\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Step 1: Connect to the MongoDB database

use EmployeeManagement

// Step 2: Creating a collection named 'employees'

db.createCollection("employees")

// Sample Employee Documents

const employees = [

{ employee\_id: 1, name: "Alice", department: "HR", salary: 50000, age: 30, joining\_date: new Date("2019-08-01") },

{ employee\_id: 2, name: "Bob", department: "Engineering", salary: 70000, age: 35, joining\_date: new Date("2020-01-15") },

{ employee\_id: 3, name: "Charlie", department: "Marketing", salary: 45000, age: 28, joining\_date: new Date("2021-03-22") },

{ employee\_id: 4, name: "David", department: "Finance", salary: 65000, age: 40, joining\_date: new Date("2018-05-10") }

]

// Step 3: Inserting Documents

// Inserting a batch of employees

db.employees.insertMany(employees)

// Single document insertion with validation

db.employees.insertOne({

employee\_id: 5,

name: "Emma",

department: "Engineering",

salary: 60000,

age: 29,

joining\_date: new Date("2023-06-20")

})

// Step 4: Removing Documents

// Removing a specific document by employee ID

db.employees.deleteOne({ employee\_id: 3 })

// Removing multiple documents (e.g., removing all employees in the Marketing department)

db.employees.deleteMany({ department: "Marketing" })

// Step 5: Updating Documents

// Document replacement (replacing an entire document)

db.employees.replaceOne(

{ employee\_id: 2 },

{ employee\_id: 2, name: "Bob", department: "Engineering", salary: 75000, age: 36, joining\_date: new Date("2020-01-15") }

)

// Using modifiers to update specific fields

// Increment salary by 5000 for employees in the Engineering department

db.employees.updateMany(

{ department: "Engineering" },

{ $inc: { salary: 5000 } }

)

// Upsert (insert a document if it does not exist, otherwise update)

db.employees.updateOne(

{ employee\_id: 6 },

{ $set: { name: "Frank", department: "IT", salary: 55000, age: 32, joining\_date: new Date("2023-08-01") } },

{ upsert: true }

)

// Updating multiple documents (e.g., set a new field 'is\_active' to true for all employees)

db.employees.updateMany(

{},

{ $set: { is\_active: true } }

)

// Returning the updated document

// Find and update an employee by ID, return the updated document

db.employees.findOneAndUpdate(

{ employee\_id: 1 },

{ $set: { salary: 52000 } },

{ returnNewDocument: true }

)

// Step 6: Cursors and Querying Options

// Limit: Retrieve only the first 3 employees

db.employees.find().limit(3)

// Skip: Retrieve employees after skipping the first two

db.employees.find().skip(2)

// Sort: Retrieve employees sorted by salary in descending order

db.employees.find().sort({ salary: -1 })

// Advanced query: Retrieve employees who joined after 2020 and have a salary above 55000

db.employees.find({

$and: [

{ joining\_date: { $gt: new Date("2020-01-01") } },

{ salary: { $gt: 55000 } }

]

}).limit(5).sort({ salary: -1 })

// Step 7: Additional Queries and Commands

// Count number of employees in the Engineering department

db.employees.countDocuments({ department: "Engineering" })

// Aggregation example: Calculate the average salary by department

db.employees.aggregate([

{ $group: { \_id: "$department", average\_salary: { $avg: "$salary" } } }

])

// Close

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Step 1: Connect to MongoDB and create/use the database

use EmployeeManagement

// Step 2: Create the collection and insert sample data into 'employees'

db.createCollection("employees")

db.employees.insertMany([

{ employee\_id: 1, name: "Alice", department: "HR", salary: 50000, age: 30, skills: ["communication", "organization"], joining\_date: new Date("2019-08-01") },

{ employee\_id: 2, name: "Bob", department: "Engineering", salary: 70000, age: 35, skills: ["programming", "data analysis"], joining\_date: new Date("2020-01-15") },

{ employee\_id: 3, name: "Charlie", department: "Marketing", salary: 45000, age: 28, skills: ["marketing", "communication"], joining\_date: new Date("2021-03-22") },

{ employee\_id: 4, name: "David", department: "Finance", salary: 65000, age: 40, skills: ["accounting", "data analysis"], joining\_date: new Date("2018-05-10") },

{ employee\_id: 5, name: "Emma", department: "Engineering", salary: 60000, age: 29, skills: ["programming", "problem-solving"], joining\_date: new Date("2023-06-20") },

{ employee\_id: 6, name: "Frank", department: "Engineering", salary: null, age: 32, skills: ["programming", "management"], joining\_date: new Date("2023-08-01") }

])

// Query Examples

// 1. Find all documents in the 'employees' collection

db.employees.find()

// 2. findOne() - Retrieve a single employee document by name

db.employees.findOne({ name: "Alice" })

// 3. Query conditionals - Find employees with salary greater than 50000

db.employees.find({ salary: { $gt: 50000 } })

// 4. OR query - Find employees who work in HR or Finance

db.employees.find({ $or: [{ department: "HR" }, { department: "Finance" }] })

// 5. $not - Find employees who do not work in the Engineering department

db.employees.find({ department: { $not: { $eq: "Engineering" } } })

// 6. Conditional semantics - Find employees with age between 30 and 40 (inclusive)

db.employees.find({ age: { $gte: 30, $lte: 40 } })

// 7. Type-specific queries - Find employees with salary field equal to null

db.employees.find({ salary: { $type: "null" } })

// 8. Regular expression query - Find employees with names starting with 'A'

db.employees.find({ name: { $regex: /^A/ } })

// 9. Querying arrays - Find employees with 'programming' as one of their skills

db.employees.find({ skills: "programming" })

