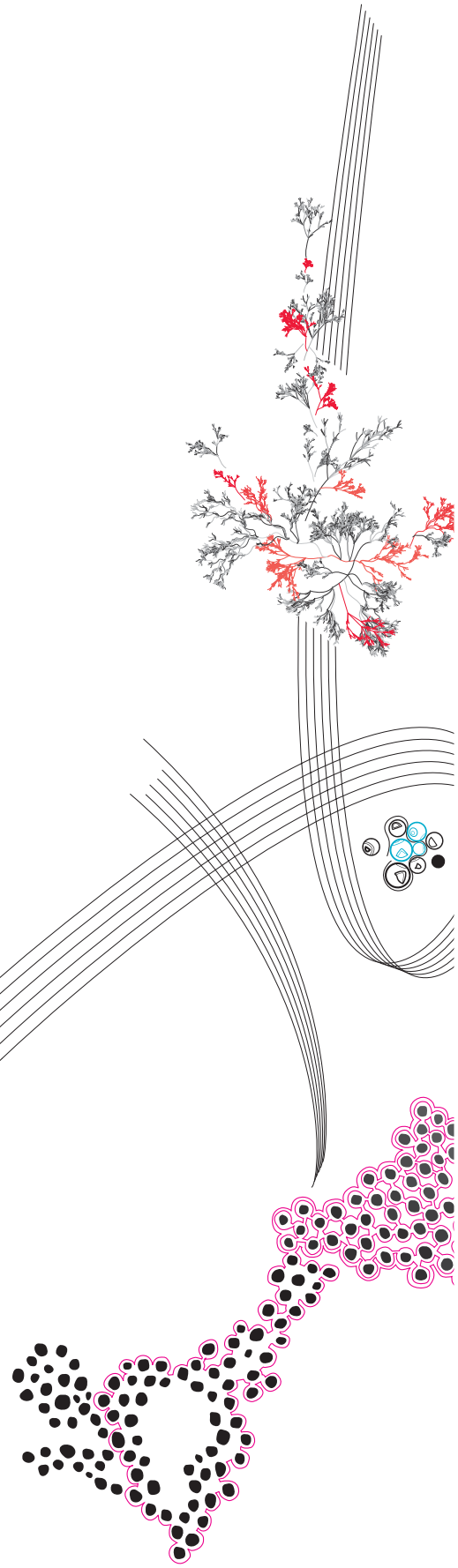


Thesis Computer Science

MBench: a benchmark suite designed for database schema migration

Shuhao Li

Supervisor:
dr.ir. Maurice van Keulen
dr. Faizan Ahmed
ir. Jorryt-Jan Dijkstra

April, 2022

Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

When I was looking for my graduation project, I was at the most lost time since I came to this country.

This confusion is caused by a variety of reasons. On the one hand, I am a data science student but gradually lost interest in machine learning and deep learning. On the other hand, the autumn recruitment fair in my home country is approaching, but I still have not decided on the future technical direction. In addition, the new crown epidemic is also one of the reasons for this situation. Forced to stay at home, I started to think all day but it was difficult to practice.

In fact, the graduation project I originally wanted to choose was not the one I have now completed. When I first contacted Jorryt Dijkstra, he told me that the project had already been chosen by a student but that maybe I could design a benchmarking tool for the field that the project applies to. This piqued my interest, I learned a lot about software engineering during my undergraduate studies, but they haven't been used in practice for a long time.

However, things didn't start well. At first I was at a loss as to what I was trying to accomplish. Thanks to Dijkstra for giving me tireless guidance in the beginning stages and getting me started to get a general idea of the problem. I also thank him for some of the questions he raised, each of which is critical now that I think about it.

Another person I would like to thank is my supervisor, Maurice van Keulen. Whenever I ask a question, I always get a very serious and detailed answer from him. I am also very grateful for his tolerance, understanding and patience with me, and I am still sorry for not being able to attend a meeting on time because I overslept one day.

In addition, I would like to thank some people I have never met. Their research in the field of schema transfer or benchmarking helped me a lot and enabled me to successfully complete this master's thesis.

Finally, I would also like to thank my classmates at the University of Twente and the residents of the city of Enschede. I'm used to walking on the road thinking, and my face is heavy when I'm lost in thought. At this time, passers-by often think that I am unhappy and tell me to cheer up and thank them for making me feel the kindness of the world. There are also many more people I want to thank, but forgive me for not being able to list them all here.

Now, the time to leave the campus is approaching, and I am about to face a completely different life. During the completion of this project, I also participated in the autumn recruitment in my home country and got three offers. I hope I can make a choice that I have no regrets.

