1. How MongoDB handles document relationships

- MongoDB handles relationships between documents through embedding and referencing.
    - **Embedding**: Nest related data directly within a document. For example, a document containing user order information can be embedded with information such as order details, shipping address, and so on. This method is suitable for situations where child documents and parent documents are closely related and frequently queried together, and all related data can be obtained through a single database operation, reducing the number of queries and improving performance.
    - **Referencing**: Holds a reference to another document in one document (similar to a foreign key in a relational database). For example, in a user document, `all orders for that user can be referenced by saving the _id of the order document. When the relationship is complex or the amount of data is large, the reference method can avoid data redundancy and improve the efficiency of data update, but it may take multiple queries to obtain complete related data.`

2. Comments on relational databases and the way MongoDB creates tables/collections and why MongoDB is considered "schema-less".

- **Relational database (SQL) is used to create tables**
    - Use the CREATE TABLE statement to explicitly define the structure of the table, including field names, data types, and constraints (such as primary keys, non-null, etc.). When inserting data, it must be provided strictly according to the defined structure, otherwise the insertion will fail. In scenarios where the data structure is relatively fixed and clear, this method can ensure the consistency and integrity of the data, and facilitate the database management system to store, query, optimize, and verify the data.
- **MongoDB creates collections**
    - Insert documents directly through operations such as insertOne, eliminating the need to pre-define a strict structure. For example, in the example, if a document containing `the custId`, contact, and loan fields is inserted directly, `the subsequent inserted document can contain different fields (e.g., another document contains the custId`, contact, `and credit fields).` The reason why MongoDB is considered "schemaless" is that it does not have mandatory pre-defined requirements for the structure of documents when inserting data, and each

document can have different fields, which is more flexible and suitable for scenarios where the data structure is not fixed or may change over time, and can quickly adapt to changes in business needs, but it may also lead to relatively complex data consistency and integrity maintenance, which requires more control at the application layer.

3. MongoDB 被称为具有“动态模式（dynamic schema）”的原因

- As you can see from the given example documents, documents in a MongoDB collection can have different fields. The first document contains `the custId`, contact, `and loan fields; The second document contains the custId`, contact, `and credit fields; The third document contains the custId` and Profession fields. This means that documents in the same collection do not need to follow a unified structural pattern, and fields can be added or removed from the documents at any time during the application run, according to actual needs, without first modifying the table structure like relational databases, reflecting the characteristics of its dynamic schema, which can better adapt to different data structures and business changes.

4. Translate SQL statements into MongoDB statements

- **a) CREATE / INSERT Statements（创建 / 插入语句）**
    - **SQL statement**
- CREATE TABLE employee (
- empNo CHAR(9) PRIMARY KEY NOT NULL,
- lastName CHAR(25) NOT NULL,
- firstName VARCHAR(25) NOT NULL,
- hours INT,
- gender CHAR
-);
- INSERT INTO employee (empNo, lastName, firstName, hours, gender) VALUES('E55','Carmen','Silva',450,'F');
- INSERT INTO employee (empNo, lastName, firstName, hours, gender) VALUES('E44','Ryad','Patel', 380,'M');
- INSERT INTO employee (empNo, lastName, firstName, hours, gender) VALUES('E66','Susan', 'Joyner',350, 'F');
- INSERT INTO employee (empNo, lastName, firstName, hours, gender) VALUES('E77','Waiman', 'Zhu', 400, 'M');
-

MongoDB statement

Inserting multiple documents is equivalent to the INSERT INTO of SQL multiple executions

```
db.employee.insertMany([
  { empNo: 'E55', lastName: 'Carmen', firstName: 'Silva', hours: 450, gender: 'F' },
```

```
    { empNo: 'E44', lastName: 'Ryad', firstName: 'Patel', hours: 380, gender: 'M' },
    { empNo: 'E66', lastName: 'Susan', firstName: 'Joyner', hours: 350, gender: 'F' },
    { empNo: 'E77', lastName: 'Waiman', firstName: 'Zhu', hours: 400, gender: 'M' }
]);
```

- **b) Select /find statements（查询语句）**
    - **SQL statement**

```
-- Query the first and last names of all employees
SELECT firstName, lastName FROM employee;
-- Query the number, last name, and working hours of employees who have worked more than 400 hours
SELECT empNo, lastName, hours FROM employee WHERE hours > 400;
-- Query the last name of an employee whose last name starts with 'S'
SELECT lastName FROM employee WHERE lastName LIKE 'S%';
-- Query the first and last names of female employees and sort them by first name
SELECT firstName, lastName FROM employee WHERE gender = 'F' ORDER BY firstName;
-- Query the number of employees whose working hours are less than 400 hours
SELECT COUNT(empNo) FROM employee WHERE (hours <400);
-- Query the first and last names of female employees or employees who have worked more than 350 hours, sorted by last name
SELECT firstName, lastName FROM employee WHERE (gender = 'F') OR (hours > 350) ORDER BY lastName;
```

