[639] Non-relational databases: MongoDB

Meenakshi Syamkumar

Learning Objectives

Define	transaction and its properties
Recognize	SQL transaction control commands
Describe	differences between relational and non-relational databases
Identify	correct syntax and structure used in the document model
Define	a collection in the document model
Differentiate	concepts of linking and embedding data

What are transactions?

- Mechanism to group a set of SQL statements such that either all or none get executed (all-or-nothing principle)
- Critical for maintaining integrity of the database
- Key properties of transactions (ACID)
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity

- ensures that all operations in a transaction are completed
- one operation failure leads to transaction abort
- example: banking transaction
  - consider transfer of money from accounts A to B
  - money deduction from account A succeeds
  - money deposit into the account B fails
  - all-or-nothing principle - transaction should be rolled back

Consistency

- ensures that a transaction brings the database from one valid state to another
- constraints (like primary keys, foreign keys, unique constraints) must be maintained
- example: banking constraints
  - consider transfer of $10 from accounts A to B when A’s balance is $5
  - accounts cannot have negative balance

Isolation

- ensures that transactions operate independently
- changes made by one transaction should not be visible to other transactions until the transaction is committed
- example: banking transactions
  - two simultaneous withdrawal of $10 from an account A which has a balance of $10
  - two transactions are updating the same data simultaneously, the changes from one transaction should not interfere with the other until the first transaction is complete

Durability

- ensures that once a transaction is committed, its changes are permanent, even in the case of a system failure
- data is persisted
- example: crash failures
  - If a transaction updates a record and is committed, the changes should survive system crashes or failures

Transaction control commands

- AUTOCOMMIT: ON by default
- BEGIN / START TRANSACTION: starts a new transaction
- COMMIT: saves all changes made during the transaction to the database
- ROLLBACK: reverts the database to the last committed state, undoing all changes made during the transaction
- SAVEPOINT: allows setting a point within a transaction to which you can later roll back
  - SAVEPOINT <savepoint_name>
  - ROLLBACK TO SAVEPOINT <savepoint_name>
  - RELEASE SAVEPOINT <savepoint_name>

Non-relational databases

Database systems timeline

1970s

late 1980s

late 1990s

Relational Databases (Raymond Boyce & Donald Chamberlin)

SQL was developed by IBM as a way to interact with the new relational databases

World Wide Web (Tim Berners-Lee)

Need for data storage explodes

NoSQL (Carlo Strozzi)

Unstructured data storage to mitigate costs and increase efficiency

Non-relational databases

polymorphic data storage support

- structured and semi-structured

schema flexibility

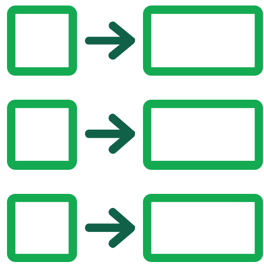
easy horizontal scalability

- vertical scalability aka scaling up: adding more computing resources like CPU, RAM, etc.,
- horizontal scalability aka scaling out: adding more computing machines to your cluster

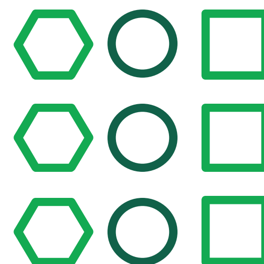
high performance

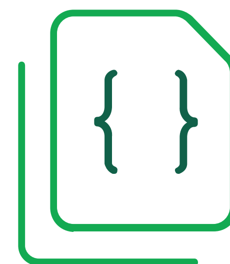
- easily handle large volumes of data (big data analytics)

Non-relational database types


Key / Value

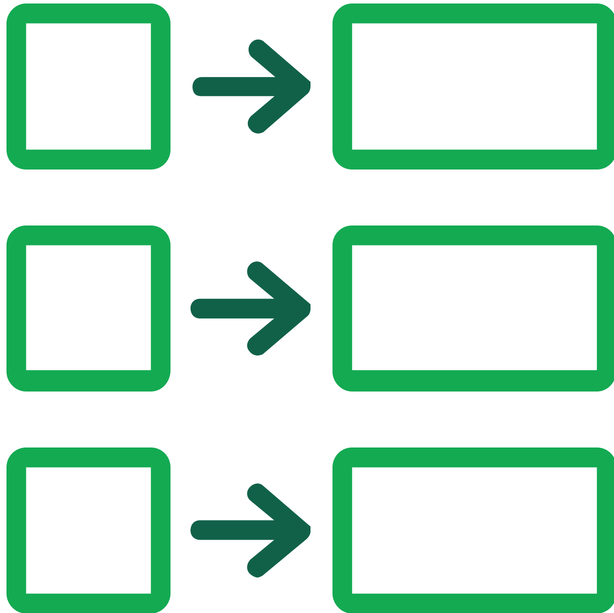
Graph


Column


Document

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
(CC BY-NC-SA 3.0)

