Cheatsheets / **Learn MongoDB**

codecademy

Introduction to MongoDB

Embedded Data Models - Denormalized

Embedded data models are an example of a denormalized data model. That is, they create relationships by nesting documents within other documents, rather than using references to model relationships.
In the following example, a relationship between two documents is established by embedding the `engine` document within its parent document.

```
{
  car_id: 48273
  model_name: "Corvette",
  engine: {
    engine_power: 490,
    engine_type: "V8",
    acceleration: "High"
  }
}
```

MongoDB: Embedded Documents

An embedded document in MongoDB is a document that is nested within a document. Embedded documents are stored as the value of one of the document's fields.
The following is an example of an embedded document named `engine` that is nested within a document from a collection called `cars` :

```
// Document from `cars` collec
{
  car_id: 48273
  model_name: "Corvette",
  engine: {
    engine_power: 490,
    engine_type: "V8",
    acceleration: "High"
  }
}
```

Data Modeling

Data modeling is the process of planning an organizational structure for the data in a database. Data modeling is especially important for non-relational databases, like MongoDB, that have a flexible nature and do not rely on a schema-based structure like the table organization used in relational databases.

Modeling Relationship: One-to-One

In MongoDB, a “one-to-one” relationship is modeled through an embedded document. One document contains another document embedded within it.
The following example demonstrates a “one-to-one” relationship between a person and their passport information:

```
{
  _id: ObjectId(...),
  first_name: "Carlton",
  last_name: "Banks",
  passport_info: {
    country: "United States of
    passport_id: 0123456,
    year_issued: 1997
  }
}
```

Modeling Relationship: One-to-Many

In MongoDB, a “one-to-many” relationship is modeled through embedded documents. One document contains multiple documents embedded within it, stored as an array.
The following example demonstrates a “one-to-many” relationship between a customer and their shopping cart:

```
{
  _id: ObjectId(...),
  last_name: "Belcher",
  first_name: "Bob",
  shopping_cart: [
    {
      item_id: 0145,
      name: "burger patties",
      quantity: 200
    },
    {
      item_id: 0147,
      name: "burger buns",
      quantity: 200
    },
    ...
  ]
}
```

MongoDB References

In MongoDB, references are used to associate data and establish relationships between distinct documents. Using references, data can be split into multiple documents while maintaining clearly defined relationships between those documents. The following example demonstrates a reference. The `engine` data is maintained in its own document but is linked (via the `engine_id`) to the `car` document:

```
//Car Document
{
  car_id: 48273
  model_name: "Corvette",
  engine_id: 2165
}

// Engine Document
{
  id: 2165
  engine_power: 490,
  engine_type: "V8",
  acceleration: "High"
}
```

codecademy

Normalized Data - MongoDB References

Reference-based data models are normalized; they use links inside of the data (typically via the `_id` field) to create relationships.
The following example demonstrates a reference. The `engine` data is maintained in its own document but is linked (via the `engine_id`) to the `car` document:

```
//Car Document
{
  car_id: 48273
  model_name: "Corvette",
  engine_id: 2165
}

// Engine Document
{
  _id: 2165
  engine_power: 490,
  engine_type: "V8",
  acceleration: "High"
}
```

codecademy

Modeling Relationship: Many-to-Many

A modeling relationship that associates many documents from one collection to many documents from another collection is a “many-to-many” relationship.
The following example demonstrates a “many-to-many” relationship between students and their classes.

```
// Students Collection



{
  _id: 1,
  name: "Alex",
  average_grade: 3.9,
  course_ids: [ 1, 2, 4 ]
},
{
  _id: 2,
  name: "Bob",
  average_grade: 2.4,
  course_ids: [ 3, 4 ]
}




// Classes Collection
{
  _id: 1,
  name: "Intro to MongoDB",
  student_ids: [ 1 ]
},
{
  _id: 2,
  name: "Programming 101",
  student_ids: [ 1 ]
 },
{
  _id: 3,
  name: "Networking Concepts",
  student_ids: [ 2 ]
 },
{
  _id: 4,
  name: "Understanding Distrib
  student_ids: [ 1, 2 ]
}
```

MongoDB - non-relational document-oriented database system

MongoDB is a non-relational document-oriented database system. Data is structured as key-value pairs, that are organized into individual records called documents.