// 10. $where query - Find employees whose salary is greater than twice their age

db.employees.find({ $where: "this.salary > (this.age \* 2)" })

// Additional Queries

// 11. Query for employees who joined after January 1, 2020, sorted by joining date in ascending order

db.employees.find({ joining\_date: { $gt: new Date("2020-01-01") } }).sort({ joining\_date: 1 })

// 12. Find all employees who have both 'programming' and 'data analysis' in their skills

db.employees.find({ skills: { $all: ["programming", "data analysis"] } })

// 13. Find employees with a specific salary or no salary value (null)

db.employees.find({ $or: [{ salary: { $eq: 70000 } }, { salary: { $type: "null" } }] })

// 14. $in query - Find employees in HR or Marketing departments

db.employees.find({ department: { $in: ["HR", "Marketing"] } })

// 15. $nin query - Find employees who do not belong to Engineering or Finance departments

db.employees.find({ department: { $nin: ["Engineering", "Finance"] } })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to the database

use OrderManagement

// Create the 'Orders' collection

db.createCollection("Orders")

// Insert sample data into 'Orders'

db.Orders.insertMany([

{ orderid: 1, custid: "C001", date: new Date("2024-01-15"), amount: 150 },

{ orderid: 2, custid: "C002", date: new Date("2024-01-17"), amount: 200 },

{ orderid: 3, custid: "C001", date: new Date("2024-01-20"), amount: 100 },

{ orderid: 4, custid: "C003", date: new Date("2024-01-22"), amount: 250 },

{ orderid: 5, custid: "C002", date: new Date("2024-01-25"), amount: 300 },

{ orderid: 6, custid: "C001", date: new Date("2024-01-27"), amount: 200 },

{ orderid: 7, custid: "C003", date: new Date("2024-01-30"), amount: 400 }

])

// Define the map function to emit each customer's ID and amount

const mapFunction = function() {

emit(this.custid, this.amount);

};

// Define the reduce function to sum the total amount for each customer

const reduceFunction = function(key, values) {

return Array.sum(values);

};

// Execute the Map-Reduce operation

db.Orders.mapReduce(

mapFunction,

reduceFunction,

{

out: "total\_amount\_per\_customer" // Output the result to a new collection

}

)

// Create an index on the 'custid' field

db.Orders.createIndex({ custid: 1 })

// Create an index on the 'date' field to speed up date-based queries

db.Orders.createIndex({ date: 1 })

// Create a compound index on 'custid' and 'amount' for queries that filter by both

db.Orders.createIndex({ custid: 1, amount: 1 })

// Confirm the indexes

db.Orders.getIndexes()

// View the results

db.total\_amount\_per\_customer.find()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Step 1: Switch to the OrderManagement database

use OrderManagement

// Step 2: Create the 'Orders' collection and insert sample data

db.createCollection("Orders")

db.Orders.insertMany([

{ orderid: 1, custid: "C001", date: new Date("2024-01-15"), amount: 150 },

{ orderid: 2, custid: "C002", date: new Date("2024-01-17"), amount: 200 },

{ orderid: 3, custid: "C001", date: new Date("2024-01-20"), amount: 100 },

{ orderid: 4, custid: "C003", date: new Date("2024-01-22"), amount: 250 },

{ orderid: 5, custid: "C002", date: new Date("2024-01-25"), amount: 300 },

{ orderid: 6, custid: "C001", date: new Date("2024-01-27"), amount: 200 },

{ orderid: 7, custid: "C003", date: new Date("2024-01-30"), amount: 400 }

])

// Step 3: Implement aggregation to calculate total amount, min, max, avg, count, first, last, push, and addToSet for each customer

db.Orders.aggregate([

{

$group: {

\_id: "$custid", // Group by custid

totalAmount: { $sum: "$amount" }, // Sum of amount

minAmount: { $min: "$amount" }, // Minimum amount

maxAmount: { $max: "$amount" }, // Maximum amount

avgAmount: { $avg: "$amount" }, // Average amount

orderCount: { $count: {} }, // Count of orders

firstOrder: { $first: "$date" }, // Date of the first order

lastOrder: { $last: "$date" }, // Date of the last order

allAmounts: { $push: "$amount" }, // Push all amounts into an array

uniqueAmounts: { $addToSet: "$amount" } // Unique amounts only

}

}

])

// Step 4: Additional aggregation with match and sort - total amount for each customer with orders after a specific date, sorted by totalAmount

db.Orders.aggregate([

{ $match: { date: { $gt: new Date("2024-01-01") } } },

{

$group: {

\_id: "$custid",

totalAmount: { $sum: "$amount" }

}

},

{ $sort: { totalAmount: -1 } } // Sort by totalAmount in descending order

])

// Step 5: Aggregation with project to show only custid and amount fields

db.Orders.aggregate([

{

$project: {

\_id: 0, // Exclude the \_id field

custid: 1,

amount: 1

}

}

])

// Step 6: Create indexes on 'custid', 'date', and a compound index on 'custid' and 'amount'

db.Orders.createIndex({ custid: 1 })

db.Orders.createIndex({ date: 1 })

db.Orders.createIndex({ custid: 1, amount: 1 })

// Confirm the created indexes

db.Orders.getIndexes()

\*\*\*\*\*\*\*\*\*\*\*\*5\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STAR SCHEMA:

// Database selection

use OnlineFoodOrderingSystem

// Orders Collection (Fact Table)

db.Orders.insertMany([

{

order\_id: 1,

customer\_id: 101,

restaurant\_id: 201,

food\_item\_id: [301, 302],

delivery\_id: 401,

order\_date: new Date("2024-10-28"),

total\_amount: 25.99

},

{

order\_id: 2,

customer\_id: 102,

restaurant\_id: 202,

food\_item\_id: [303],

delivery\_id: 402,

order\_date: new Date("2024-10-29"),

total\_amount: 15.49

}

])

