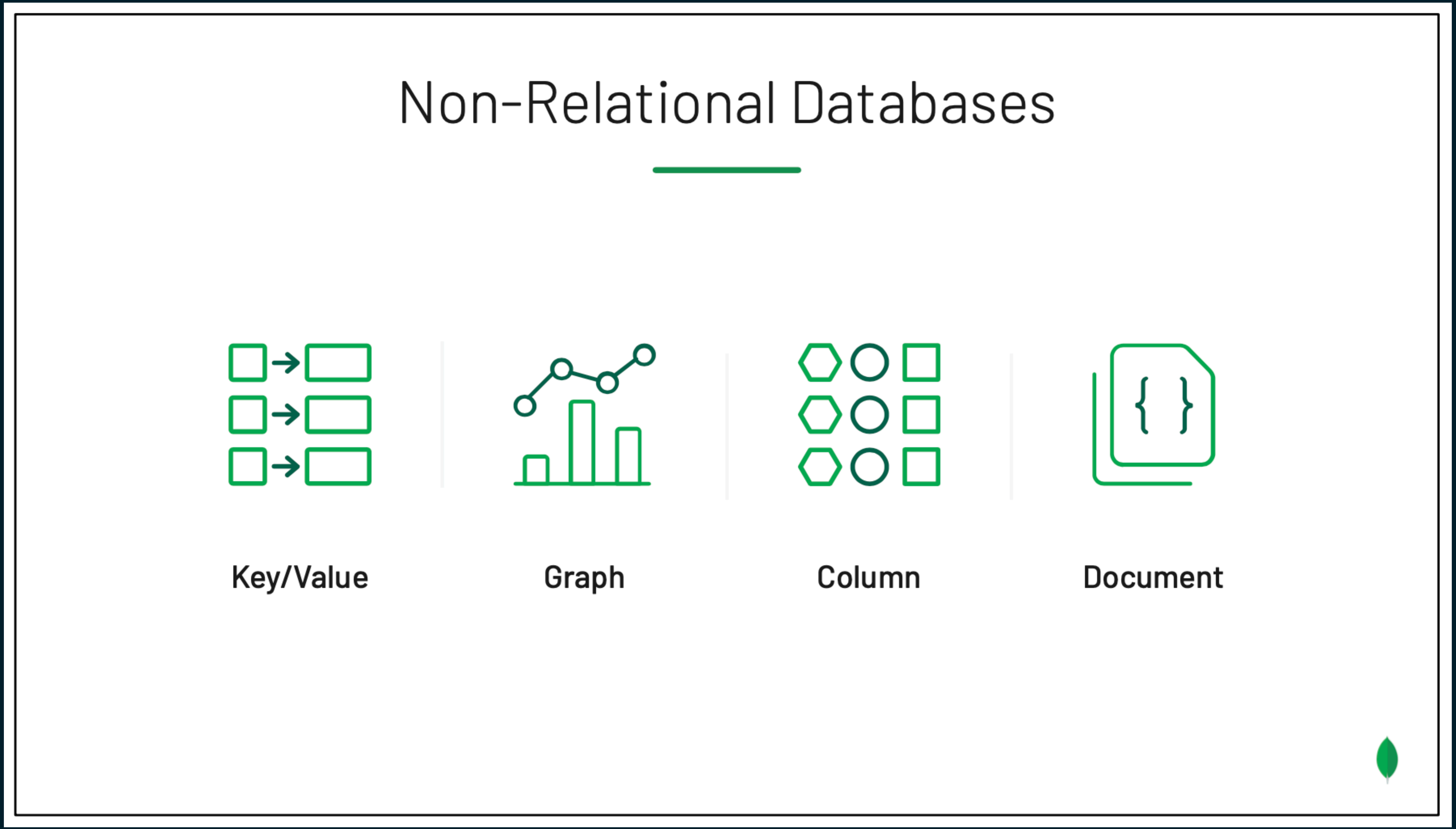
Introduction Non Relational DBs

> “There’s lots of available datasets but data can be in nasty forms… Data needs to be in a form that is easy to access and use.

