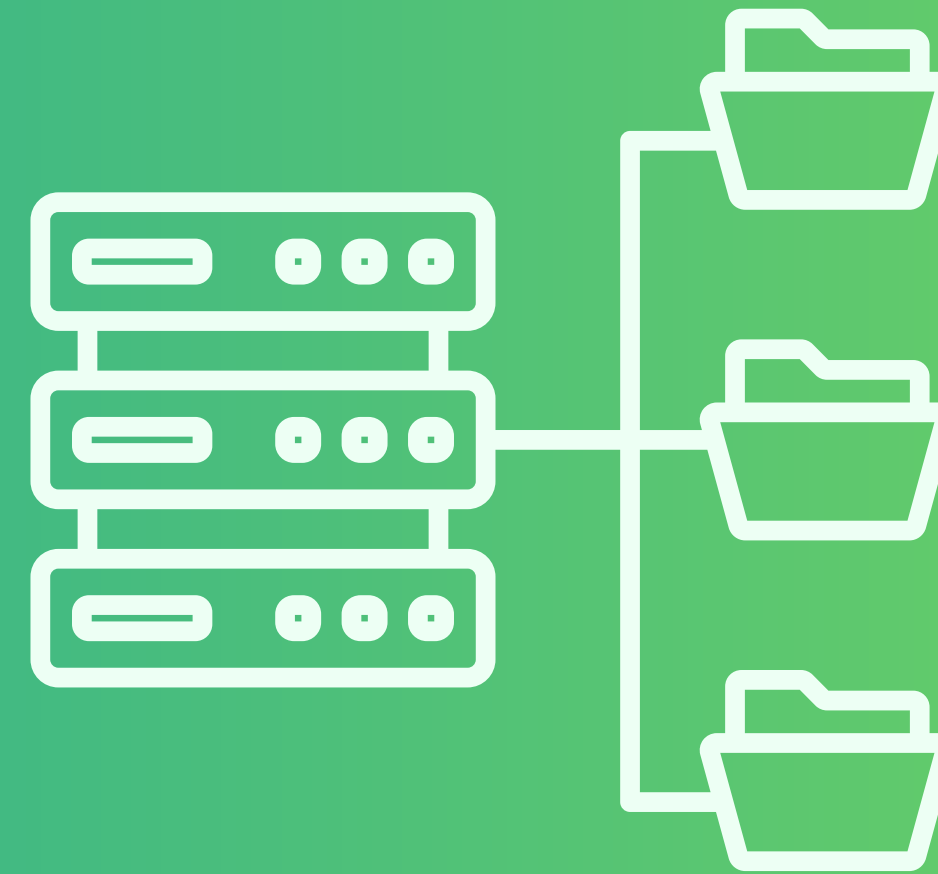
Watch the demo video





# LETS START

## MongoDB Query Writing



# DATABASE RELATED METHODS

---

## **db.help()**

Common db objects. Displays method descriptions

## **db.hostInfo()**

Returns a document record containing information about the operating system MongoDB is running on.

## **db.getName()**

Returns the name of the current database.

## **db.getMongo()**

Returns the name of the current database.

## **db.currentOp()**

Reports the current in-progress operations.



# DATABASE RELATED METHODS

---

## **db.dropDatabase()**

Removes the current database.

## **db.getCollectionInfos()**

Returns collection information for all collections in the current database.

## **db.getCollectionNames()**

Lists all collections in the current database.

## **db.getLastError()**

Checks and returns the status of the last operation

## **db.getLastErrorObj()**

Returns the status document for the last operation.

# DATABASE RELATED METHODS

---

## **db.isMaster()**

Returns a document object containing a status report for the replica set.

## **db.getReplicationInfo()**

Returns statistics for a replica set.

## **db.listCommands()**

Displays a list of common database commands.

## **db.logout()**

Terminates an authenticated session.

## **db.printCollectionStats()**

Display statistics for each collection.

# DATABASE RELATED METHODS

---

## **db.serverBuildInfo()**

Returns a mongod documentation record containing the compile parameters for the instance.

## **db.serverStatus()**

Returns a document that provides an overview of the state of the database process.

## **db.shutdownServer()**

Cleanly and safely stops the current mongod or mongos process.

## **db.stats()**

Returns a document record containing a report on the current database state.

## **db.version()**

Returns the version number of the mongod instance:

# DATABASE RELATED METHODS

---

## **db.createCollection('demo')**

Returns a mongod documentation record containing the compile parameters for the instance.

## **db.CollectionName.drop()**

It completely removes a collection from the database and does not leave any indexes associated with the dropped collections

# INSERT QUERY

---

The **insertOne()** method allows you to insert a single document into a collection.

```
db.employee.insertOne(  
  {  
    "name": "Nafis",  
    "designation": "Manager",  
    "salary": 95000,  
    "city": "Dhaka"  
  }  
)
```

# INSERT QUERY

---

The **insertMany()** allows you to insert multiple documents into a collection.

```
db.brands.insertMany(  
  [  
    {"Name": "IBM"},  
    {"Name": "BP"},  
    {"Name": "UPS"},  
    {"Name": "BMW"}  
  ]  
)
```

# FIND QUERY

---

The findOne() returns a single document from a collection that satisfies the specified condition.