// Customers Collection (Dimension Table)

db.Customers.insertMany([

{ customer\_id: 101, name: "Alice", email: "alice@example.com", phone: "555-1234", address: "123 Main St" },

{ customer\_id: 102, name: "Bob", email: "bob@example.com", phone: "555-5678", address: "456 Elm St" }

])

// Restaurants Collection (Dimension Table)

db.Restaurants.insertMany([

{ restaurant\_id: 201, name: "Pizza Place", location: "Downtown", rating: 4.5, cuisine\_type: "Italian" },

{ restaurant\_id: 202, name: "Burger Joint", location: "Uptown", rating: 4.0, cuisine\_type: "American" }

])

// FoodItems Collection (Dimension Table)

db.FoodItems.insertMany([

{ food\_item\_id: 301, name: "Pepperoni Pizza", category: "Main Course", price: 12.99 },

{ food\_item\_id: 302, name: "Garlic Bread", category: "Appetizer", price: 5.00 },

{ food\_item\_id: 303, name: "Cheeseburger", category: "Main Course", price: 15.49 }

])

// Delivery Collection (Dimension Table)

db.Delivery.insertMany([

{ delivery\_id: 401, delivery\_person: "Dave", delivery\_status: "Delivered", delivery\_time: "30 mins" },

{ delivery\_id: 402, delivery\_person: "Eve", delivery\_status: "In Transit", delivery\_time: "15 mins" }

])

SNOWFLAKE\*\*

// Orders Collection remains the same as in the Star Schema

// Customers Collection (Linked to Customer\_Address)

db.Customers.insertMany([

{ customer\_id: 101, name: "Alice", email: "alice@example.com", phone: "555-1234", address\_id: 1001 },

{ customer\_id: 102, name: "Bob", email: "bob@example.com", phone: "555-5678", address\_id: 1002 }

])

// Customer\_Address Sub-Collection

db.Customer\_Address.insertMany([

{ address\_id: 1001, street: "123 Main St", city: "Metropolis", state: "NY", zip\_code: "10001" },

{ address\_id: 1002, street: "456 Elm St", city: "Gotham", state: "NY", zip\_code: "10002" }

])

// Restaurants Collection (Linked to Restaurant\_Location)

db.Restaurants.insertMany([

{ restaurant\_id: 201, name: "Pizza Place", location\_id: 2001, rating: 4.5, cuisine\_type: "Italian" },

{ restaurant\_id: 202, name: "Burger Joint", location\_id: 2002, rating: 4.0, cuisine\_type: "American" }

])

// Restaurant\_Location Sub-Collection

db.Restaurant\_Location.insertMany([

{ location\_id: 2001, city: "Downtown", state: "NY" },

{ location\_id: 2002, city: "Uptown", state: "NY" }

])

// FoodItems Collection (Linked to Food\_Category)

db.FoodItems.insertMany([

{ food\_item\_id: 301, name: "Pepperoni Pizza", category\_id: 3001, price: 12.99 },

{ food\_item\_id: 302, name: "Garlic Bread", category\_id: 3002, price: 5.00 },

{ food\_item\_id: 303, name: "Cheeseburger", category\_id: 3001, price: 15.49 }

])

// Food\_Category Sub-Collection

db.Food\_Category.insertMany([

{ category\_id: 3001, category\_name: "Main Course" },

{ category\_id: 3002, category\_name: "Appetizer" }

])

// Delivery Collection remains the same as in the Star Schema

For the \*\*Star Schema\*\*, we use the following tables: `Orders` (fact table with order details), and dimension tables `Customers` (customer info), `Restaurants` (restaurant info), `FoodItems` (food details), and `Delivery` (delivery details).

For the \*\*Snowflake Schema\*\*, we have the same core tables, but `Customers` links to `Customer\_Address` (normalized address info), `Restaurants` links to `Restaurant\_Location` (location details), and `FoodItems` links to `Food\_Category` (food category info).

\*\*\*\*\*\*\*\*\*\*\*\*6\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to LibraryManagement database

use LibraryManagement

// Create and insert sample data into the Books collection with batch insert and validation

db.createCollection("Books", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["book\_id", "title", "author", "genre", "published\_year", "copies\_available"],

properties: {

book\_id: { bsonType: "int", description: "must be an integer and is required" },

title: { bsonType: "string", description: "must be a string and is required" },

author: { bsonType: "string", description: "must be a string and is required" },

genre: { bsonType: "string", description: "must be a string and is required" },

published\_year: { bsonType: "int", description: "must be an integer and is required" },

copies\_available: { bsonType: "int", description: "must be an integer and is required" }

}

}

}

})

// Batch insert books

db.Books.insertMany([

{ book\_id: 1, title: "1984", author: "George Orwell", genre: "Dystopian", published\_year: 1949, copies\_available: 5 },

{ book\_id: 2, title: "To Kill a Mockingbird", author: "Harper Lee", genre: "Fiction", published\_year: 1960, copies\_available: 3 },

{ book\_id: 3, title: "The Great Gatsby", author: "F. Scott Fitzgerald", genre: "Classic", published\_year: 1925, copies\_available: 4 },

{ book\_id: 4, title: "Moby Dick", author: "Herman Melville", genre: "Adventure", published\_year: 1851, copies\_available: 2 },

{ book\_id: 5, title: "Pride and Prejudice", author: "Jane Austen", genre: "Romance", published\_year: 1813, copies\_available: 6 }

])