Key / Value Database



- Structure
  - A unique key is paired with a collection of values
  - The values can be anything from a string to a large binary object
- Strength
  - Simple data model

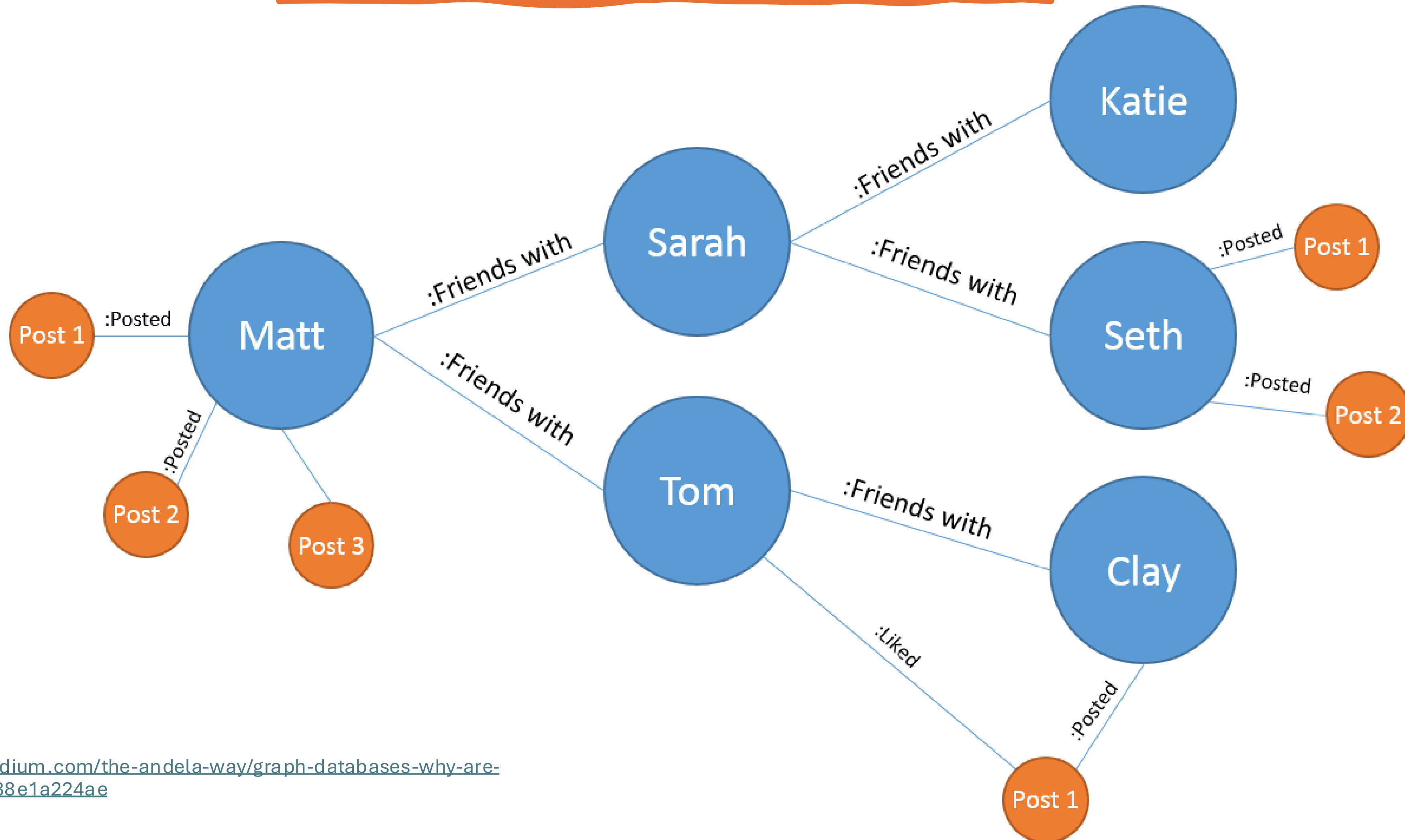
Key / Value Database: Example

Key	Value
user:1001	{ "name": "John Doe", "email": "john.doe@example.com", "age": 27, "city": "Madison" }
user:1002	{ "name": "Jane Smith", "email": "jane.smith@example.com", "age": 25, "city": "Chicago" }
user:1003	{ "name": "Alice Brown", "email": "alice.brown@example.com", "age": 28, "city": "Boston" }