KALPATHI SUBRAMANIAN
UNIVERSITY OF NORTH CAROLINA, CHARLOTTE



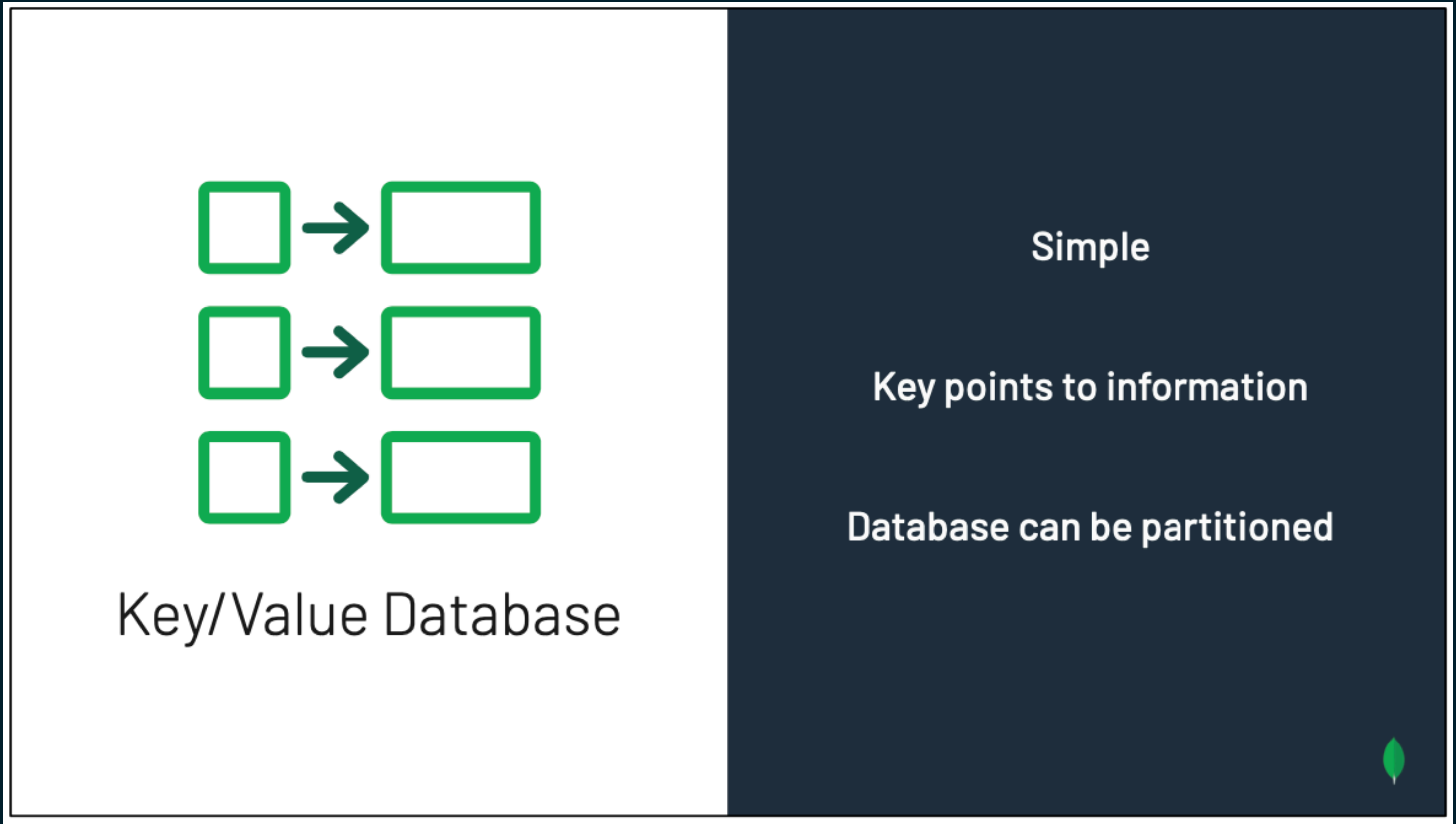
Non-Relational Databases

Key/Value

Graph

Column

Document



Key/Value Database

Simple

Key points to information

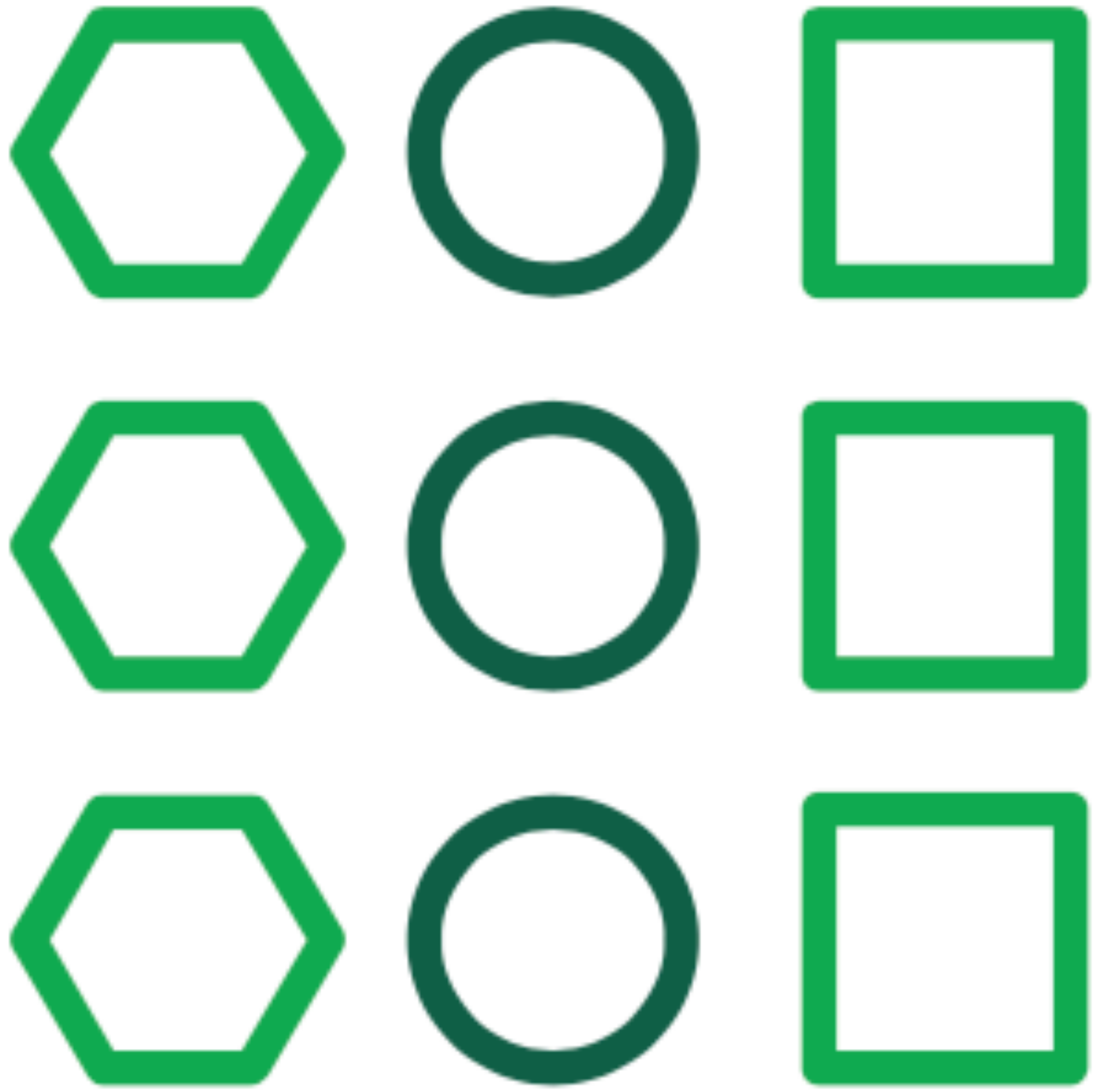
Database can be partitioned

Graph Database

Niche problems

Relations within table

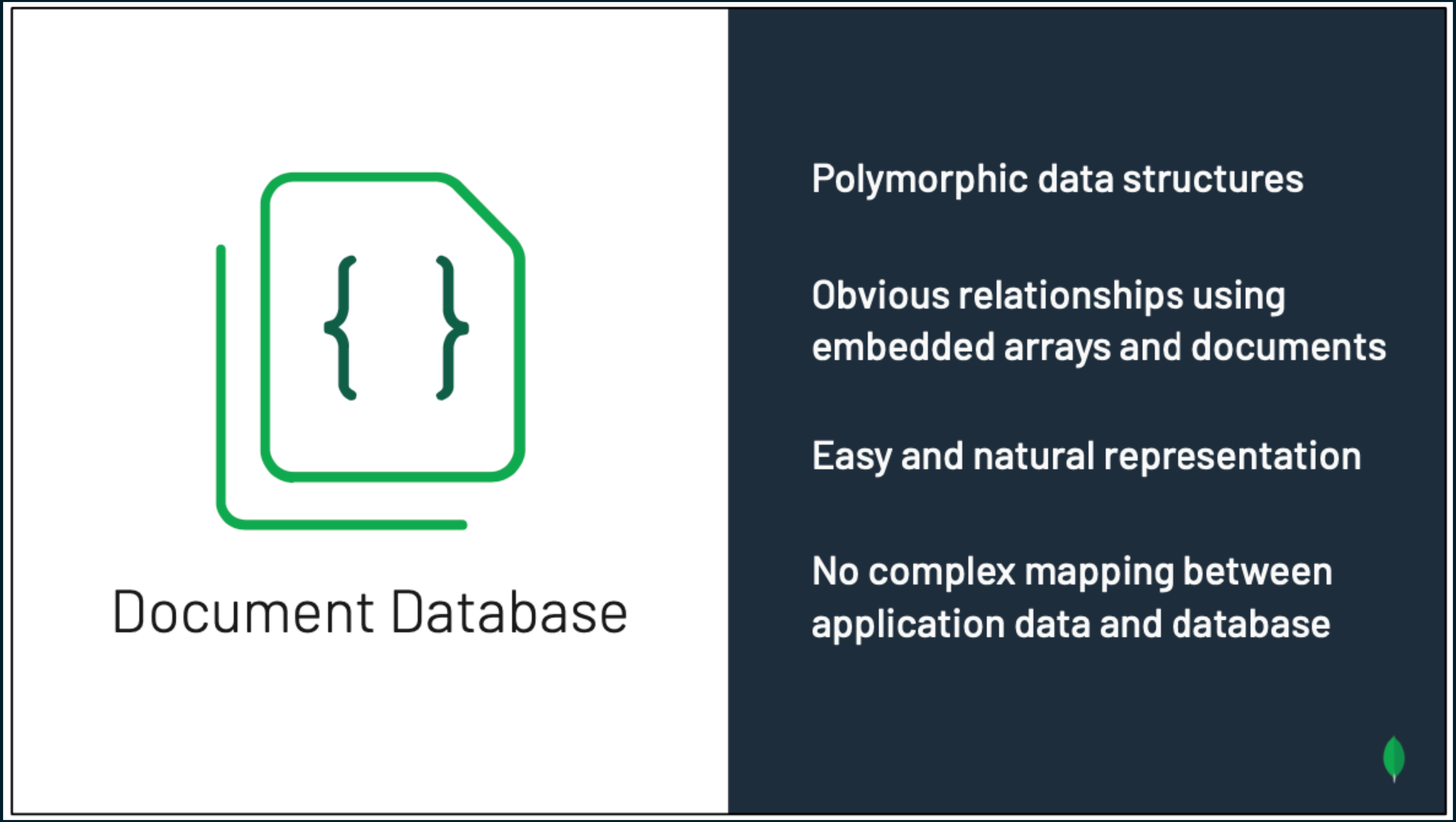
SQL statements with joins



Column Oriented or Wide Column

Data is stored per column

Designed for analytics



Document Database

Polymorphic data structures

Obvious relationships using embedded arrays and documents

Easy and natural representation