The find() method finds the documents that satisfy a specified condition and returns a cursor to the matching documents

```
db.brands.find();
```

```
db.brands.findOne(  
  { 'Name': 'Walton' }  
);
```



# PROJECTION

---

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document.

```
db.employee.find(  
    {},  
    {"_id":0,"designation":0}  
)
```

# COMPARISON QUERY OPERATOR

---

\$eq: Equal To Operator

\$lt: Less Than Operator

\$lte: Less Than or Equal To Operator

\$gt: Greater Than Operator

\$gte: Greater Than or Equal To Operator

\$ne: Not Equal To Operator

\$in: In Operator

\$nin: Not In Operator

```
db.employee.find(  
  {  
    salary: {$eq: 35000}  
  }  
)
```

# LOGICAL QUERY OPERATOR

---

\$and: Logical AND Operator

\$or: Logical OR Operator

\$not: Logical NOT operation

\$nor: Logical NOR operation

```
db.employee.find( {  
  $and: [  
    {salary:{$gte:35000}},  
    {name:{$eq:'Rupom'}},  
  ]  
})
```

```
db.employee.find( {  
  $or: [  
    {salary:{$gte:35000}},  
    {name:{$eq:'Rabbil'}},  
  ]  
})
```

# ELEMENT QUERY OPERATOR

\$exists: Matches documents that have the specified field.

\$type: Selects documents if a field is of the specified type.

Type	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"
ObjectId	7	"objectId"
Boolean	8	"bool"
Date	9	"date"
Null	10	"null"
Regular Expression	11	"regex"
32-bit integer	16	"int"
Timestamp	17	"timestamp"
64-bit integer	18	"long"
Decimal128	19	"decimal"

# ELEMENT QUERY OPERATOR

---

```
db.employee.find(  
  {  
    address: { $exists: true }  
  }  
)
```

```
db.employee.find(  
  {  
    salary: { $type: 2 }  
  }  
)
```

# EVALUATION QUERY OPERATOR

---

\$expr

Allows use of aggregation expressions within the query language.

\$jsonSchema

Validate documents against the given JSON Schema.

\$mod

Performs a modulo operation on the value of a field and selects documents with a specified result.

\$regex

Selects documents where values match a specified regular expression.

\$text

Performs text search.

\$where

Matches documents that satisfy a JavaScript expression.

# EVALUATION QUERY OPERATOR

---

```
db.monthlyBudget.find(  
  {  
    $expr:  
    {  
      $gt: [ "$budget" , "$spent" ]  
    }  
  }  
)
```

```
db.monthlyBudget.find(  
  { spent: { $mod: [ 2, 0 ] } }  
)
```



# EVALUATION QUERY OPERATOR

---

```
db.employee.find(
  { name: { $regex: "(?i)R(?-i)abb" } }
)
```

```
db.monthlyBudget.find(
  { $where: "this.budget>this.spent" }
)
```

# SORT LIMIT DISTINCT & ROW COUNT

---



```
db.employee.find().sort({salary:-1})
```



```
db.employee.find().count('total')
```



```
db.employee.find().limit(2)
```



```
db.brands.distinct( "Name")
```

# DELETE

---

```
db.employee.deleteOne(  
  {  
    "_id" : ObjectId("646b1073731f226d9c806daf")  
  }  
)
```

```
db.employee.deleteMany({  
  salary:{$gt:30000}  
})
```

# UPDATE ONE & MANY

---

```
db.monthlyBudget.updateMany(
  { budget: {$eq:100}},
  {
    $set: {
      budget: 105
    }
  })
```

```
db.monthlyBudget.updateOne(
  { _id:ObjectId('646b2b64f34dfc345beacdf0')},
  {
    $set: {
      budget: 450
    }
  })
```

# INCREMENT & DECREMENT

---

```
db.monthlyBudget.updateMany(
  { budget: {$eq:105}},
  {
    $inc: {
      budget: 5
    }
  }
})
```

```
db.monthlyBudget.updateMany(
  { budget: {$eq:110}},
  {
    $inc: {
      budget: -5
    }
  }
})
```

# UNSET & RENAME

---

```
db.monthlyBudget.updateMany(
  { budget: {$eq:105}},
  {
    $unset: {
      budget: ""
    }
  })
```

```
db.monthlyBudget.updateMany({}, {
  $rename: {
    spent: "expense"
  }
})
```

# UPSERT

---

Upsert is a combination of update and insert. Upsert performs two functions:

- Update data if there is a matching document.
- Insert a new document in case there is no document matches the query criteria.