// Inserting a single book with validation

db.Books.insertOne({ book\_id: 6, title: "War and Peace", author: "Leo Tolstoy", genre: "Historical", published\_year: 1869, copies\_available: 3 })

// Remove a document by title

db.Books.deleteOne({ title: "Moby Dick" })

// Update a document (document replacement)

db.Books.replaceOne(

{ book\_id: 1 },

{ book\_id: 1, title: "1984", author: "George Orwell", genre: "Dystopian", published\_year: 1949, copies\_available: 10 }

)

// Update using modifiers ($set and $inc)

db.Books.updateOne(

{ title: "The Great Gatsby" },

{ $set: { genre: "Modern Classic" }, $inc: { copies\_available: 2 } }

)

// Upsert (update if exists, insert if not)

db.Books.updateOne(

{ book\_id: 7 },

{

$setOnInsert: { title: "The Catcher in the Rye", author: "J.D. Salinger", genre: "Fiction", published\_year: 1951, copies\_available: 4 }

},

{ upsert: true }

)

// Update multiple documents (increasing available copies for all Classic books)

db.Books.updateMany(

{ genre: "Classic" },

{ $inc: { copies\_available: 1 } }

)

// Return updated document

const updatedBook = db.Books.findOneAndUpdate(

{ title: "1984" },

{ $inc: { copies\_available: -1 } },

{ returnNewDocument: true }

)

print("Updated Book:", updatedBook)

// Cursors - limit, skip, and sort

db.Books.find().limit(3) // Limit to 3 documents

db.Books.find().skip(2).sort({ published\_year: -1 }) // Skip 2 and sort by published year in descending order

// Advanced query options

db.Books.find(

{ $or: [{ genre: "Fiction" }, { published\_year: { $gte: 1900 } }] }

).sort({ title: 1 }).limit(5)

\*\*\*\*\*\*\*\*\*\*\*\*7\*\*\*\*\*\*\*\*\*\*

// Switch to LibraryManagement database

use LibraryManagement

// Sample data insertion into Books collection

db.Books.insertMany([

{ book\_id: 1, title: "1984", author: "George Orwell", genre: "Dystopian", published\_year: 1949, copies\_available: 5, tags: ["classic", "novel"] },

{ book\_id: 2, title: "To Kill a Mockingbird", author: "Harper Lee", genre: "Fiction", published\_year: 1960, copies\_available: 3, tags: ["classic", "law"] },

{ book\_id: 3, title: "The Great Gatsby", author: "F. Scott Fitzgerald", genre: "Classic", published\_year: 1925, copies\_available: 4, tags: ["classic", "novel"] },

{ book\_id: 4, title: "Moby Dick", author: "Herman Melville", genre: "Adventure", published\_year: 1851, copies\_available: 2, tags: ["sea", "adventure"] },

{ book\_id: 5, title: "Pride and Prejudice", author: "Jane Austen", genre: "Romance", published\_year: 1813, copies\_available: 6, tags: ["classic", "romance"] },

{ book\_id: 6, title: "War and Peace", author: "Leo Tolstoy", genre: "Historical", published\_year: 1869, copies\_available: 3, tags: ["war", "history"] },

{ book\_id: 7, title: "Harry Potter and the Philosopher's Stone", author: "J.K. Rowling", genre: "Fantasy", published\_year: 1997, copies\_available: 8, tags: ["fantasy", "magic"] },

{ book\_id: 8, title: "The Hobbit", author: "J.R.R. Tolkien", genre: "Fantasy", published\_year: 1937, copies\_available: 7, tags: ["fantasy", "adventure"] },

{ book\_id: 9, title: "Alice's Adventures in Wonderland", author: "Lewis Carroll", genre: "Fantasy", published\_year: 1865, copies\_available: 5, tags: ["fantasy", "classic"] },

{ book\_id: 10, title: "The Catcher in the Rye", author: "J.D. Salinger", genre: "Fiction", published\_year: 1951, copies\_available: 4, tags: ["classic", "novel"] }

])

// Queries

// 1. Find a single book by title using `findOne`

db.Books.findOne({ title: "1984" })

// 2. Find all books in a specific genre (e.g., "Fantasy") using `find`

db.Books.find({ genre: "Fantasy" })

// 3. Using query criteria - Find books published before 1900

db.Books.find({ published\_year: { $lt: 1900 } })

// 4. Using OR queries - Find books in either "Fantasy" or "Historical" genre

db.Books.find({ $or: [{ genre: "Fantasy" }, { genre: "Historical" }] })

// 5. Using $not - Find books not written by "J.K. Rowling"

db.Books.find({ author: { $not: /J\.K\. Rowling/ } })

// 6. Conditional semantics - Find books with 4 or more copies available

db.Books.find({ copies\_available: { $gte: 4 } })

// 7. Type-specific query for `Null` - Find books without the `tags` field (demonstration only, requires such data)

db.Books.find({ tags: { $exists: false } })

// 8. Using regular expressions - Find all books with "Harry" in the title

db.Books.find({ title: /Harry/ })

// 9. Querying arrays - Find books tagged as "classic"

db.Books.find({ tags: "classic" })

// 10. `$where` query - Find books where the number of copies available is greater than the published year modulo 100 (just an example condition)

db.Books.find({ $where: "this.copies\_available > (this.published\_year % 100)" })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to PopulationDB database

use PopulationDB

// Create the Population collection and insert sample data

db.createCollection("Population")

// Insert documents into the Population collection

db.Population.insertMany([

{ CountryId: 1, statId: 101, City: "New York", year: 2020, Population: 8419600 },

{ CountryId: 1, statId: 102, City: "Los Angeles", year: 2020, Population: 3980400 },