Graph Database

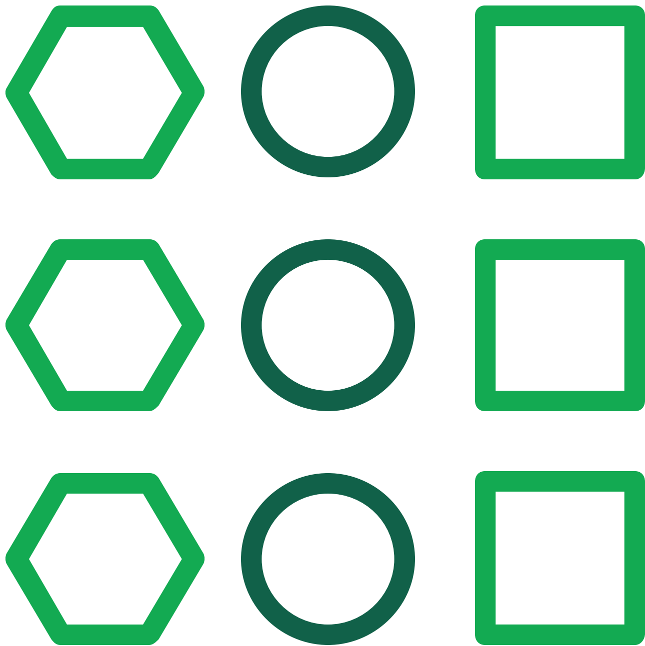
- Structure
  - Captures connected data
  - Each element is stored as a node
  - Connections between nodes are called links or relationships
- Strength
  - Traverses the connections between data rapidly

Graph Database: Example



Source: https://medium.com/the-andela-way/graph-databases-why-are-they-important-c438e1a224ae

Column databases



- Structure
  - Data is stored using key rows that can be associated with one or more dynamic columns
- Strengths
  - Highly performant queries
  - Designed for analytics

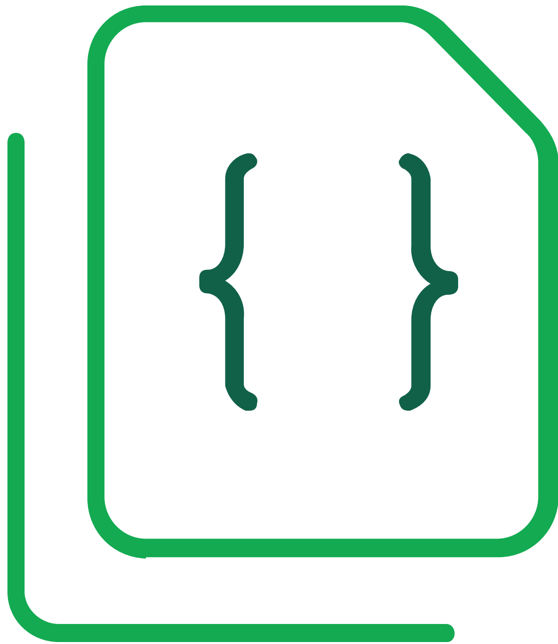
Column databases: Example

Name	ID
Sherlock	001
John	002
Irene	003

Age	ID
40	001
45	002
43	003

Height	ID
6’2	001
5’9	002
5’7	003

Document Database



- Structure
  - Polymorphic data models
  - Each document contains markup that identifies fields and values
- Strengths
  - Obvious relationships using embedded arrays and documents
  - No complex mapping

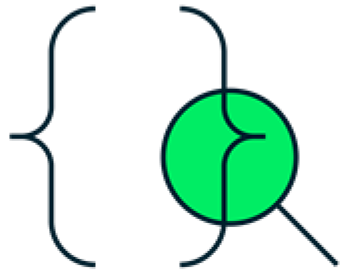
Document Database: Example

```
{
  "_id":
ObjectId("5ef2d4b45b7f11b6d7a
"),
  "user_id": "Sherlock
Holmes",
  "age": 40,
  "address":
     {
       “Country: “England”
       “City”: “London”,
       “Street”: “221B Baker
     St.”
     },
  “Hobbies”:[ violin, crime-
solving ]
}
```

