NoSQL Data Manipulation

Review

Basic MongoDB Commands

- **CRUD Commands Overview**: MongoDB's core commands allow for creating, reading, updating and deleting documents within collections.
- **Find Command (find)**:
  - **Purpose**: retrive documents from a collection based on specified criteria.
  - **Primary use**: retrieving specific fields and filtering documents by conditions.

```
db.collection.find(query, projection)
```

- **Insert Command (insertOne, insertMany)**:
  - **Purpose**: add new documents to a collection.
  - **Primary use**: create new entries in a collection, one document at a time or multiple documents at once.

```
// Insert one
db.collection.insertOne(document)

// Insert multiple
db.collection.insertMany([ document1, document2, ...])
```

- **Update Command (updateOne, updateMany)**:
  - **Purpose**: modify existing documents in a collection based on a query.
  - **Primary use**: change document fields, add new fields or update multiple documents simultaneously.

```
// Update one
db.collection.updateOne(filter, update)
```

```
// Insert multiple
db.collection.updateMany(filter, update)
```

- **Delete Command (deleteOne, deleteMany)**:
  - **Purpose**: remove documents from a collection based on a specified condition.
  - **Primary use**: delete document that match the filter, a single document or multiple documents.

```
// Delete one
db.collection.deleteOne(filter)

// Delete multiple
db.collection.deleteMany(filter)
```

MongoDB Command Structure

MongoDB commands generally follow the format:

```
db.collection.method(query, projection)
```

This structure includes the **database (db), collection, method** and optional **parameters**.

- **Database (db)**: refers to the currently selected database.
- **Collection**: specifies the collection within the database where the command will operate.
- **Method**: defines the operation to perform.
- **Query Filter**: the first parameter, typically {}, filters documents based on conditions.
- **Projection (for find only)**: the second parameter specifies fields to include or exclude in results.

Brackets in MongoDB

- **Curly Braces { } for Objects**: denote objects and are used for **query filters, projections** and **update operations**.
- **Square Brackets [ ] for Arrays**: define arrays within documents, storing multiple values in a single field. Used for array-specific operators in queries and updates.
- **Nested Brackets in Queries and Documents**: MongoDB allows nesting of { } and [ ] to access and manipulate complex structures like arrays within objects or objects within arrays.

The Role of _id in MongoDB Documents

_id is a unique identifier in MongoDB collections. MongoDB automatically assigns an _id field to each document in a collection, making it a unique identifier or primary key for that document.

- The _id field ensures each document is uniquely identifiable, preventing duplicate entries within a collection.
- Custom _id values can be assigned during insertion, though each _id must still be unique within the collection.

_id in group Stages as a Grouping Key:

- MongoDB's aggregation framework, the _id field in the $group stage acts as the **grouping key** rather than the document's unique identifier.
- This use of _id groups documents based on a specified field, allowing aggregation operations (counting, averaging) for each group.

```
// Set _id to age
{ $group: { _id: "$age", count: { $sum: 1 } } }
```

Example of _id Dual Purpose

- **Document Identifier:** in a students collection, _id could uniquely identify each student, with each document's _id serving as a reference key.
- **Grouping Key in Aggregation**: using _id as a grouping key, groups documents by the selected attribute instead of using the _id of each document.

- **Custom _id for Specific Uses**: custom _id values can serve as unique identifiers for specific applications, but are still treated as unique within the collection.

Group by Age and Calc Avg Attendance

```
db.students.aggregate([
{ $group: { _id: "$age", averageAttendance: { $avg: "$attenda
])
```

- **Key Concepts Illustrated by _id in this Query**:
  - **Grouping by Field (age)**: by setting _id to $age, MongoDB groups documents based on age, aggregating all documents with the same age value into a single group.
  - **Calculating Aggregates within Groups**: the averageAttendance expression calculates the average attendance for each age group, allowing you to analyze attendance patterns by age.
- **Dual Role of _id in MongoDB**:
  - **As a Document Identifier**: In regular collection documents, _id serves as a unique identifier.
  - **As a Grouping Key in Aggregation**: In this aggregation, _id becomes a label for grouping
    purposes, allowing you to group documents by age. The result will display a list of age groups, each represented by a unique _id equal to the age value, not the original document _id.

Tips for Formulating MongoDB Commands

Strategies for Defining Clear Filters, Projections and Updates

- **Filters**: start with simple specific conditions to narrow down results. Use operators like $gt, $lt, $in and $regex for targeted queries.
- **Projections**: only include necessary fields by specifying them in the projection, which can reduce output clutter and improve readability.
- **Updates**: use $set to modify specific fields without overwriting the entire document. Other operators like $inc (increment), $push (add to array), and

$unset (remove field) help you make precise updates.

Using MongoDB Documentation and mongosh Autocomplete

- **Command Help**: enter a command followed by .help() to see available options and usage hints. For example, db.collection.find.help() displays find method options.

Advanced Data Manipulation Techniques

Complex Structures

Embedded Document

An **embedded document** is a document stored within another document, creating a hierarchical structure. It is the act of including a related object as part of a larger object.

It is MongoDB's way of handling **one-to-one** or **one-to-few** relationships without creating a separate collection.

The embedded document itself is stored as a **single field** in the parent document, but it contains its own set of key-value pairs.

Use embedding when the related data is small, closely tied to the parent document, and often retrieved together.

```
{
    "name": "Alice",
    "address": {
        "street": "123 Main St",
        "city": "Dublin",
        "zipcode": "10001"
    }
}
```

Array

An **array** is a data structure used to store multiple values or objects in a single field. These values can be scalars or **embedded documents.**

Arrays can contain multiple **embedded documents**, effectively modelling **one-to-many relationships**