MBench: a benchmark suite designed for schema migration

Shuhao Li*

April, 2022

Abstract

How to provide users with continuous network software services is the focus of current computer field. As one of the keys to solving this problem, many researchers have begun to explore how to transfer the database schema online and many tools have been designed for this purpose. However, there is currently no tool for benchmarking schema migration. Thus, a benchmarking tool designed for schema migration is implemented. The SQL quries of this test tool is varied compared to traditional benchmarks and benchmarking tools.

Benchmarking software or systems is necessary because it gives developers a better grasp of how the software are performing and gives users greater confidence. However, existing benchmarks provided by other researchers do not meet the needs of schema migration tool developers and testers, such as including schema migrations that are not comprehensive enough and do not implement the mixed state. Now there is a new benchmarking tool not only meets the needs of these persons, but also helps organizations that want to migrate schemas better plan their migrations.

The software was developed with modifications and additions of new features to another benchmark suite. In the process of choosing which software should be based on, a scoring mechanism is designed according to the requirements. Relying on this scoring mechanism, multiple softwares are compared comprehensively and carefully. Finally, OLTPBenchmark is selected. Besides, this master thesis also summarized the requirements for schema migration benchmarks made by previous researchers, and designed and implemented new benchmarks based on these requirements.

After further identifying the requirements and implementing the software, some experiments were performed to test the performance of the tool. Experimental results show that this tool can maintain good performance when running for a long time and meets our requirements for schema migration benchmarking tools. Subsequently, some experiments were conducted to test the tool in practical use. The results of these experiments also provide insight into some of the schema migration tools.

In the discussion section, we further analyze the performance of this tool. Compared to traditional benchmarking tools, its queries are variable during the workload execution phase. Compared with schema migration tools used by researchers in other schema migration fields, it is more extensible and provides multiple bench cases. Users can easily implement their own benchmarks according to their needs. In addition, the tool is currently the only benchmarking tool that implements the mixed state.

Keywords: benchmark, benchmark suite, databases, schema migration

*Email: s.li-6@student.utwente.nl

Contents

1 Introduction

1.1 Motivation

With the advent of the information age, computers have played an increasingly important role in people's lives. A variety of computer systems provide people with a variety of services, including work, entertainment and travel. These systems are not immutable since development, and developers will update them according to user needs. For example, an online shopping website may initially only save key information such as product prices and descriptions in their databases. But with the development of the business, in order to provide customers with better services, the company may keep some other information like the lowest price recently. At this time, it is necessary to change the schema of the databases.
Migrating the schema of the database will cause the database service to be temporarily inaccessible. How to reduce the impact of temporary inaccessibility of databases caused by database schema migration is a topic that some researchers have focused on in the past six decades. In the past, schema migration often required temporarily shutting down the service and waiting for the system upgrade to complete before allowing customers to access it. However, this method has shortcomings. For commercial companies, every minute of downtime means a loss of profit. For the government, it may cause inconvenience in public services. Based on this background, online computer system developers began to find various ways to reduce the time that the system was offline, and a variety of zero-downtime deployment technologies were gradually developed,such as PRISM[26] and Facebook's OSC[45] . These solutions solve the problem that the system must be shut down for a long time to upgrade when the database structure changes. In recent years, with the rise of cloud services, more and more companies have chosen to migrate their services to cloud servers. Amazon, as one of the largest cloud service providers, also provides DMS(Data migration services) to facilitate the migration of customers.
However, there exist no benchmarking tools and benchmarks for this area. Classic database testing tools, such as Jmeter and Hammer DB, tend to pay more attention to the load capacity of the database, and use indicators such as throughput, response time and error ratio to measure database performance. These tools measure the performance of the same database on different storage devices. However, when the database's schema is changed, the situation is different. While paying attention to the above indicators, it should also pay attention to the differences of above indicators before and after schema changes of the database, that is, the impact of database structural changes on the load capacity, the latency and other indicators of the database. Existing database testing tools lack attention to this aspect. Moreover, when database managers want to test the impact of these changes, they usually need to do them in multiple steps: first let the database run normally for a period of time, then use tools such as Liquibase to define the changes, and wait for the changes to complete before passing the database test tool observe the performance indicators during the migration process, which caused a certain degree of inconvenience. In addition, the existing database testing tools also have other problems. For example, some tools have weak visual analysis capabilities after the test is completed, the code and programming style are old and the extensibility is insufficient, etc.