No complex mapping between application data and database

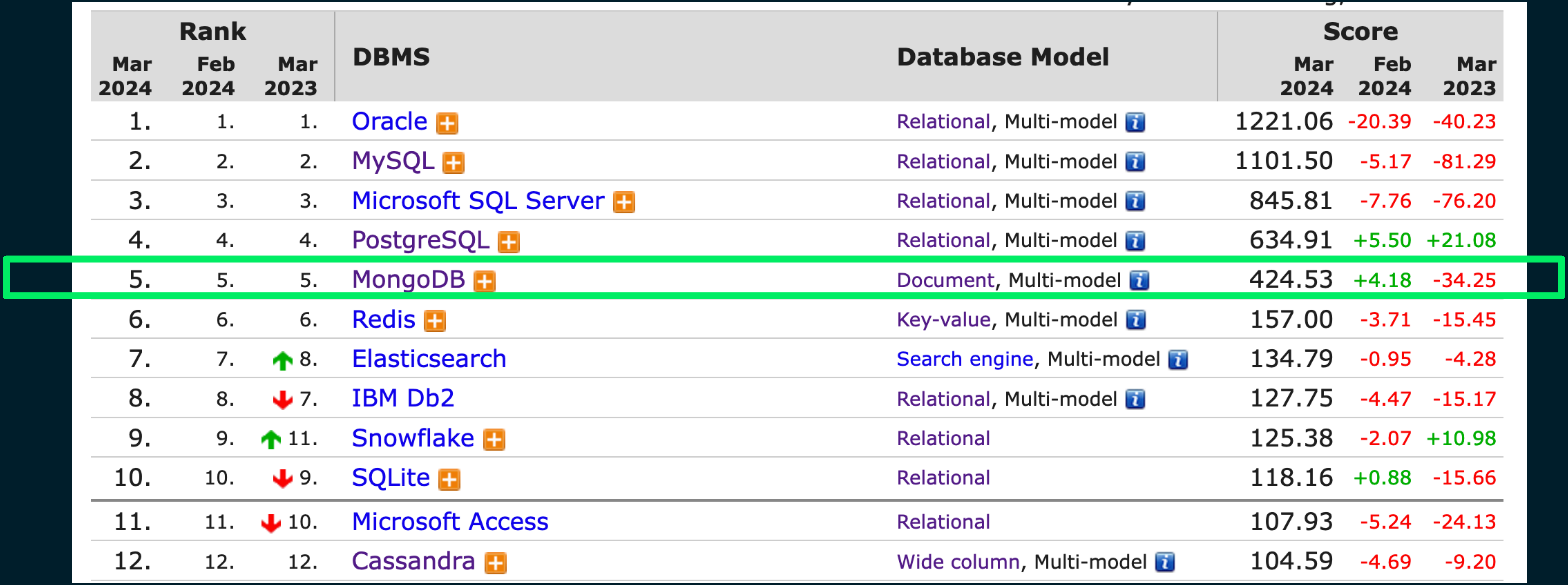
Wide column stores 2.7%
Vector DBMS 0.3%
Time Series DBMS 1.2%
Spatial DBMS 0.5%
Search engines 4.5%
Document stores 10.3%
Graph DBMS 1.7%
Key-value stores 5.3%
Multivalue DBMS 0.2%
Native XML DBMS 0.3%
Object oriented DBMS 0.2%
RDF stores 0.4%
Relational DBMS 72.4%
© 2024, DB-Engines.
Wide column stores 3.1%
Time Series DBMS 1%
Spatial DBMS 0.4%
Search engines 4.7%
Document stores 10%
Graph DBMS 1.8%
Key-value stores 5.6%
Multivalue DBMS 0.2%
Native XML DBMS 0.3%
Object oriented DBMS 0.2%
RDF stores 0.4%
Relational DBMS 72.1%
© 2022, DB-Engines.com