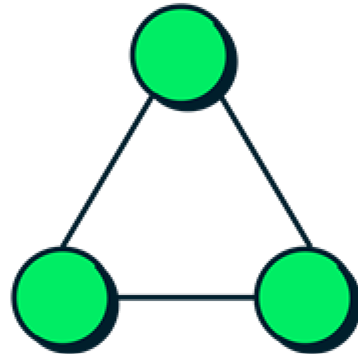
```
{
  "_id":
ObjectId("6ef8d4b32c9f12b6d4a")
,
  "user_id": "John Watson",
  "age": 45,
  "address":
     {
       “Country: “England”
       “City”: “London”,
       “Street”: “221B Baker
     St.”
     },
  “Medical license”: “Active”
}
```

The Document Model and MongoDB

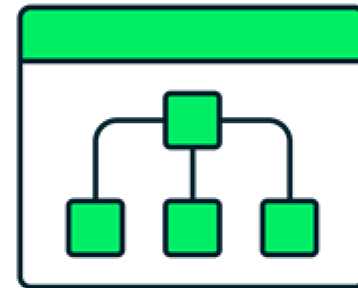
Document Model Key Features



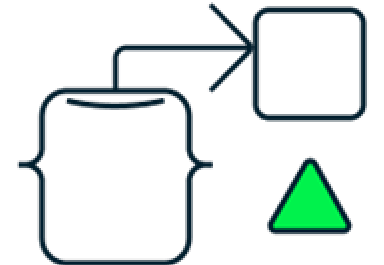
API query or query language



Distributed and resilient



Flexible schema



Object mapping

The Document Model: Structure and Syntax

```
{
   "_id": ObjectId(
         "5f4f7fef2d4b45b7f11b6d7a"),
   "user_id": "Sean",
   "age": 29,
   "Status": "A”,
   "accounts”: [101, 934]
}
```

User details example

The Document Model: JSON format

```
{
  "_id": ObjectId(
         "5f4f7fef2d4b45b7f11b6d7a"),
  "user_id": "Sean",
  "age": 29,
  "Status": "A”,
  "accounts”: [101, 934]
}
```

- curly brackets to mark the start and the end of the document ({})
- field-value pairs are separated by colons (:)
- each field must be enclosed within quotation marks ("")
- each field-value pair is separated by commas (,)
- lists are enclosed using square brackets ([])