Same as the test tool, there are seldom benchmarks for database migration. Existing well-known benchmarks such as TPC-C and TPC-H all define a system, and then simulate the operation of this system to perform tests on the database. For example, the model used by TPC-C is a large-scale commodity wholesale sales company, which has several

commodity warehouses distributed in different regions. The benchmark simulates the company's business system, and when the business expands, the company can add new warehouses. However, none of these benchmarks takes into account the situation of schema changes. Taking TPC-C as an example, the development of the company's business is not only an increase or decrease in the number of warehouses, but also a change in the database structure. For a large shopping company like Amazon, the schema of the shopping-related database will have many changes compared to when it was first established. With the development of the company's business, some new necessary columns are added, and some redundant columns are deleted. This is something that sometimes happens in the real development process. Therefore, we can see that it is very necessary and meaningful to test the database's ability to respond to changes. But according to the author's knowledge, there is no benchmark to do this.
Based on the above reasons, we believe that it is very important and realistic to develop a new database testing tool for database schema changes and design a new benchmark. Because database structure changes are exactly what will happen in the displayed business system. But now there is few benchmark and test tool that can simulate and test schema migration.

1.2 Research Question

The main purpose of this research is to design a benchmarking tool for schema migration. Generally, developers can develop a tool in the following two ways. The first way is to build the tool from scratch, and the other way is to develop it based on existing tools. Since the second approach can save development time by avoiding duplication of effort, this study will further modify existing tools and benchmarks rather than starting from scratch.
It is different from other benchmarking tools that focus on indicators such as database capacity and latency. The benchmarking tool proposed in this thesis also pay attention to the performance comparison before and after the schema change, the performance behavior before and after schema changes and the performance of schema changes itself. Therefore, some new indicators in this regard will be proposed to measure the performance of the database.

In this master thesis, in order to better create a benchmark tool for zero-downtime structure migration and evaluate it, the following research questions are proposed:

1. Which of the existing benchmarking tools can be more easily re-developed to adapt to the scenario of schema change?
    - What criteria should be met when selecting tools for secondary development?
    - How should the tool be re-developed to make it meet the demand?

    SW
2. In view of the database schema changes, what requirements must a benchmark and benchmarking tool meet?
    - What requirements can we get from the existing benchmarks and benchmarking tools?
    - What new requirements can be found in case of schema migration?
3. In case of schema migrations, what metric will allow us to better measure the performance of the database?

4. How the quality of a benchmark can be measured?

1.3 Methodology

This section will introduce the methodology used in this thesis. The main purpose of this thesis is to design and implement a benchmarking tool for schema migration. Because this thesis needs to find a solution to the problem raised, the main problem of this thesis is a design problem. For some research questions, because they ask for some knowledge and some requirements are found in some literature. These questions are knowledge questions. Therefore, the design science proposed by Wieringa to solve design problems and knowledge problems will be used as the methodology of this thesis[64] . For design problem, Wieringa introduced a method of how to design and implement systems or software and introduced how to investigate the performance of the finished product. The methodology is divided into five tasks, which are: problem investigation, implementation evaluation, treatment design, treatment validation and treatment implementation. Figure 1 shows the design science in the form of diagrams and these four steps show the life cycle of software design and implementation.

