BT:

1.1) **Embedding Documents:** In MongoDB, embedding is a way to handle relationships where one document is nested inside another. This is a natural fit for "one - to - many" relationships. For example, consider a blogging application. A blog post document can embed multiple comment documents. The embedded documents become part of the parent document's structure.

2) **Document Referencing:** Referencing involves storing a reference (usually the **_id** of another document) in one document to establish a relationship. This is useful for "many - to - many" or more complex "one - to - many" relationships. For example, in a social networking system, a user document might store an array of the _id values of the users they follow. The actual details of the followed users are stored in separate documents, and the reference in the user's document points to these other documents.

3)**DB Refs (Database References):** DB Refs are a special structure in MongoDB used to represent references between documents. A DB Ref contains three parts: $ref (the name of the collection being referenced), $id (the _id of the referenced document), and $db (optionally, the name of the database where the referenced document is located). For example, in a scenario where you need to reference documents across different databases, the $db field becomes useful.

2. 1) **Relational Database Approach (e.g., PostgreSQL), Explicit Schema Definition:** In a relational database like PostgreSQL, the CREATE TABLE statement is used to define the structure of a table. The columns (custId, contact, loan in the example) are explicitly declared with their data types (VARCHAR, INT). The PRIMARY KEY constraint is also defined for the custId column. This means that the database enforces a strict schema. When inserting data using the INSERT INTO statement, the values must match the defined data types and any constraints. For example, if you try to insert a non - integer value into the contact column, the database will raise an error because the column is defined as INT.

2)**MongoDB Approach Schema - Less Insertion:** In MongoDB, the insert One operation creates a new document in a collection (customers in the example) without the need to pre - define a schema. The document is simply a JSON - like structure with key - value pairs ({ custId: 'zxc456', contact: 3355, loan: 2000 }). MongoDB allows you to insert documents with different sets of fields into the same collection. For example, you could insert another document into the customers collection that has additional fields like address or is missing the loan field.

3) MongoDB is considered schema - less because it doesn't enforce a fixed schema like

a traditional relational database. The structure of documents in a collection can vary from one document to another. You can insert documents with different sets of fields and data types into the same collection. There is no requirement to define columns and their data types in advance. The database stores the documents as they are, allowing for a more flexible and dynamic data model that can adapt to changing application requirements over time.

3. MongoDB is referred to as having a "dynamic schema" because of the following characteristics demonstrated by the given sample of documents: Variable Field Existence, Data Type Flexibility, Schema Evolution

MT:

4.

a)// Insert the first employee document

```
db.employees.insertOne({
    empNo: 'E55',
    lastName: 'Carmen',
    firstName: 'Silva',
    hours: 450,
    gender: 'F'
});


// Insert the second employee document
db.employees.insertOne({
    empNo: 'E44',
    lastName: 'Ryad',
    firstName: 'Patel',
    hours: 380,
    gender: 'M'
});
```

```
// Insert the third employee document
db.employees.insertOne({
    empNo: 'E66',
    lastName: 'Susan',
    firstName: 'Joyner',
    hours: 350,
    gender: 'F'
});

// Insert the fourth employee document
db.employees.insertOne({
    empNo: 'E77',
    lastName: 'Waiman',
    firstName: 'Zhu',
    hours: 400,
    gender: 'M'
});
```

b)

```
db.employees.find({}, { firstName: 1, lastName: 1, _id: 0 });
db.employees.find({ hours: { $gt: 400 } }, { empNo: 1, lastName: 1, hours: 1, _id: 0 });
db.employees.find({ lastName: { $regex: '^S' } }, { lastName: 1, _id: 0 });
db.employees.find({ gender: 'F' }, { firstName: 1, lastName: 1, _id: 0 }).sort({ firstName: 1 });
db.employees.countDocuments({ hours: { $lt: 400 } });
db.employees.find({
```

```
    $or: [
        { gender: 'F' },
        { hours: { $gt: 350 } }
    ]
}, { firstName: 1, lastName: 1, _id: 0 }).sort({ lastName: 1 });
```

5.

a) Volume, Velocity, Variety, Veracity, Value

b) Structured Data, Semi - Structured Data, Unstructured Data, Metadata

c) Flexible Document - Based Model, Schema - Less Design, Support for Multiple Data Types, ndexing and Querying Capabilities

AT:

a)

```
{
    "book_id": "552020",
    "author": " D. Sullivan",
    "title": "NoSQL for Mere Mortals",
    "publisher": "Addison-Wesley",
    "year": "2019",
    "ISBN": "9080134023212",
    "comments": [
        {
            "author": "Anonymous",
            "text": "How do I get a copy?"
        }
    ]
}
```

```
{
    "book_id": "3450",
    "authors": [" P. Saladage", "M. Fowler"],
    "title": "NoSQL Distilled",
    "publisher": "Addison-Wesley",
    "year": "2022",
    "ISBN": "9080321826626",
    "comments": [
        {
            "author": "Matt",
            "text": "Nice overview of SQL"
        },
        {
            "author": "Thomas",
            "text": "Still relevant"
        }
    ]
}
```


b)

i) To find the books that were published between 2019 and 2024:

```
db.books.find({
    "year": {
        $gte: "2019",
        $lte: "2024"
    }
});
```

ii) To find the books whose book_id is not equal to 552020:

```
db.books.find({
    "book_id": {
        $ne: "552020"
    }
});
```

iii) To find the books by D. Sullivan or whose ISBN is equal to 9780134023212:

```
db.books.find({
    $or: [
        {
            "author": "D. Sullivan"
        },
        {
            "ISBN": "9780134023212"
        }
    ]
});
```

iv) To find the books whose ISBN is equal to 9876543210 or to 0123456789:

```
db.books.find({
    $or: [
        {
            "ISBN": "9876543210"
        },
        {
            "ISBN": "0123456789"
        }
    ]
});
```

v) To find the books whose title contains the string "SQL":

```
db.books.find({
    "title": {
        $regex: "SQL"
    }
});
```

vi) To find the number of books published by Addison-Wesley:

```
db.books.countDocuments({
    "publisher": "Addison-Wesley"
});
```

vii) To find the books published in 2019 and whose title contains the string "Mortals", sorted by title in alphabetical order:

```
db.books.find({
    "year": "2019",
    "title": {
        $regex: "Mortals"
    }
}).sort({
    "title": 1
});
```

viii) To add another field named "subject" with the value "computing" to the books published in 2019:

```
db.books.updateMany({
    "year": "2019"
}, {
    $set: {
        "subject": "computing"
```

```
        }

});
```