{ CountryId: 1, statId: 103, City: "Chicago", year: 2020, Population: 2716000 },

{ CountryId: 2, statId: 201, City: "London", year: 2020, Population: 8982000 },

{ CountryId: 2, statId: 202, City: "Manchester", year: 2020, Population: 547627 },

{ CountryId: 3, statId: 301, City: "Tokyo", year: 2020, Population: 13929286 },

{ CountryId: 3, statId: 302, City: "Osaka", year: 2020, Population: 2698000 },

{ CountryId: 4, statId: 401, City: "Delhi", year: 2020, Population: 30290936 },

{ CountryId: 4, statId: 402, City: "Mumbai", year: 2020, Population: 20185064 }

])

// Implement Map-Reduce to get the total population of each country

// Map function to emit CountryId as the key and Population as the value

var mapFunction = function() {

emit(this.CountryId, this.Population);

};

// Reduce function to sum the Population for each CountryId

var reduceFunction = function(key, values) {

return Array.sum(values);

};

// Run the map-reduce operation

db.Population.mapReduce(

mapFunction,

reduceFunction,

{ out: "TotalPopulationPerCountry" }

)

// Display the results from the map-reduce operation

db.TotalPopulationPerCountry.find()

// Indexing on CountryId to optimize population queries by country

db.Population.createIndex({ CountryId: 1 })

// Indexing on year to optimize year-specific queries

db.Population.createIndex({ year: 1 })

// Indexing on City and Population for optimized city-specific population queries

db.Population.createIndex({ City: 1, Population: 1 })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*9\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to PopulationDB database

use PopulationDB

// Create the Population collection

db.createCollection("Population")

// Insert sample documents into the Population collection

db.Population.insertMany([

{ CountryId: 1, statId: 101, City: "New York", year: 2020, Population: 8419600 },

{ CountryId: 1, statId: 102, City: "Los Angeles", year: 2020, Population: 3980400 },

{ CountryId: 1, statId: 103, City: "Chicago", year: 2020, Population: 2716000 },

{ CountryId: 2, statId: 201, City: "London", year: 2020, Population: 8982000 },

{ CountryId: 2, statId: 202, City: "Manchester", year: 2020, Population: 547627 },

{ CountryId: 3, statId: 301, City: "Tokyo", year: 2020, Population: 13929286 },

{ CountryId: 3, statId: 302, City: "Osaka", year: 2020, Population: 2698000 },

{ CountryId: 4, statId: 401, City: "Delhi", year: 2020, Population: 30290936 },

{ CountryId: 4, statId: 402, City: "Mumbai", year: 2020, Population: 20185064 }

])

// Aggregation example to calculate total population, min, max, avg, count, first, last, push, and addToSet by CountryId

const aggregationResult = db.Population.aggregate([

{

$group: {

\_id: "$CountryId",

totalPopulation: { $sum: "$Population" }, // Sum of populations

minPopulation: { $min: "$Population" }, // Minimum population

maxPopulation: { $max: "$Population" }, // Maximum population

avgPopulation: { $avg: "$Population" }, // Average population

cityCount: { $count: {} }, // Count of cities

firstCity: { $first: "$City" }, // First city in the group

lastCity: { $last: "$City" }, // Last city in the group

cityList: { $push: "$City" }, // Array of cities

uniqueCities: { $addToSet: "$City" } // Unique cities

}

}

])

// Print aggregation result

printjson(aggregationResult)

// Indexing on CountryId to optimize aggregation queries by country

db.Population.createIndex({ CountryId: 1 })

// Indexing on year to optimize year-specific queries

db.Population.createIndex({ year: 1 })

// Indexing on City and Population for optimized city-specific population queries

db.Population.createIndex({ City: 1, Population: 1 })

\*\*\*\*\*\*\*\*\*\*\*\*10\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STAR SCHEMA

// Switch to OnlineSalesDB database

use OnlineSalesDB

// Create the collections for Star Schema

// Fact Table: Sales

db.createCollection("Sales")

db.Sales.insertMany([

{ SaleId: 1, ProductId: 101, CustomerId: 201, DateId: 301, Amount: 150.00 },

{ SaleId: 2, ProductId: 102, CustomerId: 202, DateId: 302, Amount: 200.00 },

{ SaleId: 3, ProductId: 101, CustomerId: 201, DateId: 303, Amount: 120.00 },

{ SaleId: 4, ProductId: 103, CustomerId: 203, DateId: 301, Amount: 300.00 },

{ SaleId: 5, ProductId: 104, CustomerId: 204, DateId: 302, Amount: 250.00 }

])

// Dimension Table: Product

db.createCollection("Product")

db.Product.insertMany([

{ ProductId: 101, ProductName: "Laptop", CategoryId: 1, Price: 1000 },

{ ProductId: 102, ProductName: "Smartphone", CategoryId: 2, Price: 500 },

{ ProductId: 103, ProductName: "Tablet", CategoryId: 1, Price: 300 },

{ ProductId: 104, ProductName: "Headphones", CategoryId: 3, Price: 100 }

])

// Dimension Table: Customer

db.createCollection("Customer")

db.Customer.insertMany([

{ CustomerId: 201, CustomerName: "Alice", Email: "alice@example.com", Address: "123 Main St" },

{ CustomerId: 202, CustomerName: "Bob", Email: "bob@example.com", Address: "456 Elm St" },

{ CustomerId: 203, CustomerName: "Charlie", Email: "charlie@example.com", Address: "789 Maple St" },

{ CustomerId: 204, CustomerName: "Diana", Email: "diana@example.com", Address: "321 Oak St" }

])

// Dimension Table: Date

db.createCollection("Date")

db.Date.insertMany([

{ DateId: 301, Date: "2024-01-01", Month: "January", Year: 2024 },

{ DateId: 302, Date: "2024-01-02", Month: "January", Year: 2024 },

{ DateId: 303, Date: "2024-01-03", Month: "January", Year: 2024 }

])

