Nowadays, database is essential for most applications, and this is also true for cloud-native applications. Here, three concepts related to database and database management in cloud-native applications are illustrated as follows. First, event sourcing. Event sourcing is an approach that is used for data storage and management in microservices. It can store the all the status of the database after every change and trace the any past status of database, which helps to maintain and develop databases better in cloud-native applications. Second, traditional database approaches. The traditional database approaches usually store data in a database in the same place. And for traditional relational databases, they usually use structured query languages to process data. In addition, traditional database approaches only display the current database status. Third, MongoDB database management. MongoDB database is a document-oriented non-relational database. This database structure is good at processing a substantial number of data changes and complex data structures. In addition, the MongoDB database has its own cloud service platform, which supports better performance in cloud-native environment.

In cloud-native applications, when using MongoDB for data management, event sourcing and traditional database approaches respectively show strengths and weaknesses. For event sourcing, event sourcing has the advantage of supporting tracing every status of MongoDB database for different microservices and there is no requirement of time consistency for the statuses that different microservices trace to, which provides support for the independent development and maintenance of different microservices [1]. When in use of MongoDB with event sourcing, the expansion of different microservices will be easier, and when a fault occurs, MongoDB can cooperate with event sourcing to handle the fault separately without affecting other microservice parts. In addition, event sourcing supports quick search of data in complex document structures, which increases MongoDB's efficiency in searching using indexes. However, there are weaknesses for event sourcing. Because event sourcing supports tracing back to every status of the database part corresponding to every microservice, it requires much more space for MongoDB to store the database's status and leads to a more complex structure of MongoDB database. For traditional database approaches, they have the advantage of simple database structure and occupy less storage space compared to event sourcing because traditional database apps only store the current database status. However, there are also weaknesses in traditional database approaches. Because traditional database approaches do not support tracing back to past database status, database maintenance and data analysis according to past data in the past database becomes more difficult when in use of MongoDB database. In addition, it is difficult for traditional database approaches to support the independent development of different parts of microservices and therefore the extension of different microservices using MongoDB databases becomes a harder thing compared with the event sourcing approach.

For different factors such as data volume, they can influence different approaches on different application performance. Here, several factors are considered to analyze application performance difference when using event souring and using traditional database approaches. For data volume factor, because event sourcing stores every previous status of the database, event sourcing occupies more memory space compared to traditional database approaches, which may lead to a decreasing application performance when too much memory space is occupied. However, for traditional database approaches, they only store the current database status, so high data volume has less impact on application performance. For data structure factor, event sourcing supports complex data structures and has good application performance when facing complex data structures [2]. However, for traditional database approaches, complex data structures may bring excessive burden on database operation, and therefore reduce application performance. For data access pattern factor, traditional database approaches mainly use query statements to search data in database. Because the data query system is very mature, it is simple and convenient to use query statements for data query. However, frequent use of query statements can lead to waste of system resources, and it is also difficult to use traditional database approaches to search data in complex database structure. In addition, traditional database approaches can only access data from current databases and cannot access data in databases in the past status. Speaking of event sourcing, it allows access data from any status of databases by accessing

operational events related to data and therefore allows to find a wider range of data. However, this data access pattern is more complex than traditional database apps, and in some cases, it takes longer time to search for data than traditional database approaches.

Besides, different data management techniques can influence cloud-native application performance. Their impacts are illustrated as follows. For data caching, because for the most of time cloud-native applications search data in caching and data caching is closely related to cloud-native application performance. When the lack of cache space occurs, application may lead to decreasing performance because of excessive burden for database. For data partitioning, it helps to divide database reasonably through horizontal partitioning and vertical partitioning and makes database easier to manage data. With more reasonable data partitioning, there will be better application performance. For data replication, it can enable microservices to independently have complete data of the cloud-native application and there is no need for microservices to get data from a central database, thus improving the scalability of microservices and the application performance.

To find out the best data management approach with use of MongoDB in cloud-native applications from event sourcing and traditional database approaches, a series of aspects for data management are analyzed as follows. First, performance. Traditional database approaches perform better than event sourcing when it comes to simple operations such as data reading and data modification. However, event sourcing has advantages over traditional database approaches in processing multiple data or data in complex database structures. Second, scalability. Event sourcing allows microservices in cloud native applications to remain loosely coupled, which helps microservices to develop independently and increase applications' scalability [3]. However, traditional database approaches lack the flexibility to support microservices to develop independently. Third, reliability. Because event sourcing can trace back to every status of the database, it can help databases recover its data when databases encounter any errors or lose data. Traditional database approaches can only obtain the data from the current database. Therefore, event sourcing has better reliability compared with traditional database approaches. Fourth, maintainability. Event sourcing can isolate the detected problem from the database and the maintenance does not affect the other part of databases. However, for traditional database approaches, to deal with the detected problem in databases, sometimes problem isolation is not available, thus bringing inconvenience to maintenance. In short, event sourcing is more recommended compared with traditional database approaches.

In cloud-native environment, speaking of the best practices for data management, here are several suggestions given as follows. First, establish a detailed process for each stage of the data. In a cloud native environment, there is a large amount of data in the database. Therefore, it is necessary to establish a detailed process for each stage of the data such where it will be stored, when it will be backed up and will be cleaned, so that we can manage the data successfully and avoid database overload. Second, set up security protection for the data. Because there is much data in cloud-native environment, it is essential to ensure data security and avoid data breach. Whether using event sourcing or traditional database approaches, it is very important to set up a firewall or carry out other security measures for the database.

In conclusion, I strongly recommend using event sourcing in databases of cloud-native applications because it is excellent in application performance and prominent in other aspects such as scalability and maintainability.

Reference: [1]. Lee, G., Jeong, J., Won, J., Cho, C., You, S. J., Ham, S., & Kang, H. (2014). Query performance of the IFC model server using an object-relational database approach and a traditional relational database approach. Journal of Computing in Civil Engineering, 28(2), 210-222.

[2]. Alongi, F., Bersani, M. M., Ghielmetti, N., Mirandola, R., & Tamburri, D. A. (2022). Event-sourced, observable software architectures: An experience report. Software: Practice and Experience, 52(10), 2127-2151.

[3]. Pacheco, V. F. (2018). Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices. Packt Publishing Ltd.