MongoDB - Document Model

MongoDB adheres to the document model. Data is formatted as JSON, BSON, or YAML and is stored inside documents. Documents containing related content are grouped into collections. The document model used by MongoDB is in contrast to the relational database model where data is stored in tables via rows and columns.

MongoDB - Document Modification Effects

Within a document-oriented database, like MongoDB, modifications made to a single document will only impact that document. This flexibility is an advantage over relational databases where changing a column of a table will impact every record in the table.

codecademy

MongoDB Atlas

MongoDB Atlas is MongoDB's database-as-a-service or DBaaS platform. Using Atlas, MongoDB databases can be created, managed, and deployed from the cloud. Data analytics and data visualization are also available on this platform.

MongoDB Realm

Realm is an application development platform that allows developers to build various applications that are fully integrated with MongoDB. We can use Realm to build mobile, web, desktop, and internet-of-things applications and synchronize data between each of the devices the application is installed on. We can also use Realm to facilitate administrative tasks like authenticating and managing users.

MongoDB Document

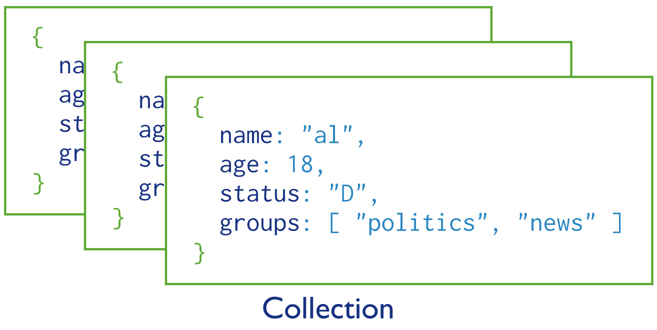
A MongoDB document is an individual record of data stored as "field-value" pairs. A field uniquely identifies a data point while a value is the data point itself.
Check out the following example of a MongoDB document:

```
{
 _id: ObjectId("98232303df349
 name: { first: "Ezio", last:
 age: 33,
 major: [ "Italian", "Physics'
}
```

```
{
  <field1>: <value1>,
  <field2>: <value2>,
  ...
}
```

MongoDB Collection

A MongoDB collection is a group of documents containing similar information. Documents within a collection can have different fields, though they tend to share a similar structure.



MongoDB Data Hierarchy

A MongoDB database is a number of collections grouped together for a specific use case. A database is made of many collections and collections are made of many documents.

BSON in MongoDB

Binary Javascript Object Notation (BSON) is a non-human-readable data format that MongoDB uses to store data more efficiently than JSON. BSON takes up less space, is faster to parse, and can store more datatypes than JSON.

JSON/BSON Relationship in MongoDB

MongoDB allows developers to insert or retrieve data in readable JSON format. Internally, MongoDB stores data as BSON, a format that is not readable by humans, but is a more efficient way for MongoDB to store and parse data.

JSON Advantages Over BSON

JSON's advantage over BSON is it is highly structured and parsable by humans.
The following is an example of "field-value" pairs formatted as JSON:

```
{
 first_name: "Lucius",
 last_name: "Malfoy",
 year: 7
}
```

BSON Advantages Over JSON

BSON's advantage over JSON is storage efficiency and supporting data formats that JSON does not - like dates. BSON is also faster to parse than JSON.

codecademy

JSON Definition in MongoDB

JavaScript Object Notation (JSON) is a human-readable, text-based data format that MongoDB uses for insertion and retrieval of data.
Check out this example of a MongoDB document structured as JSON. Data is inserted and retrieved in MongoDB documents as JSON.

```
{
 _id:
ObjectId("9021032303df34948d67wd391"),
 name: { first: "Jin", last: "Sakai" },
 age: 21,
 major: [ "Japanese", "Chemistry"]
}
```

Print Share