SNOWFLAKE:

// Create the collections for Snowflake Schema

// Fact Table: Sales

db.createCollection("Sales")

db.Sales.insertMany([

{ SaleId: 1, ProductId: 101, CustomerId: 201, DateId: 301, Amount: 150.00 },

{ SaleId: 2, ProductId: 102, CustomerId: 202, DateId: 302, Amount: 200.00 },

{ SaleId: 3, ProductId: 101, CustomerId: 201, DateId: 303, Amount: 120.00 },

{ SaleId: 4, ProductId: 103, CustomerId: 203, DateId: 301, Amount: 300.00 },

{ SaleId: 5, ProductId: 104, CustomerId: 204, DateId: 302, Amount: 250.00 }

])

// Dimension Table: Product

db.createCollection("Product")

db.Product.insertMany([

{ ProductId: 101, ProductName: "Laptop", CategoryId: 1, Price: 1000 },

{ ProductId: 102, ProductName: "Smartphone", CategoryId: 2, Price: 500 },

{ ProductId: 103, ProductName: "Tablet", CategoryId: 1, Price: 300 },

{ ProductId: 104, ProductName: "Headphones", CategoryId: 3, Price: 100 }

])

// Dimension Table: Category

db.createCollection("Category")

db.Category.insertMany([

{ CategoryId: 1, CategoryName: "Electronics" },

{ CategoryId: 2, CategoryName: "Mobiles" },

{ CategoryId: 3, CategoryName: "Accessories" }

])

// Dimension Table: Customer

db.createCollection("Customer")

db.Customer.insertMany([

{ CustomerId: 201, CustomerName: "Alice", Email: "alice@example.com", Address: "123 Main St" },

{ CustomerId: 202, CustomerName: "Bob", Email: "bob@example.com", Address: "456 Elm St" },

{ CustomerId: 203, CustomerName: "Charlie", Email: "charlie@example.com", Address: "789 Maple St" },

{ CustomerId: 204, CustomerName: "Diana", Email: "diana@example.com", Address: "321 Oak St" }

])

// Dimension Table: Date

db.createCollection("Date")

db.Date.insertMany([

{ DateId: 301, Date: "2024-01-01", Month: "January", Year: 2024 },

{ DateId: 302, Date: "2024-01-02", Month: "January", Year: 2024 },

{ DateId: 303, Date: "2024-01-03", Month: "January", Year: 2024 }

])

### Star Schema Tables:

- \*\*Fact Table\*\*: `Sales` (SaleId, ProductId, CustomerId, DateId, Amount)

- \*\*Dimension Tables\*\*: `Product` (ProductId, ProductName, CategoryId, Price), `Customer` (CustomerId, CustomerName, Email, Address), `Date` (DateId, Date, Month, Year)

### Snowflake Schema Tables:

- \*\*Fact Table\*\*: `Sales` (SaleId, ProductId, CustomerId, DateId, Amount)

- \*\*Dimension Tables\*\*: `Product` (ProductId, ProductName, CategoryId, Price), `Category` (CategoryId, CategoryName), `Customer` (CustomerId, CustomerName, Email, Address), `Date` (DateId, Date, Month, Year)

\*\*\*\*\*\*\*\*\*\*\*\*11\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to AdmissionManagementDB database

use AdmissionManagementDB

// Create the collection for Admission

db.createCollection("Admissions")

// Inserting and saving documents (batch insert, insert validation)

db.Admissions.insertMany([

{ StudentId: 1, Name: "Alice", Age: 18, Course: "Computer Science", Status: "Applied" },

{ StudentId: 2, Name: "Bob", Age: 19, Course: "Electrical Engineering", Status: "Admitted" },

{ StudentId: 3, Name: "Charlie", Age: 20, Course: "Mechanical Engineering", Status: "Rejected" },

{ StudentId: 4, Name: "Diana", Age: 17, Course: "Civil Engineering", Status: "Applied" }

], { ordered: true }) // Ensure that documents are validated upon insertion

// Removing a document

db.Admissions.deleteOne({ StudentId: 3 }) // Remove the student with StudentId 3

// Updating documents

// Document replacement

db.Admissions.replaceOne({ StudentId: 2 }, { StudentId: 2, Name: "Bob", Age: 19, Course: "Electronics", Status: "Admitted" })

// Using modifiers (upserts)

db.Admissions.updateOne(

{ StudentId: 5 }, // Condition

{ $set: { Name: "Eva", Age: 18, Course: "Data Science", Status: "Applied" } }, // Update

{ upsert: true } // Create new document if it doesn't exist

)

// Updating multiple documents

db.Admissions.updateMany(

{ Status: "Applied" },

{ $set: { Status: "Processing" } } // Update status for all students who have applied

)

// Returning updated documents

const updatedDocuments = db.Admissions.find({ Status: "Processing" }).toArray()

printjson(updatedDocuments) // Output updated documents to the console

// Cursors (Limit, skip, sort, advanced query options)

// Example: Find students, limit to 2, skip 1, sort by Name

const cursor = db.Admissions.find().sort({ Name: 1 }).skip(1).limit(2)

cursor.forEach(doc => printjson(doc)) // Print each document in the cursor

// Advanced query options: Find students who are 18 or older and applied

const advancedQueryCursor = db.Admissions.find({ $or: [{ Age: { $gte: 18 } }, { Status: "Applied" }] })

advancedQueryCursor.forEach(doc => printjson(doc)) // Print documents from the advanced query