MongoDB statement

Check the first and last names of all employees

```
db.employee.find({}, { firstName: 1, lastName: 1, _id: 0 });
```

Query the number, surname, and working hours of employees whose working hours are greater than 400 hours

```
db.employee.find({ hours: { $gt: 400 } }, { empNo: 1, lastName: 1, hours: 1, _id: 0 });
```

Look up the last name of an employee whose last name starts with 'S'

```
db.employee.find({ lastName: /^S/ }, { lastName: 1, _id: 0 });
```

Query the first and last names of female employees and sort them by first name

```
db.employee.find({ gender: 'F' }, { firstName: 1, lastName: 1, _id: 0 }).sort({ firstName: 1 });
```

Query the number of employees whose working hours are less than 400 hours

```
db.employee.countDocuments({ hours: { $lt: 400 } });
```

Query the first and last names of female employees or employees who work more than 350 hours, and sort them by last name

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

5.

- **a) Comment on the nature of big data from the "V(s)" characteristic**
    - **Volume**: Big data often means that the amount of data is huge, far beyond the scale that traditional database systems can easily handle. Data comes

from a wide range of sources, including sensors, social media, internet transactions, and more, and this massive amount of data requires special technologies and architectures to store, manage, and analyze to extract valuable information.

    - **Velocity**: Extremely fast data generation and transmission, requiring the ability to process data in real-time or near real-time. For example, in the field of financial transactions, a large amount of transaction data is generated every second, which needs to be processed in a timely manner to make decisions, such as risk assessment, transaction execution, etc., otherwise the value of the data may be rapidly reduced.
    - **Variety**: There are many types of data, including not only structured data (such as tabular data in relational databases), but also semi-structured data (such as data in XML and JSON formats) and unstructured data (such as text, images, audio, video, etc.). Different types of data require different processing methods and tools, increasing the complexity of data processing.
    - **Value**: Although big data itself may be massive, high-speed, and diverse, what really matters is to mine valuable information from it. Through data analysis and mining technology, we can discover patterns, trends, and associations hidden in big data, provide support for enterprise decision-making, scientific research, social management, etc., and maximize the value of data.
    - **Veracity:** Big data comes from a wide range of sources and has varying data quality, which can contain errors, noise, duplicates, or incomplete data. Ensuring the authenticity and reliability of data is critical for accurate analysis and decision-making, requiring data cleaning, validation, and quality assessment.
- **b) Explain "variety" in big data**
    - The "diversity" of big data is reflected in the richness of data types. Structured data has predefined formats and schemas, such as tabular data in relational databases, that are easy to store and query, and are suitable for traditional data analysis methods. Semi-structured data has a certain structure, but it is not as strict as structured data, such as data in XML and JSON formats, which are widely used in internet applications and data exchanges. Unstructured data, on the other hand, does not have a predefined format, such as text files, images, audio and video, etc., which is difficult to process with traditional database schemas, and requires the use of specialized technologies such as natural language processing, computer vision, and audio processing to extract information and analyze. This diversity of data types makes big data processing challenging, requiring a combination of technologies and tools to process different types of data to fully exploit its value.
- **c) MongoDB's approach to data diversity**
    - MongoDB handles the diversity of data well. For structured data, documents with fixed fields can be stored and queried like traditional databases (similar to rows in a relational database). For semi-structured data, due to its flexible

document model, it is very convenient to store the data directly in JSON format, without the need to define a strict schema beforehand, and can adapt to semi-structured data with different structures. For unstructured data, MongoDB can store it as binary data (BSON format) in documents, such as binary data that can store images or audio files. In addition, by embedding or referencing relevant data in documents, it can be flexibly organized according to the actual relationship of data, which is convenient for managing and querying diverse data, and can process multiple types of data in one database, without the need to build different storage systems for different types of data, simplifying the data architecture.

6.

- **b) 编写 MongoDB 查询语句**
  - **i) Find books published between 2019 and 2024**

Collapse

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

- **ii) Find books where book_id is not equal to 552020**

Collapse

javascript
copy
```
db.books.find({ book_id: { $ne: '552020' } });
```

- **iii) 查找 D. Sullivan 撰写的书籍或 ISBN 等于 9780134023212 的书籍**

Collapse

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

- **iv) Look for books with an ISBN equal to 9876543210 or 0123456789**

Collapse

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

- **v) Look for books with titles that contain the "SQL" string**

Collapse

javascript
copy
```
db.books.find({ title: { $regex: 'SQL' } });
```

- **vi) 查找 Addison - Wesley 出版的书籍数量**

Collapse

javascript
copy
```
db.books.countDocuments({ publisher: 'Addison - Wesley' });
```

- **vii) Find books published in 2019 with titles containing the string "Mortals" and sort them alphabetically by title**

Collapse

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

- **viii) Add a field named "subject" with a value of "computing" to a book published in 2019**

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