Collections in the Document Model

Collections in the Document Model

Document

Collection

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

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

Collection: Example

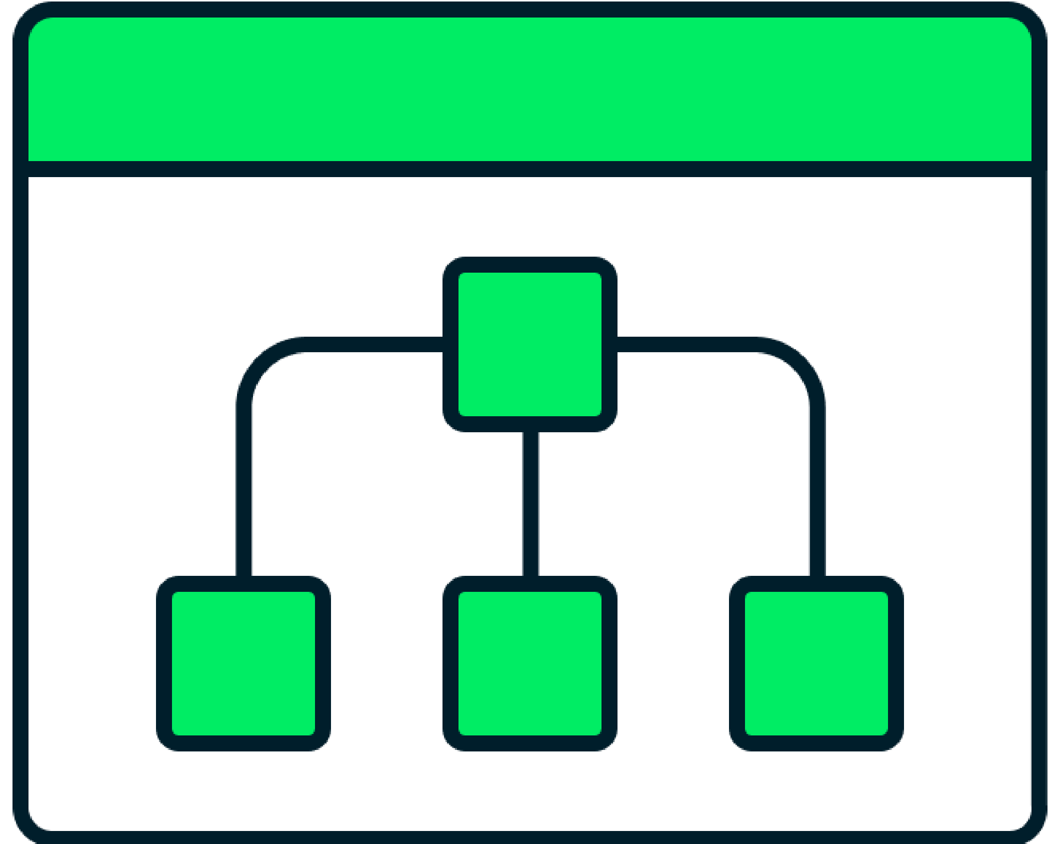
```
{
  "_id": ObjectId(
 "5f4f7fef2d4b45b7f11b6d7a"),
  "user_id": "Sean",
  "age": 29,
  "Status": "A"
}
```

```
{
  "_id": ObjectId(
"5f4f7fef2d4b45b7f11b6d7a"),
  "user_id": "Daniel",
  "age": 25,
  "Status": "A",
  "Country": "USA"
}
```

MongoDB does not enforce a single schema on a collection. Documents can have common fields, but they are not required to by default.

Collections and Schema Validation

- The document model used by MongoDB can enforce a schema if required, the recommended approach is to do so using JSON Schema.
- JSON Schema
  - Allows a prescribed document structure to be configured on a per collection basis.
  - Can tune schema validation according to use case.
  - Can be used by any query to inspect document structure and content.



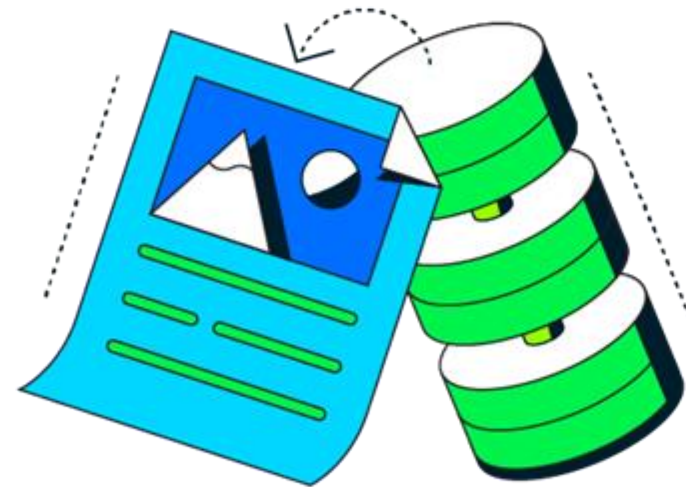
Data Modeling and MongoDB

Schema Design

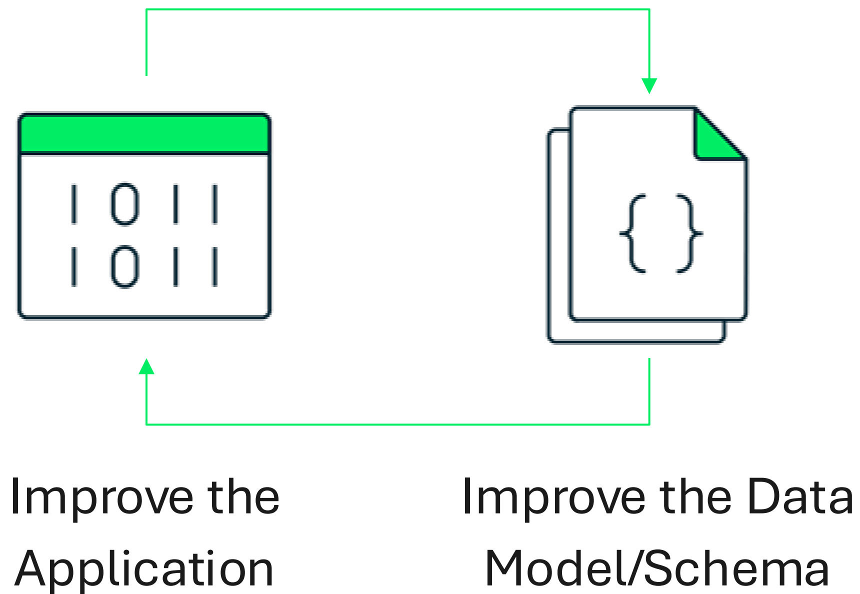
The design comes from the needs of the application first. Therefore, the schema should evolve as the application changes.

