SECTION 11 — MODELLING DATA AND ADVANCED MONGOOSE

"DATA... WHAT? 🤔"

DATA MODELLING

Real-world scenario

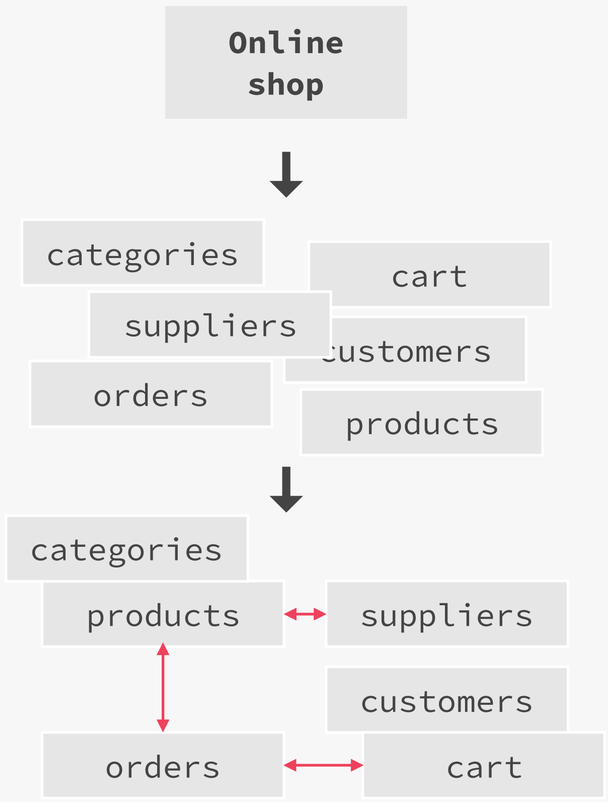
↓

Unstructured data

↓

Structured, logical data model

Example 👇



1. Different types of **relationships** between data
2. **Referencing**/normalization vs. **embedding**/denormalization
3. **Embedding** or **referencing** other documents?
4. **Types** of referencing

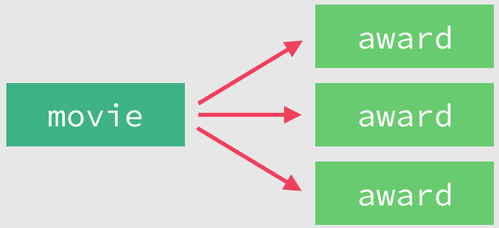
1. TYPES OF RELATIONSHIPS BETWEEN DATA

1:1

*(**1** movie can only have **1** name)*

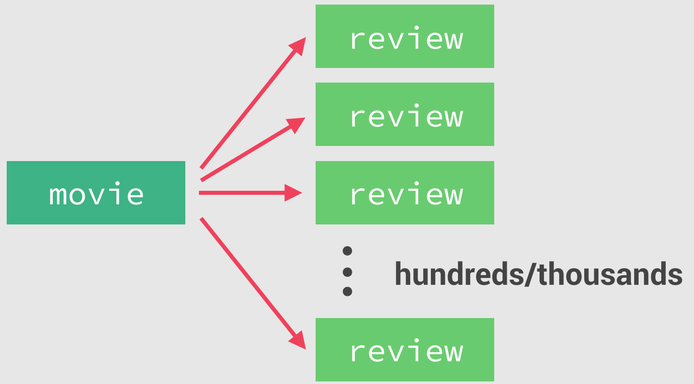
1:MANY

👉 **1:FEW**

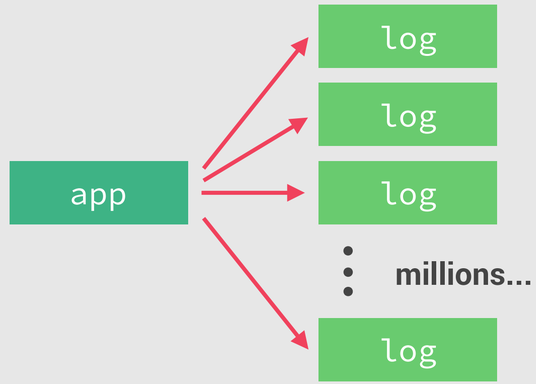


*(**1** movie can win **many** awards)*

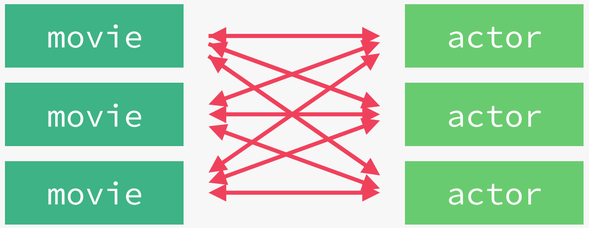
👉 **1:MANY**



👉 **1:TON**



MANY:MANY



*(One movie can have **many** actors, but one actor can also play in **many** movies)*