…the "growth" is in non-relational
-- define growth

150
140
130
120
110
100
90
80
Popularity Changes
May 2021
Sep 2021
Jan 2022
May 2022
Sep 2022
Jan 2023
May 2023
Sep 2023
Jan 2024
Vector DBMS
Time Series DBMS
RDF stores
Object oriented DBMS
Key-value stores
Document stores
Native XML DBMS
Multivalue DBMS
Search engines
Spatial DBMS
Relational DBMS
Graph DBMS
Wide column stores

© 2024, DB-Engines.com

Most popular non-relational database



Rank			DBMS	Database Model	Score		
Mar 2024	Feb 2024	Mar 2023			Mar 2024	Feb 2024	Mar 2023
1.	1.	1.	Oracle	Relational, Multi-model	1221.06	-20.39	-40.23
2.	2.	2.	MySQL	Relational, Multi-model	1101.50	-5.17	-81.29
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	845.81	-7.76	-76.20
4.	4.	4.	PostgreSQL	Relational, Multi-model	634.91	+5.50	+21.08
5.	5.	5.	MongoDB	Document, Multi-model	424.53	+4.18	-34.25
6.	6.	6.	Redis	Key-value, Multi-model	157.00	-3.71	-15.45
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	134.79	-0.95	-4.28
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	127.75	-4.47	-15.17
9.	9.	↑ 11.	Snowflake	Relational	125.38	-2.07	+10.98
10.	10.	↓ 9.	SQLite	Relational	118.16	+0.88	-15.66
11.	11.	↓ 10.	Microsoft Access	Relational	107.93	-5.24	-24.13
12.	12.	12.	Cassandra	Wide column, Multi-model	104.59	-4.69	-9.20