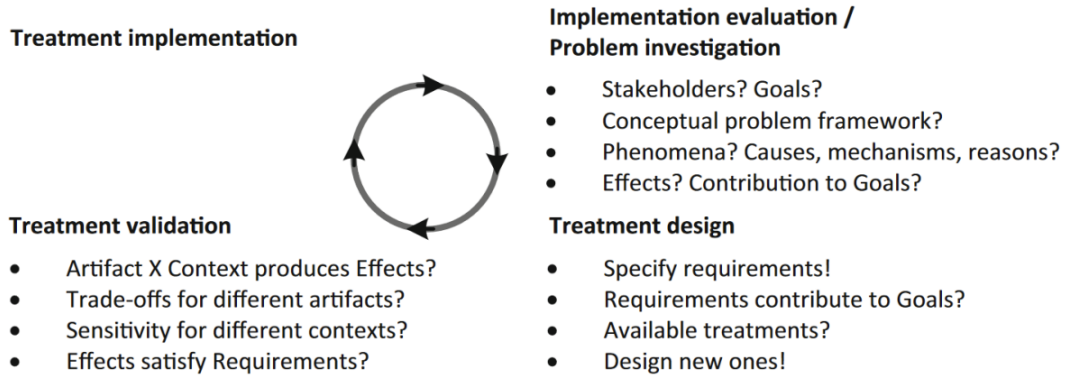


FIGURE 1: Design science cycle in the book

Because the main goal of this thesis is to design a tool for benchmarking zero downtime scheme migration. This research is a solution-oriented research. For this kind to research, the treatment design, the treatment validation and treatment implementation steps are applied. The main task of the treatment design step is to design the artifacts used to solve the problem. For this step, since it asks knowledge of the world, so we can use the empirical cycle used to solve knowledge questions to obtain the requirements. The empirical cycle is showed in figure 2. In this thesis, because it is only used to solve some of the subproblems like research question 2.1 and 3, only three steps named Research problem analysis, Research and inference design and Data analysis in the circle will be used. This thesis firstly judges whether the problem is a knowledge problem. It then summarizes researches in related fields and analyzes the collected data and information and give the answer to the question after the analysis. The main task of the treatment validation step is to evaluate whether the designed software can solve the problem. In this thesis, we will also apply the tool to compare the performance of different databases and migration tools in the scenario of schema migration and assess whether the tool meets the requirements.

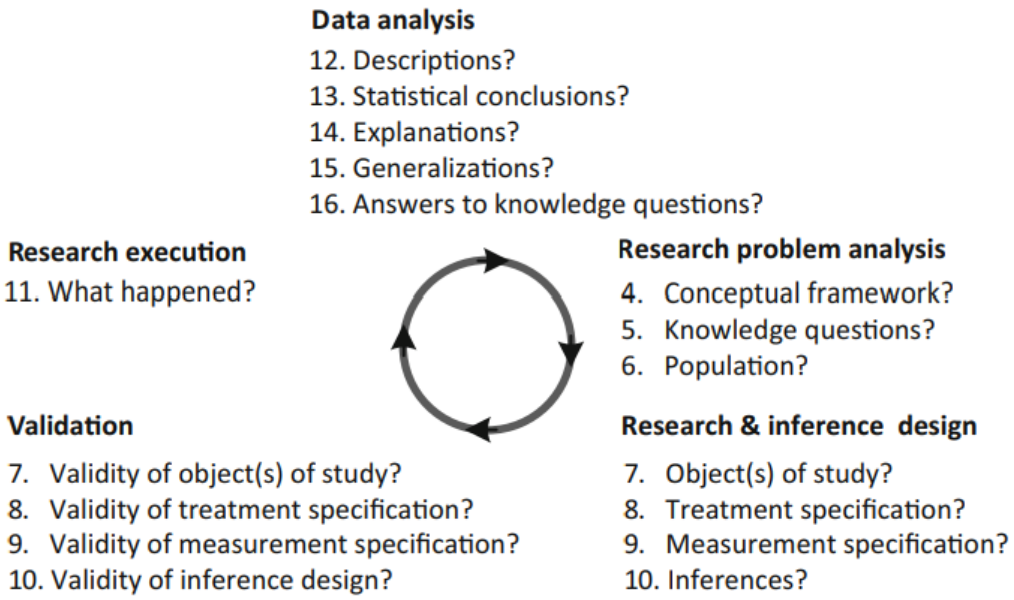


FIGURE 2: Design science cycle in the book

1.4 Organization

This chapter mainly introduces the organization of this thesis. The motivation and research questions of the research are given in the introduction section. In the background chapters that follow, some knowledge related to databases and database testing is provided. This knowledge can help readers better understand the next content of the thesis. The third section is a literature review. This chapter first summarizes the researches in the field of schema migration, and then introduces some benchmark-related researches. After that, the design, implementation, and evaluation of the treatment are sequentially introduced according to the steps of design science. In Chapter 6, the experimental results in the evaluation section are further analyzed and future work is also listed. Finally, the main contributions and research questions of this thesis are summarized and answered in the final chapter.

2 Background

2.1 Databases

In the field of computer science, a database is a place where data is digitally stored on a computer in a certain order[8] . Before the advent of databases, data was mainly stored on computers in the form of simple files. Using this method to store and read data is logically simple, but it cannot shield the complexity of data access. In this context, the database was born, which solved a series of problems such as data integrity, consistency and security.
In 1960, Bachman designed an integrated database system, which maybe also the first database management system[6] . By the mid-1960s, with the development of computers, the speed and flexibility of computers began to spread and many general database systems could be used. Many customers requested the development of a standardized language for public services, the languages that was not developed for public service at the time have became a nuisance for many customers. Bachman formed a database task group to solve this problem. The database task group proposed this standard in 1971, also known as the "CODASYL method"[24] . In 1970, out of dissatisfaction with the CODASYL method and the lack of search engines in the IMS model, Codd wrote a series of papers outlining novel methods for constructing databases[21, 22, 23] . His idea eventually evolved into a paper called "Data Relation Model for Large Shared Databases", which described a new method for storing data and processing large databases, which used tables with a fixed length to store data. In 1974, Stonebraker and Wong were very interested in Codd's research and made a decision to study relational database systems. They successfully proved that the relational model may be an effective and practical method for storing and processing structured data[59] .