Data Modeling and the Document Model

The core of data modeling in the document model is to understand what data is needed by your queries. Once that information is known, can you begin designing the schema.



Data Modeling with MongoDB



- Several design possibilities
- Design for the usage pattern
- Evolving the schema is easy
- No migrations or downtime required for a new version of the schema

Schema Design: Considerations

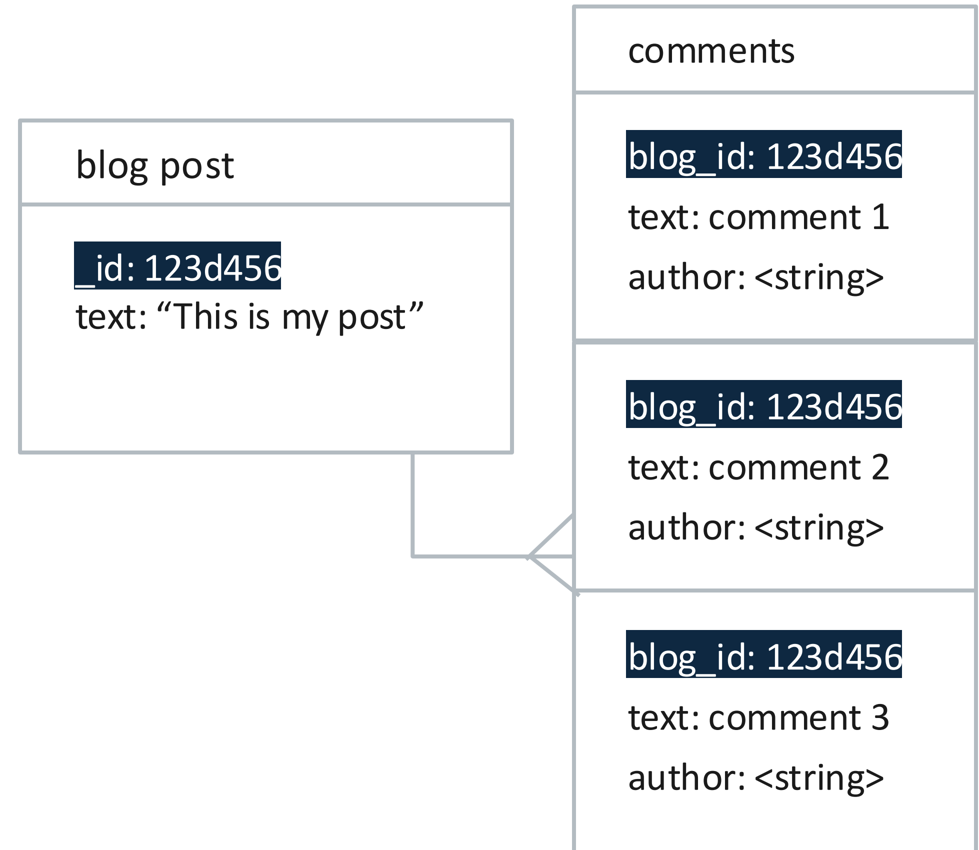
- Your queries and the specific data your application requires.
- How your application reads the data (read patterns).
- How your application writes the data (write patterns).
- What are the relationships between your data (linked or embedded).

Schema Design - Link or Embed?