Mongo

RDBMS & Mongo

Basic terms

- A set of **databases**
  - each database contains a set of **tables**
    - each table specifies a set of columns
    - each table contains a set of **rows (relations)**
      - each row has exactly the columns specified by the table

- A set of **databases**
  - each database contains a set of **collections**
    - each collection contains a set of **documents**
      - each document is an independent thing
        - Assuming no schema checking

```
{
  "_id": 11,
  "user_id": "Eoin",
  "age": 29,
  "Status": "A"
}
```

Collection

An organized store of documents in MongoDB, usually with common fields between documents

Document

A way to organize and store data as a set of field-value pairs

Collection

An organized store of documents in MongoDB, usually with common fields between documents

Relational

Cars

_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Wheels

_id	partNo
007	234819
007	281928
007	392838
007	928038
008	950555
008	950556
008	950557
008	950558

MongoDB

```
{
 "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  "wheels": [
    { "partNo": 234819 },
    { "partNo": 281928 },
    { "partNo": 392838 },
    { "partNo": 928038 }
  ],
  ...
}
```

Relational

Cars

_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Wheels

_id	partNo
007	234819
007	281928
007	392838
007	928038
008	950555
008	950556
008	950557
008	950558

MongoDB

```
{
 "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  "wheels": [
    { "partNo": 234819 },
    { "partNo": 281928 },
    { "partNo": 392838 },
    { "partNo": 928038 }
  ],
  ...
}
```

Car in
Relational Database

Car in
MongoDB

Rich, Flexible Document Model -- it is just JSON (plus some)

```
{
  "_id": ObjectId("573a1390f29313caabcd4135"),
  "title": "Blacksmith Scene",
  "plot": "Three men hammer on an anvil and pass ...",
  "cast": [ "Charles Kayser", "John Ott" ],
  "directors": [ "William K.L. Dickson" ],
  "lastupdated": "2015-08-26 00:03:50.133000000",
  "year": 1893,
  "imdb": {
     "rating": 6.2,
     "votes": 1189,
     "id": 5
  }
}
```

Internally documents are stored in BSON (Binary JSON)

JSON	BSON
Text Encoding	Binary Encoding
Human Readable	Machine Readable
Slower Parsing	Fast Parsing
Basic Data Types	Advanced Data Types
Not as efficient	Efficient

JSON-like

Extends json

- JSON has only:
  - String, number, boolean, null (and object, array)
- Mongo adds
  - integers (4 or 8 byte)
    - the default is float
      - {"x":NumberInt("3"), "y":NumberLong("4"), "z":5}
  - Date:
      - {"d":new Date()}
  - ObjectID
    - a special 12 byte thing (every document in Mongo has an ObjectID)