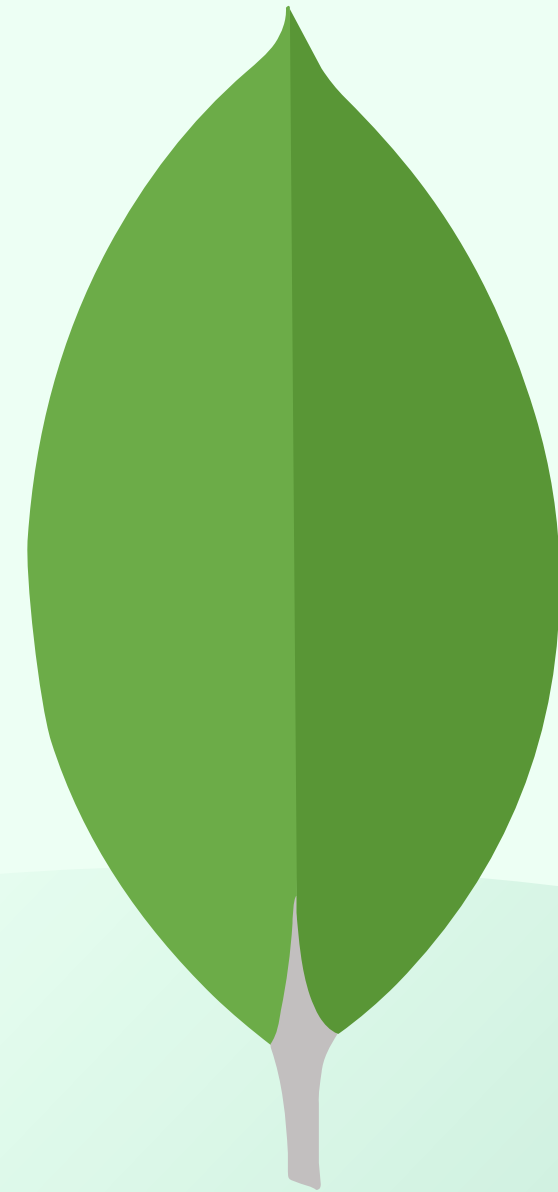
```
db.monthlyBudget.updateOne(
  {category: "Soft Drinks"},
  {
    $set:{
      "category": "Soft Drinks",
      "budget": 4,
      "expense": 450
    }
  },
  { upsert: true }
)
```



# MEET AGGREGATION

---

- Allow you to process multiple documents and return the results.
- To perform aggregation operations, use aggregation pipelines.
- An aggregation pipeline contains one or more stages that process the input documents.



# SELECT, ROW COUNT IN AGGREGATION

```
db.monthlyBudget.aggregate([])
```

```
db.monthlyBudget.aggregate([  
  {$count: 'total'}  
])
```

# SORT LIMIT IN AGGREGATION

---

```
db.monthlyBudget.aggregate([
  {$sort: {
    budget: -1
  }}
])
```

```
db.monthlyBudget.aggregate([
  {$limit: 2}
])
```

# SELECT FIRST 3 & LAST 3 AGGREGATION

---

```
db.monthlyBudget.aggregate([
  {$sort: {
    _id: -1
  }},
  {$limit: 3}
])
```

```
db.monthlyBudget.aggregate([
  {$sort: {
    _id: 1
  }},
  {$limit: 3}
])
```

# CAMPARISON OPERATOR AGGREGATION

---

\$eq: Equal To Operator

\$lt: Less Than Operator

\$lte: Less Than or Equal To Operator

\$gt: Greater Than Operator

\$gte: Greater Than or Equal To Operator

\$ne: Not Equal To Operator

\$in: In Operator

\$nin: Not In Operator

```
db.employee.aggregate([
  {$match: {
    salary: {$gt: 3000}
  }}
])
```

# LOGICAL OPERATOR AGGREGATION

---

\$and: Logical AND Operator

\$or: Logical OR Operator

\$not: Logical NOT operation

\$nor: Logical NOR operation

```
db.employee.aggregate([
  {$match: {
    $and: [
      {salary:{$gte:3000}},
      {name:{$eq:'Rabbil'}}
    ]
  }}
])
```

# ELEMENT QUERY OPERATORS AGGREGATION

\$exists: Matches documents that have the specified field.

\$type: Selects documents if a field is of the specified type.

Type	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"
ObjectId	7	"objectId"
Boolean	8	"bool"
Date	9	"date"
Null	10	"null"
Regular Expression	11	"regex"
32-bit integer	16	"int"
Timestamp	17	"timestamp"
64-bit integer	18	"long"
Decimal128	19	"decimal"

# ELEMENT QUERY OPERATORS AGGREGATION

---

```
db.employee.aggregate([
  {$match: {
    salary:{$type:16}
  }}
])
```

```
db.employee.aggregate([
  {$match: {
    address:{$exists:true}
  }}
])
```