SuperDataScience

# ULTIMATE MONGODB CHEATSHEET

# INTUITION

## KEY DIFFERENCES BETWEEN SQL AND NOSQL

| SQL | NOSQL |
|---|---|
| Table Based | **Documents**, Key-Value pairs, Graph-based, or Wide Column Stores |
| Defined Schema | Undefined / Flexible Schema |
| Better for Complex Queries | Better for Complex Data Structures |
| Better for Transactional Systems | Better for Horizontal Scaling |
| Examples: MySQL, Postgres, Oracle, SQLite | Examples: MongoDB, Cassandra, HBase, Redis, Neo4j |

## CRUD OPERATIONS

- **Create:** Insert one or more records into the database
- **Read:** Pull one or more records from the database
- **Update:** Modify one or more existing records in the database
- **Delete:** Remove one or more records from the database

## PRIMARY KEYS

- Always unique for each document in a collection
- Cannot have null or missing values
- In MongoDB, this is the **"_id"** key and is added to every document by default:

```
"_id" : ObjectId("5c92eca2cc7efa53af5dae22")
```

# RELATIONSHIPS IN DATA

In MongoDB, you can establish relationships in two primary ways:

1. Using a key in one document or collection that corresponds to a key in another (user_id in the images).

**User:**

```
{
    "_id" : ObjectId("5c943413cc7efa53af5dae24"),
    "user_id" : 3173,
    "name" : "helen",
    "age" : 32
}
```

**Blog Post:**

```
{
    "_id" : ObjectId("5c943545cc7efa53af5dae25"),
    "post_id" : 5186,
    "user_id" : 3173,
    "body" : "Lorem ipsum dolor sit amet",
    "topic" : "health and wellness",
    "likes" : 57,
    "dislikes" : 31
}
```

2. Embedded documents: Storing an entire document inside of another

**User document with blog posts embedded:**

```
{
    "_id" : ObjectId("5c943654cc7efa53af5dae26"),
    "user_id" : 3173,
    "name" : "helen",
    "age" : 32,
    "posts" : [
          {
                "body" : "Lorem ipsum dolor sit amet",
                "topic" : "health and wellness",
                "likes" : 57,
                "dislikes" : 31
          }
    ]
}
```

## GUIDING PRINCIPLES FOR STRUCTURING YOUR DATABASE

- Remember: MongoDB has a flexible schema, so you can change the the structure of your data as needed after creating your database.
- One structure might be better for a certain situation than another.
- In general, for each situation in which you're reading data from your database, your goals should be to:
  - Minimize the amount of data that is loaded unnecessarily
  - Access all of the data you need in one single query.
- These goals often conflict with each other! It's difficult to perfectly achieve both of them, but it's helpful to try to get as close as we can.

# 2. PRACTICAL

## STARTING UP THE MONGO SHELL

- In one terminal window, type **mongod** and hit enter. This will start your MongoDB server.
  - *Note: You will also need to have a MongoDB server running to use MongoDB Compass.
- Open up another terminal, type **mongo** and hit enter
  - You should now have a running MongoDB shell!

## GETTING STARTED WITH THE SHELL

- To see what databases already exist, type **show dbs** in the command prompt and hit enter.
- You can create a new database or switch to an existing database with the same command! Type **use database-name** and hit enter
  - Replace **"database-name"** with the actual name of the database.
- Once you're using a database, type **show collections** and hit enter to see what collections you have.
- To start working with data in a collection you can use the CRUD operations shown below!

## CRUD OPERATIONS IN THE MONGO SHELL

**Create:** Inserts a document with a name of patrick

```
db.collection.insert({“name”: “patrick”})
```

**Read:** Finds all documents with an age of 42

```
db.collection.find({“age”: 42})
```

**Update:** Replaces the first document found with a country of “US” with a new document, containing only a country of “USA”

```
db.collection.update({“country”: “US”}, {“country”: “USA”})
```

**Delete:** Removes all documents with a user_id of 4106

```
db.collection.remove({“user_id”: 4106})
```

## DIFFERENT QUERIES FOR DATA ANALYSIS

Returns all posts where the topic is not equal to sports

```
{“topic”: {$ne: “sports”}}
```

Returns all posts with more than 36 likes

```
{“likes”: {$gt: 36}}
```

Returns all posts with less than 52 dislikes

```
{“dislikes”: {$lt: 36}}
```

Returns all posts where the topic is null

```
{“topic”: null}
```

Returns all posts where the topic is either fitness or cooking

```
{“topic”: {$in: [“fitness”, “cooking”]}}
```

You can also combine several queries with **$and** and **$or**!

Returns all posts with less than 43 likes and more than 25 dislikes

```
{$and: [{"likes": {$lt: 43}}, {"dislikes": {$gt: 25}}]}
```

Returns all posts with less than than 34 dislikes or where the topic is not equal to cooking

```
{$or: [{"dislikes": {$lt: 34}}, {"topic": {$ne: "cooking"}}]}
```