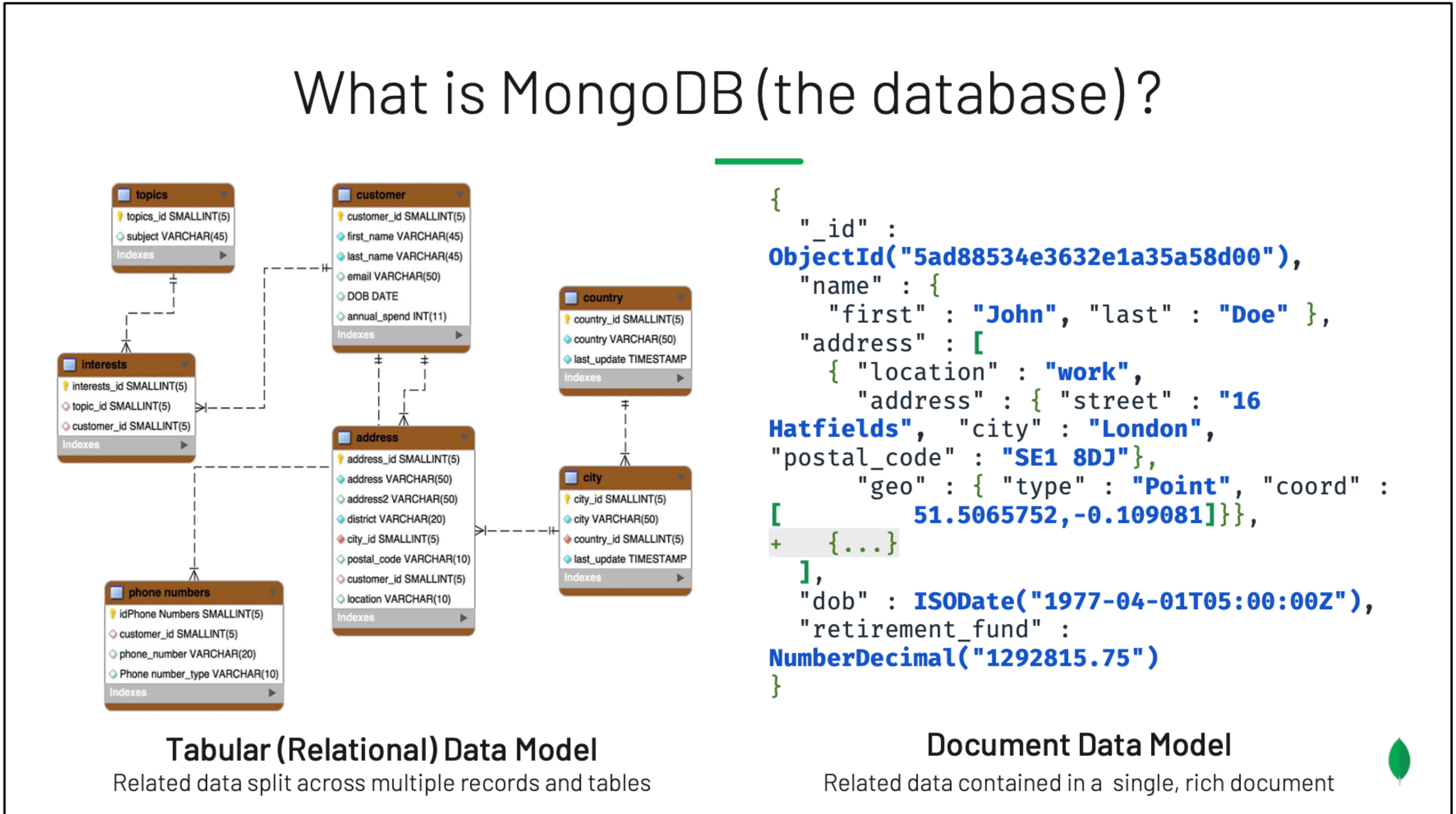
Mongo has polymorphic data

- Polymorphic data means that in one collection you have many versions of document schema
  - so, when you create a collection, you just put data in.
    - {"_id": 123, "car":"ferrari", "Cylinders":8, "cid":400, "hp":450}
    - {"_id":123, "car":"Tesla", "hp":300}
    - Some items are missing fields
      - In RDBMS null -- Mongo -- simply not there!

DB - PL mapping

- Since all Mongo data is in JSON-like container, mapping into objects is fairly natural.
    - If you have data in PostgreSQL
        - export table as JSON.
            - `select json_agg(t) from (select * from TABLE) as t;`
            - What is missing??

Document-Oriented Data



What is MongoDB (the database)?

topics
- topics_id SMALLINT(5)
- subject VARCHAR(45)
- Indexes

customer
- customer_id SMALLINT(5)
- first_name VARCHAR(45)
- last_name VARCHAR(45)
- email VARCHAR(50)
- DOB DATE
- annual_spend INT(11)
- Indexes

country
- country_id SMALLINT(5)
- country VARCHAR(50)
- last_update TIMESTAMP
- Indexes

interests
- interests_id SMALLINT(5)
- topic_id SMALLINT(5)
- customer_id SMALLINT(5)
- Indexes

address
- address_id SMALLINT(5)
- address VARCHAR(50)
- address2 VARCHAR(50)
- district VARCHAR(20)
- city_id SMALLINT(5)
- postal_code VARCHAR(10)
- customer_id SMALLINT(5)
- location VARCHAR(10)
- Indexes

city
- city_id SMALLINT(5)
- city VARCHAR(50)
- country_id SMALLINT(5)
- last_update TIMESTAMP
- Indexes

phone numbers
- idPhone Numbers SMALLINT(5)
- customer_id SMALLINT(5)
- phone_number VARCHAR(20)
- Phone number_type VARCHAR(10)
- Indexes

```
{
   "_id" :
ObjectId("5ad88534e3632e1a35a58d00"),
   "name" : {
      "first" : "John", "last" : "Doe" },
   "address" : [
      { "location" : "work",
         "address" : { "street" : "16
Hatfields",  "city" : "London",
"postal_code" : "SE1 8DJ"},
         "geo" : { "type" : "Point", "coord" :
[        51.5065752,-0.109081]}},
+     {...}
   ],
   "dob" : ISODate("1977-04-01T05:00:00Z"),
   "retirement_fund" :
NumberDecimal("1292815.75")
}
```