\*\*\*\*\*\*\*\*\*\*\*\*12\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to AdmissionManagementDB database

use AdmissionManagementDB

// Create the collection for Admissions

db.createCollection("Admissions")

// Insert sample data

db.Admissions.insertMany([

{ StudentId: 1, Name: "Alice", Age: 18, Course: "Computer Science", Status: "Applied", Subjects: ["Math", "Physics"] },

{ StudentId: 2, Name: "Bob", Age: 19, Course: "Electrical Engineering", Status: "Admitted", Subjects: ["Math", "Chemistry"] },

{ StudentId: 3, Name: "Charlie", Age: 20, Course: "Mechanical Engineering", Status: "Rejected", Subjects: ["Physics", "Chemistry"] },

{ StudentId: 4, Name: "Diana", Age: 17, Course: "Civil Engineering", Status: "Applied", Subjects: ["Math"] },

{ StudentId: 5, Name: "Eva", Age: 21, Course: "Data Science", Status: "Admitted", Subjects: null },

{ StudentId: 6, Name: "Frank", Age: 22, Course: "Biotechnology", Status: "Applied", Subjects: ["Biology", "Chemistry"] },

{ StudentId: 7, Name: "Grace", Age: 23, Course: "Architecture", Status: "Rejected", Subjects: ["Design", "Math"] },

{ StudentId: 8, Name: "Hannah", Age: 19, Course: "Mathematics", Status: "Admitted", Subjects: ["Math"] },

{ StudentId: 9, Name: "Ian", Age: 20, Course: "Computer Science", Status: "Applied", Subjects: ["Math", "Computer Science"] },

{ StudentId: 10, Name: "Jack", Age: 22, Course: "Electrical Engineering", Status: "Admitted", Subjects: ["Math", "Physics"] }

])

// Query 1: Find a specific student by StudentId

db.Admissions.findOne({ StudentId: 1 }) // Find student with StudentId 1

// Query 2: Find all students who are "Admitted"

db.Admissions.find({ Status: "Admitted" }) // Find all admitted students

// Query 3: Query criteria with conditionals (find students older than 20)

db.Admissions.find({ Age: { $gt: 20 } }) // Find students older than 20

// Query 4: OR query (find students who are either "Applied" or "Rejected")

db.Admissions.find({ $or: [{ Status: "Applied" }, { Status: "Rejected" }] })

// Query 5: Using $not (find students who are not in a specific course)

db.Admissions.find({ Course: { $not: { $eq: "Computer Science" } } }) // Students not in Computer Science

// Query 6: Conditional semantics (find students with Age not specified)

db.Admissions.find({ Age: { $exists: false } }) // Find students where Age field does not exist

// Query 7: Type-specific query (find students with Subjects set to null)

db.Admissions.find({ Subjects: null }) // Find students with no subjects

// Query 8: Regular expression (find students whose names start with 'A')

db.Admissions.find({ Name: { $regex: /^A/, $options: 'i' } }) // Find names starting with 'A'

// Query 9: Querying arrays (find students who have "Math" as one of their subjects)

db.Admissions.find({ Subjects: "Math" }) // Find students who have Math in their subjects

// Query 10: Using $where (find students who are older than the average age of 20)

db.Admissions.find({ $where: "this.Age > 20" }) // Find students older than 20 using $where

\*\*\*\*\*\*\*\*\*\*\*\*13\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to AdmissionManagementDB database

use AdmissionManagementDB

// Create the UnitTest collection

db.createCollection("UnitTest")

// Insert sample data into UnitTest collection

db.UnitTest.insertMany([

{ TestId: 1, StudentId: 101, year: 2023, totalmarks: 85 },

{ TestId: 1, StudentId: 102, year: 2023, totalmarks: 78 },

{ TestId: 1, StudentId: 103, year: 2023, totalmarks: 92 },

{ TestId: 2, StudentId: 101, year: 2023, totalmarks: 88 },

{ TestId: 2, StudentId: 102, year: 2023, totalmarks: 75 },

{ TestId: 2, StudentId: 103, year: 2023, totalmarks: 80 },

{ TestId: 3, StudentId: 101, year: 2023, totalmarks: 90 },

{ TestId: 3, StudentId: 102, year: 2023, totalmarks: 82 },

{ TestId: 3, StudentId: 103, year: 2023, totalmarks: 95 }

])

// Create an index on the TestId field for efficient querying

db.UnitTest.createIndex({ TestId: 1 })

// Define the Map function to emit TestId and totalmarks

var mapFunction = function() {

emit(this.TestId, this.totalmarks);

};

// Define the Reduce function to calculate the average

var reduceFunction = function(key, values) {

return Array.sum(values) / values.length;

};

// Execute the MapReduce operation

db.UnitTest.mapReduce(

mapFunction,

reduceFunction,

{

out: "AverageMarksByTest", // Output collection for results

finalize: function(key, reducedValue) {

return reducedValue; // Finalize the average value

}

}

);

// Fetch and display the results

db.AverageMarksByTest.find().forEach(printjson)

\*\*\*\*\*\*\*\*14\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to AdmissionManagementDB database

use AdmissionManagementDB

// Create the UnitTest collection

db.createCollection("UnitTest")

// Insert sample data into UnitTest collection

db.UnitTest.insertMany([

{ TestId: 1, StudentId: 101, year: 2023, totalmarks: 85 },

{ TestId: 1, StudentId: 102, year: 2023, totalmarks: 78 },

{ TestId: 1, StudentId: 103, year: 2023, totalmarks: 92 },

{ TestId: 2, StudentId: 101, year: 2023, totalmarks: 88 },

{ TestId: 2, StudentId: 102, year: 2023, totalmarks: 75 },

{ TestId: 2, StudentId: 103, year: 2023, totalmarks: 80 },

{ TestId: 3, StudentId: 101, year: 2023, totalmarks: 90 },

{ TestId: 3, StudentId: 102, year: 2023, totalmarks: 82 },

{ TestId: 3, StudentId: 103, year: 2023, totalmarks: 95 }

])

