# MongoDB Assignment - Solutions

## Part 1 – Basics of MongoDB

1. MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. Unlike relational databases, it does not rely on tables and rows but instead uses collections and documents, which makes it highly scalable and flexible.

2. Databases: Logical containers for collections.  
 Collections: Groups of related documents.  
 Documents: Individual records stored in BSON (Binary JSON) format.

3. MongoDB stores documents in BSON (Binary JSON) format.

4. Ways to install or use MongoDB:  
 - Download and install from MongoDB official website.  
 - Use MongoDB Atlas (cloud service).  
 - Use Docker image for MongoDB.

5. MongoDB Compass is the GUI tool provided by MongoDB.

6. To verify installation: Open terminal as Admin and run `mongod --version` or `mongo --version`.

## Part 2 – MongoDB Commands

1. Show all databases: show dbs
2. Switch to or create a database called mydb: use mydb
3. Create a collection called users: db.createCollection('users')
4. Insert one user: db.users.insertOne({ name: “Abhi”, age: 20, city: “Naraingarh” })
5. Insert multiple users: db.users.insertMany([  
    { name: 'Abhishek', age: 20, city: 'Mulana' },   
    { name: 'Manu', age: 20, city: 'Naraingarh' }  
    ])
6. View all documents in users: db.users.find()
7. Find a user named Alice: db.users.find({ name: 'Alice' })
8. Find users with age > 25: db.users.find({ age: { $gt: 25 } })

## Part 3 – CRUD Operations

1. Update one user's city to Berlin: db.users.updateOne({ name: 'John' }, { $set: { city: 'Berlin' } })
2. Update all users from London to add status active: db.users.updateMany({ city: 'London' }, { $set: { status: 'active' } })
3. Delete one user named Alice: db.users.deleteOne({ name: 'Alice' })
4. Delete all users from Paris: db.users.deleteMany({ city: 'Paris' })

## Part 4 – Query Operators & Projection

1. Operator for greater value: $gt
2. Operator for not equal: $ne
3. Find users from London or Paris: db.users.find({ city: { $in: ['London', 'Paris'] } })
4. Find users from London OR Berlin: db.users.find({ city: { $in: ['London', 'Berlin'] } })
5. Show only name and city fields while hiding \_id : db.users.find({}, { name: 1, city: 1, \_id: 0 })

## Part 5 – Relations & Schema Design

1. Being schema-less means MongoDB collections do not enforce a fixed structure, allowing documents with different fields in the same collection.

2. Embedding stores related data within a single document, while referencing links documents using ObjectId references.

3. Embedded document example:  
 ```  
 { name: 'John', orders: [{ item: 'Laptop', price: 120000 }, { item: 'Mouse', price: 400 }] }  
 ```

4. Referenced schema example:  
 ```  
 Users: { \_id: ObjectId(), name: 'Abhi' }  
 Orders: { \_id: ObjectId(), userId: ObjectId('...'), item: 'Laptop', price: 120000 }  
 ```

5. For small related data, embedding is preferred.

6. Schema design is important for optimizing queries, ensuring consistency, and improving performance.

## Part 6 – Aggregation Framework

1. Aggregation in MongoDB is a framework for processing data and transforming it into meaningful results, such as summaries or computed values.  
   It is more powerful than **find()** because:

* It supports advanced operations like grouping, filtering, sorting, joining, and transforming data.
* It allows complex pipelines to analyze data in multiple stages.
* It can perform operations similar to SQL GROUP BY, JOIN, and aggregate functions in one query.

2. Find only delivered orders ($match): db.orders.aggregate([

{ $match: { status: "delivered" } }

]);

3. Show only customerName and total fields ($project): db.orders.aggregate([  
 { $project: { customerName: 1, total: 1, \_id: 0 } }  
 ])

4. Total amount spent by each customer ($group):   
db.orders.aggregate([  
 { $group: { \_id: '$customerName', totalSpent: { $sum: '$total' } } }  
 ])

5. $unwind usage: The $unwind stage deconstructs an array field into multiple documents, one for each element of the array.

6. Join orders with customers using $lookup:  
 db.orders.aggregate([  
{ $lookup: {   
from: 'customers',  
 localField: 'customerId',  
 foreignField: '\_id',   
as: 'customerDetails' } }])

7. Top 2 highest orders ($sort and $limit):   
db.orders.aggregate([  
 { $sort: { total: -1 } },  
 { $limit: 2 }  
 ])