Tabular (Relational) Data Model

Related data split across multiple records and tables

Document Data Model

Related data contained in a single, rich document

Object/sub-document: a one-to-one relationship

Cars		
_id	owner	make
007	Daniel	Ferrari
Q08	Daniel	Fiat

Engines			
_id	car_id	power	consumption
234808	007	660	10
Q08	008	120	45

OR

Cars				
_id	owner	make	power	consumption
007	Daniel	Ferrari	660	10
Q08	Daniel	Fiat	120	45

```
{
  "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  "engine": {
       "power": 660hp,
       "consumption": 10mpg
  }
  …
}
```

Tabular (Relational) Data Model

A car as one Engine. A one-to-one relationship in a single document or across 2 documents

Document Data Model

The engine information is in its own structure in the parent entity

Array: a one-to-many

Cars		
_id	owner	make
007	Daniel	Ferrari
008	Daniel	Fiat

Wheels	
_id	car_id
234819	007
281928	007
392838	007
928038	007
950555	008

Tabular (Relational) Data Model

One-to-Many relationship
from a car to the its wheels

```
{
  "_id": 007,
  "owner": "Daniel",
  "make": "Ferrari",
  wheels: [
       { "partNo": 234819 },
       { "partNo": 281928 },
       { "partNo": 392838 },
       { "partNo": 928038 }
  ],
  ...
}
```

Document Data Model

One-to-Many wheels
expressed as an array

Many-to-Many relations

- Consider Sakila DB and actors
  - the actor table has information about the actor (name, etc)
  - the film_actor table has info showing what what films an actor was in but to get the names of those files you need to do a join.

- Mongo documents does not model this well.
- Think hard about the data .. do I need to allow querying from both directions??
  - If yes, then best course is accept the duplication of data
    - represent many explicitly in in each document

```
Person Collection
                 {ID:1, Name:"Rachel", advisees:[2,3,4,5], teaches:[1,2]}
                 {ID:10, Name:"Angie", advisees:[2,12,22], teaches:[2,3]}
                 {ID:2, Name:"Sarah", advisors:[1,10], takes:[1,2]}
                 {ID:3, Name:"Femi", advisors[1], takes:[2,4]}

Section Collection
        {section:1, dept:"A", course:123, section: 1 instructor:[1], students:[2]}
```

When to use which?

SQL is a good match for structured, slowly changing data

Non-relational, particularly the document model, is well suited to polymorphic data that can change frequently

Non-relational can provide greater developer productivity as it requires less code to translate between the database and the application

Non-relational systems are cloud-native and designed as distributed systems

Using Mongo

- UNIX> mongosh

- test==>**show dbs**
- test==>**use sakila**
    - switch to the sakila database
    - create the database if it does not exist
- sakila==>**show collections**

- **exit**

Mongosh
part 2

```
test==> use geoff
switched to db geoff
geoff==> db.movies.insertOne( { title: "The Favourite", genres: ["Drama", "History"] } )
{
 acknowledged: true,
 insertedId: ObjectId('65f0e9666407a274f80f71d5')
}
geoff==> db.movies.insertMany( [ {title: "Poor things"}, {genres: ["drama", "Fantasy"] } ] )
{
 acknowledged: true,
 insertedIds: {
  '0': ObjectId('65f0e9d66407a274f80f71d6'),
  '1': ObjectId('65f0e9d66407a274f80f71d7')
 } }
geoff==> db.movies.find({}) // select * from table;
[ {
   _id: ObjectId('65f0e9666407a274f80f71d5'),
   title: 'The Favourite',
   genres: [ 'Drama', 'History' ]
 },
 { _id: ObjectId('65f0e9d66407a274f80f71d6'), title: 'Poor things' },
 {
   _id: ObjectId('65f0e9d66407a274f80f71d7'),
   genres: [ 'drama', 'Fantasy' ]
 }]
geoff==>
```

If the collection 'movies' does not exist, it wll be created in the geoff db

The things inserts are not the same!