Embedded vs Linked relationship in the Post-Comment example

Embedded

Linked

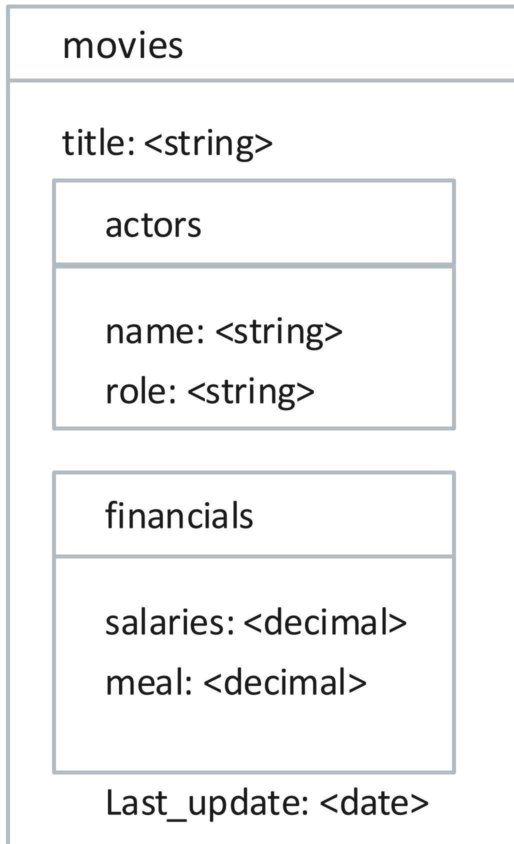


Schema Design - Link or Embed?

- Do I want most of the data’s information embedded?
- Do I need to search within the embedded data?
- How frequently will the embedded data change?
- Is the embedded data shared or private?

Example: Movies and Reviews

Embedded



Linked

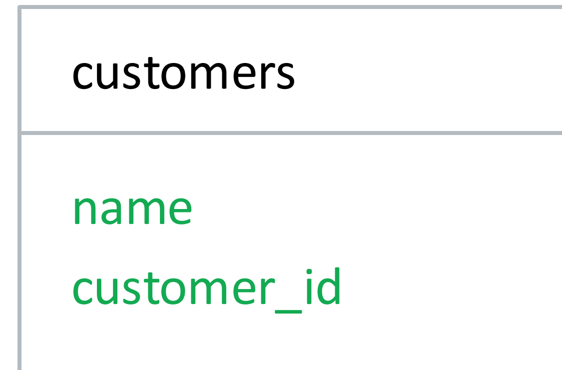
Relationships

Relationships and Data Modeling

- One to One (1-1)
- One to Many (1-N)
- Many to Many (N-N)

Relationships One-to-One (1-1)

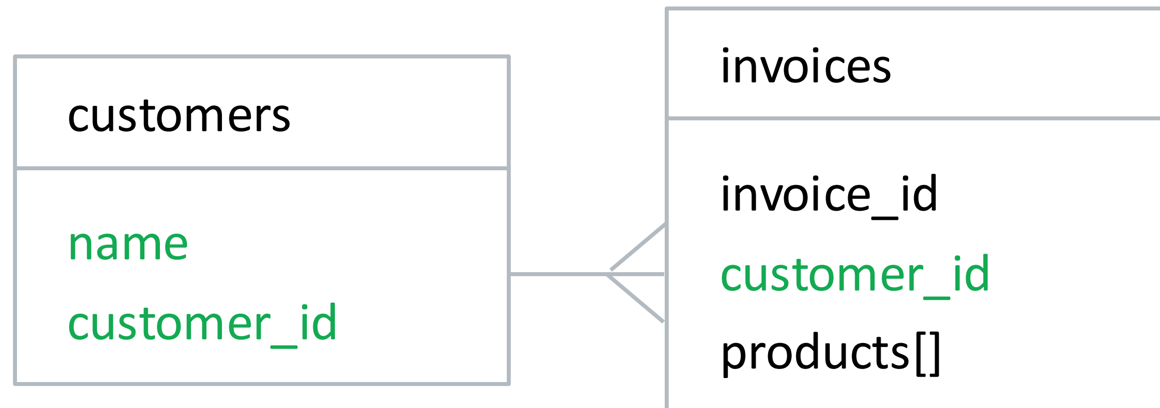
- A one-to-one relationship is represented and stored in a single document, this would typically be data like a person's name and the customer id.



- Example scenario: map patron and address relationships - you’ll need to view one data entity in context of the other.

Relationships One-to-Many (1-N)

- A one-to-many relationship can be considered when an object of a given type is associated with N objects of a second type.