2. REFERENCING VS. EMBEDDING

REFERENCED / NORMALIZED

movie

```
{
  "_id": ObjectID('222'),
  "title": "Interstellar",
  "releaseYear": 2014,
  "actors": [
    ObjectID('555'),
    ObjectID('777')
  ]
}
```

Referencing (child)

actor

```
{
  "_id": ObjectID('555'),
  "name": "Matthew McConaughey",
  "age": 50,
  "born": "Uvalde, USA"
}
```

actor

```
{
  "_id": ObjectID('777'),
  "name": "Anne Hathaway",
  "age": 37,
  "born": "NYC, USA"
}
```

EMBEDDING/
DENORMALIZATION

REFERENCING /
NORMALIZATION

EMBEDDED / DENORMALIZED

```
{
  "_id": ObjectID('222'),
  "title": "Interstellar",
  "releaseYear": 2014,
  "actors": [
    {
      "name": "Matthew McConaughey",
      "age": 50,
      "born": "Uvalde, USA"
    },
    {
      "name": "Anne Hathaway",
      "age": 37,
      "born": "NYC, USA"
    }
  ]
}
```

Main document

Embedded documents

movie

👍 Performance: it's easier to query each document on its own

👎 We need 2 queries to get data from referenced document

👍 Performance: we can get all the information in one query

👎 Impossible to query the embedded document on its own

3. WHEN TO EMBED AND WHEN TO REFERENCE? A PRACTICAL FRAMEWORK

👉 Combine all 3 criteria to take decision!

	EMBEDDING	REFERENCING
1 RELATIONSHIP TYPE (How two datasets are related to each other)	👉 1:FEW 👉 1:MANY	👉 1:MANY 👉 1:TON 👉 MANY:MANY
2 DATA ACCESS PATTERNS (How often data is read and written. Read/write ratio)	👉 Data is mostly **read** 👉 Data does **not** change quickly 👉 (**High** read/write ratio) `Movies + Images`	👉 Data is **updated** a lot 👉 (**Low** read/write ratio) `Movies + Reviews`
3 DATA CLOSENESS (How "much" the data is related, how we want to query)	👉 Datasets **really** belong together `User + Email Addresses`	👉 We frequently need to query both datasets **on their own** `Movies + Images`

`Movies + Images (100)` ?

4. TYPES OF REFERENCING

CHILD REFERENCING

app

```
{
  "_id": ObjectID('23'),
  "app": "My Movie Database",
  "logs": [
    ObjectID('1'),
    ObjectID('2'),
    // ... Millions of ObjectID
    ObjectID('28273927')
  ]
}
```

2M

log

```
{
  "_id": ObjectID('1'),
  "type": "error",
  "timestamp": 1412184926
}
```

log

```
{
  "_id": ObjectID('28273927'),
  "type": "error",
  "timestamp": 1412844672
}
```

👉 1:FEW

PARENT REFERENCING

app

```
{
  "_id": ObjectID('23'),
  "app": "My Movie Database"
}
```

log

```
{
  "_id": ObjectID('1'),
  "app": ObjectID('23'),
  "type": "error",
  "timestamp": 1412184926
}
```

log

```
{
  "_id": ObjectID('28273927'),
  "app": ObjectID('23'),
  "type": "error",
  "timestamp": 141284...
}
```

BEST SOLUTION

👉 1:MANY

👉 1:TON

TWO-WAY REFERENCING

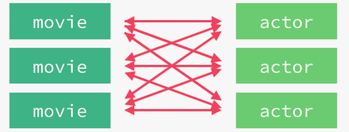
movie

```
{
  "_id": ObjectID('23'),
  "title": "Interstellar",
  "releaseYear": 2014,
  "actors": [
    ObjectID('67'),
    // ... and many more
  ]
}
```

actor

```
{
  "_id": ObjectID('67'),
  "name": "Matthew McConaughey",
  "age": 50,
  "movies": [
    ObjectID('23'),
    // ... and many more
  ]
}
```

👉 MANY:MANY



SUMMARY 🥳

- 👉 The most important principle is: Structure your data to **match the ways that your application queries and updates data**;
- 👉 In other words: Identify the questions that arise from your **application's use cases** first, and then model your data so that the **questions can get answered** in the most efficient way;
- 👉 In general, **always favor embedding**, unless there is a good reason not to embed. Especially on 1:FEW and 1:MANY relationships;
- 👉 A 1:TON or a MANY:MANY relationship is usually a good reason to **reference** instead of embedding;
- 👉 Also, favor **referencing** when data is updated a lot and if you need to frequently access a dataset on its own;
- 👉 Use **embedding** when data is mostly read but rarely updated, and when two datasets belong intrinsically together;
- 👉 Don't allow arrays to grow indefinitely. Therefore, if you need to normalize, use **child referencing** for 1:MANY relationships, and **parent referencing** for 1:TON relationships;
- 👉 Use **two-way referencing** for MANY:MANY relationships.

THE NATOURS DATA MODEL