// Create an index on the TestId field for efficient querying

db.UnitTest.createIndex({ TestId: 1 })

// Aggregation example

db.UnitTest.aggregate([

{

$group: {

\_id: "$TestId",

totalMarks: { $sum: "$totalmarks" }, // Sum of total marks

minMarks: { $min: "$totalmarks" }, // Minimum marks

maxMarks: { $max: "$totalmarks" }, // Maximum marks

avgMarks: { $avg: "$totalmarks" }, // Average marks

count: { $count: {} }, // Count of students

firstMark: { $first: "$totalmarks" }, // First total marks

lastMark: { $last: "$totalmarks" }, // Last total marks

allMarks: { $push: "$totalmarks" }, // Push all marks into an array

uniqueStudents: { $addToSet: "$StudentId" } // Unique Student IDs

}

},

{

$sort: { \_id: 1 } // Sort results by TestId

}

]).forEach(printjson)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*15\*\*\*\*\*\*\*\*\*\*\*\*

// Switch to the UniversityDB database

use UniversityDB

// Create collections for the Star Schema

db.createCollection("Fact\_Enrollment")

db.createCollection("Dim\_Student")

db.createCollection("Dim\_Course")

db.createCollection("Dim\_Semester")

db.createCollection("Dim\_Department")

// Insert sample data for Dim\_Student

db.Dim\_Student.insertMany([

{ StudentID: 1, FirstName: "Alice", LastName: "Smith", DOB: new Date("2000-05-15"), Major: "Computer Science", EnrollmentDate: new Date("2018-08-20") },

{ StudentID: 2, FirstName: "Bob", LastName: "Johnson", DOB: new Date("1999-09-22"), Major: "Mathematics", EnrollmentDate: new Date("2017-08-20") },

{ StudentID: 3, FirstName: "Charlie", LastName: "Williams", DOB: new Date("2001-02-10"), Major: "Physics", EnrollmentDate: new Date("2019-08-20") }

])

// Insert sample data for Dim\_Course

db.Dim\_Course.insertMany([

{ CourseID: 1, CourseName: "Data Structures", CourseDescription: "Introduction to Data Structures", Credits: 3, Department: "Computer Science" },

{ CourseID: 2, CourseName: "Calculus", CourseDescription: "Calculus I", Credits: 4, Department: "Mathematics" },

{ CourseID: 3, CourseName: "Physics I", CourseDescription: "Introduction to Physics", Credits: 4, Department: "Physics" }

])

// Insert sample data for Dim\_Semester

db.Dim\_Semester.insertMany([

{ SemesterID: 1, SemesterName: "Fall", Year: 2023 },

{ SemesterID: 2, SemesterName: "Spring", Year: 2024 }

])

// Insert sample data for Dim\_Department

db.Dim\_Department.insertMany([

{ DepartmentID: 1, DepartmentName: "Computer Science", HeadOfDepartment: "Dr. Alice Brown" },

{ DepartmentID: 2, DepartmentName: "Mathematics", HeadOfDepartment: "Dr. Bob White" },

{ DepartmentID: 3, DepartmentName: "Physics", HeadOfDepartment: "Dr. Charlie Black" }

])

// Insert sample data for Fact\_Enrollment

db.Fact\_Enrollment.insertMany([

{ EnrollmentID: 1, StudentID: 1, CourseID: 1, SemesterID: 1, Year: 2023, Grade: "A" },

{ EnrollmentID: 2, StudentID: 2, CourseID: 2, SemesterID: 1, Year: 2023, Grade: "B" },

{ EnrollmentID: 3, StudentID: 3, CourseID: 3, SemesterID: 2, Year: 2024, Grade: "C" },

{ EnrollmentID: 4, StudentID: 1, CourseID: 2, SemesterID: 2, Year: 2024, Grade: "A" },

{ EnrollmentID: 5, StudentID: 2, CourseID: 3, SemesterID: 1, Year: 2023, Grade: "B" }

])

// Create indexes for faster querying

db.Fact\_Enrollment.createIndex({ StudentID: 1 })

db.Fact\_Enrollment.createIndex({ CourseID: 1 })

db.Fact\_Enrollment.createIndex({ SemesterID: 1 })

db.Dim\_Student.createIndex({ StudentID: 1 })

db.Dim\_Course.createIndex({ CourseID: 1 })

db.Dim\_Semester.createIndex({ SemesterID: 1 })

db.Dim\_Department.createIndex({ DepartmentID: 1 })

// Sample query to fetch enrollments with student details

db.Fact\_Enrollment.aggregate([

{

$lookup: {

from: "Dim\_Student",

localField: "StudentID",

foreignField: "StudentID",

as: "StudentDetails"

}

},

{

$lookup: {

from: "Dim\_Course",

localField: "CourseID",

foreignField: "CourseID",

as: "CourseDetails"

}

},

{

$lookup: {

from: "Dim\_Semester",

localField: "SemesterID",

foreignField: "SemesterID",

as: "SemesterDetails"

}

},

{

$unwind: "$StudentDetails"

},

{

$unwind: "$CourseDetails"

},

{

$unwind: "$SemesterDetails"

},

{

$project: {

\_id: 0,

EnrollmentID: 1,

"StudentDetails.FirstName": 1,

"StudentDetails.LastName": 1,

"CourseDetails.CourseName": 1,

"SemesterDetails.SemesterName": 1,

Year: 1,

Grade: 1

}

}

]).pretty()