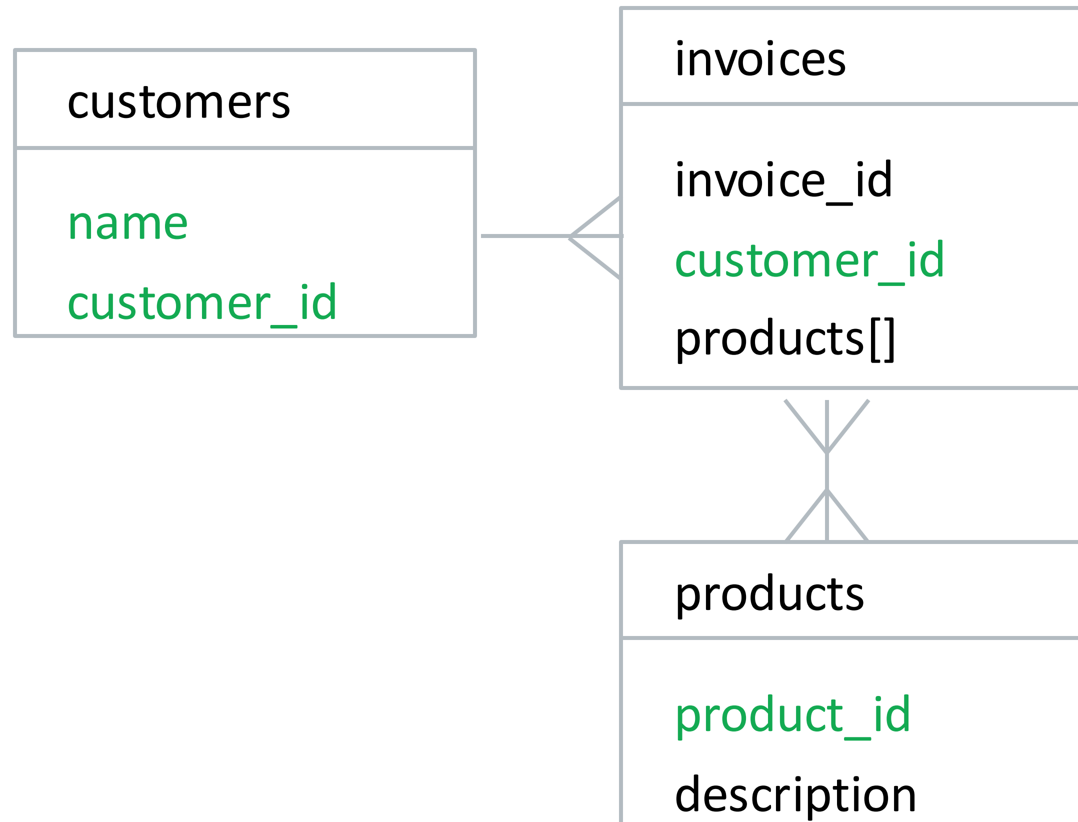


Relationships One-to-Many (1-N)

- Example scenario (link): map publisher and book relationships.
  - Suppose you had the same publisher data for the same book. Embedding the [publisher] document inside the [book] document would lead to repetition of publisher information.
- Example scenario (embed): map a patron with multiple address relationships.
  - In this one-to-many relationship between [patron] and [address] data, the [patron] has multiple [address] entities.

Relationships Many-to-Many (N-N)

- A Many-to-Many relationship between two entities where they both might have many relationships between each other.



Relationships Many-to-Many (N-N)

- Example scenario:
  - a book was written by multiple authors and similarly, one of the authors has written multiple books.
  - How would we go about mapping these relationships?

Embed or Link?

Embed

- For integrity with read operations
- For integrity with write operations
- On one-to-one and one-to-many
- For data that is deleted together by default

Link

- When the "many" side is a huge number
- For integrity on write operations on many-to-many
- When a piece is frequently used, but not the other and memory is an issue

MongoDB ACID support

- **Atomicity**
  - All the operations within a transaction are executed as a single unit.
  - If any operation fails, the entire transaction is rolled back.
- **Consistency**
  - MongoDB ensures that the database remains in a consistent state before and after a transaction, maintaining data integrity.
- **Isolation**
  - Transactions provide isolation, meaning that the operations in a transaction are invisible to other operations until the transaction completes.
- **Durability**
  - Once a transaction is committed, the changes are guaranteed to be saved, even in the event of a server crash or power failure.