Data Modeling in Relational and Document-based Databases

Objectives

After completing this reading, you will be able to:

- Compare modeling the same data in relational and document databases.
- Describe the complexities of querying data.
- Describe how data duplication affects data modeling
- Describe the considerations associated with evolving the schema.

Introduction

First, compare the information organization in a relational database to a NoSQL document database.

A relational database organizes data into tables with predefined schemas and relationships among tables. In a relational model, you avoid duplicating any data so that the data is stored once and updated in one location. Wherever and whenever the data is needed, the data is referenced. That 's the relational aspect of the data.

On the other hand, an excellent document-based design starts with using that data.

Let's use the example of library books and data organization.

Example: A library book catalog

Library book catalogs store information about books and their authors.

Using a relational database for a library books catalog

If you work with library book data using a relational database, you will create the Books, Authors, Genres and BookGenres tables:

Books

BookID	Title	AuthorID	ISBN	Published Year
1	The Great Gatsby	1	978-0743273565	1925
2	To Kill a Mockingbird	2	978-0061120084	1960
3	1984	3	978-0451524935	1949
4	Pride and Prejudice	4	978-0141439518	1813
5	The Hobbit	5	978-0618260300	1937

Authors

AuthorID	Name
1	F. Scott Fitzgerald
2	Harper Lee
3	George Orwell
4	Jane Austen
5	J.R.R. Tolkien

Genres

GenreID	Genre Name
1	Fiction
2	Classic
3	Dystopian
4	Romance
5	Fantasy

BookGenres

BookID	GenreID
1	1
1	2

BookID	GenreID
2	1
3	1
3	3
4	1
4	4
5	1
5	5

Using a cocument database for a library books catalog

In contrast, a document database is simplified and displayed as a single document with all the required information.

```
{“_id”: 1,”title”: “The Great Gatsby”,”author”: “F. Scott Fitzgerald,””isbn”: “978-0743273565”,”published_year”: 1925,”genres”: [“Fiction”, “Classic”]}
```

Next, compare the complexities of querying the data in relational and document databases.

Complexities of querying data

Let's examine the complexities of querying the same book data using a relational database compared to a NoSQL database.

Querying book data using a relational database

Now, query the data. You will need all the book details at once. You will structure your SQL query by performing a JOIN among those tables. Here's your SQL query example:

```
SELECTBooks.Title AS BookTitle,Authors.AuthorName AS Author,GROUP_CONCAT(Genres.GenreName) AS GenresFROMBooksJOINAuthors ON Books.AuthorID = Authors.AuthorIDJOINBookGenres ON Books.BookID = BookGenres.BookIDJOINGenres ON BookGenres.GenreID = Genres.GenreIDGROUP BYBooks.Title, Authors.AuthorName;
```

The number of joins, use of the GROUP_CONCAT function, and the GROUP BY function make this query complex. Here's how the query breaks down.

- The SELECT command obtains the book title, author name, and a concatenated list of genres for each book.
- The JOIN command joins the Books table with the Authors table on the AuthorID field to get the author’s name.
- The JOIN command joins the Books table with the BookGenres table on the BookID field to associate books with genres.
- The JOIN command joins the BookGenres table with the Genres table on the GenreID field to get genre names.
- The query uses the GROUP_CONCAT function (the exact function may vary depending on your SQL database system; GROUP_CONCAT is used in MySQL and STRING_AGG in PostgreSQL) to concatenate multiple genre names into a single string.
- The query uses GROUP BY to display the results by book title and author name.

Here is an example output:

BookTitle	Author	Genres
1984	George Orwell	Fiction, Dystopian
Pride and Prejudice	Jane Austen	Fiction, Romance
The Great Gatsby	F. Scott Fitzgerald	Fiction, Classic
The Hobbit	J.R.R. Tolkien	Fiction, Fantasy
To Kill a Mockingbird	Harper Lee	Fiction

Querying book data using a NoSQL document database

You already know the output you need and that the output is a first-order criterion in a document database. Remember that your book document looks like this:

```
{“_id”: 1,”title”: “The Great Gatsby”,”author”: “F. Scott Fitzgerald,””isbn”: “978-0743273565”,”published_year”: 1925,”genres”: [“Fiction”, “Classic”]}
```

When creating a document database query, using MongoDB, for example, you specify empty brackets as the first argument denoting you want to see all documents. The second argument, called projection, allows you to choose which fields to present to the client information. You won 't always need to show all of the fields all of the time. And you'll notice the simplicity of this query, as it didn 't need any joins with other collections.

Here is your query:

```
Db.books.find({}, { title:1, author: 1, genres: 1 })
```

The data you need to fulfill the request is already in the document and does not require any joins with other collections.

Important! What about data duplication?

Data duplication is a common practice and is known as "denormalization." Data duplication can improve read performance by avoiding complex joins and queries. However, you will face challenges with data consistency and increased storage requirements.
For example, imagine that the author, J.R.R. Tolkien, wants to be known by his full name, John Ronald Reuel Tolkien, and you need to make this change in a relational database and a document database. How would you implement this change in these two different types of databases?

Relational database	**Document database**
This request requires only one change in the Authors table in a relational database.	Since the author has written 12 books in a document database, you must update this information in 12 documents, which is a small price to pay, as this is a rare event.

Next, explore the considerations associated with evolving a schema.

Schema Evolution Considerations

Changing the schema in a database can be necessary to accommodate evolving application requirements. However, a significant difference exists in how a relational and document database enables schema changes.

In relational databases, changing the schema typically involves modifying existing tables, adding new tables, or altering relationships between tables. Here 's an example:

```
ALTER TABLE table_nameADD column_name data_type;
```

After making schema changes, you might need to migrate existing data to match the new schema, which can involve data transformation and migration scripts.

In document databases like MongoDB, the schema is typically more flexible, and you can often add or remove fields to documents (individual documents, as there is generally no collection-wide enforced schema) without a predefined structure. Here 's an example:

```
Db.books.update ({ _id: 1 },{$set: {Newfield: “Some value”}});
```

Summary

In this reading, you learned that:

- A relational database organizes data into tables with predefined schemas and relationships among tables.
- A good document-based design starts with the usage of data.
- The number of SELECT, JOIN, GROUP, and additional commands contribute to the complexity of building a relational database query.
- For document database queries, you won 't always need to show all of the fields all of the time, and you'll notice the simplicity of these queries, as the queries might not need joins with other collections.
- In relational databases, changing the schema typically involves modifying existing tables, adding new tables, or altering relationships between tables.
- In document databases, such as MongoDB, the schema is typically more flexible, and you can often add or remove fields to documents without a predefined structure.

Congratulations! You have completed this reading and are ready for the next topic.

Author(s)

- Muhammad Yahya

Other Contributor(s)

- Patsy R. Kravitz