2.2 Relational Databases

Relational databases were proposed in the 1960s and became dominant in the 1980s[24] , and it is still a popular database to this day. Relational databases organize data as a set of tables with columns and rows. In a relational database, each row in the table has a unique ID. The columns in the table store the attributes of the data. Each row of records often has multiple attributes, and each attribute has its own value. Relational database provides a convenient and flexible way to access structured data.
The relational database is based on the relational model. In this model, logical data structures such as data tables and views, and indexes are separated from the physical storage structure of the data. In addition to the data is divided into logical and physical, operations are also divided into logical operations and physical operations. Logical operations specify what the application needs and physical operations determine how to access data[13] .
The relational model provides a standard way of querying and loading data for all applications, which has changed the inconvenience caused by each application using its own unique data storage structure in the past. In the relational model, different data entities types are stored in different tables, which is an intuitive, efficient and flexible way of storage.

2.3 Database Manage System(DBMS)

As a storage location for data, a database usually requires a program to comprehensively manage it, and this program is called a database management system (DBMS). DBMS is the interface between the database and the user, allowing users to retrieve, update and

manage the organization and optimization of information. The DBMS also helps to monitor and control the database to support various management operations such as performance monitoring, tuning, backup and recovery.
When used, people often refer to a database management system simply as a "database". However, the two terms do not have the same meaning. A database can be any collection of data, not just a collection of data stored on a computer, and a DBMS is software that allows you to interact with the database.
All DBMSs have an underlying model that is used to structure how data is stored and accessed. A relational database management system is a DBMS that adopts the relational data model. In this model, data is organized into tables. In the RDBMS environment, it is more formally called a relationship. In this thesis, the relational database management system is also the DBMS we will mainly discuss.

2.4 Structured Query Language

Structured Query Language, abbreviated as SQL, is a language used to query and update data and interact with the database. Users can complete functions such as querying, modifying and deleting data through SQL and can also control access rights to the database through SQL. Standard SQL is managed by the ANSI Standards Committee and is therefore called ANSI SQL. Each DBMS also has its own implementation, such as PL/SQL, Transact-SQL. SQL is divided into four categories, including data definition language (DDL), data manipulation language (DML), transaction control language (TCL) and data control language (DCL).

2.4.1 DDL

Data Definition Language (DDL) is the language responsible for data structure definition and database object definition in the SQL language set. The main function of DDL is to define database objects. The core commands of DDL are CREATE, ALTER, and DROP. DDL enables users to create or delete tables, add and delete indexes, specify links between tables and impose constraints between tables. Usually, using DDL to modify the database is a very expensive operation because it will lead to the migration of the database schema and further affect the DML language for data access.

2.4.2 DML

Data manipulation language (DML) is a programming statement used for database operations to run access to objects and data in the database. The main function of DML is to access data, so its syntax is mainly to read and write databases. The core commands of DML are INSERT, UPDATE, DELETE, and SELECT. These four commands are collectively referred to as CRUD (Create, Read, Update, Delete), that is, add, read, modify, and delete. Back-end developers often use the DML language to interact with the server and read the data they want from the database. At the same time, DML commands are often used with keywords such as WHERE and JOIN to achieve the purpose of reading the desired data from the database.

2.4.3 TCL

Transaction Control Language (TCL) is used to manage transactions in the database. These are used to manage changes made by DML statements. Transactions are an important part of maintaining database consistency. All DML statements in the same transaction

are either executed or not executed at all. It also allows grouping of statements into logical transactions. The core commands of TCL are COMMIT and ROLLBACK. The first keyword is used to commit the transaction, and the second keyword is used to roll back the transaction.

2.4.4 DCL

Data Control Language (DCL) is an instruction that can control data access rights. In actual project development, it is very important to have access to different personnel according to requirements. DCL can control a specific user account's control over database objects such as data tables, view tables, stored procedures and user-defined functions. The core commands of DCL are GRANT and REVOKE. DCL mainly controls the user's access rights, so its instruction is not complicated. The rights that can be controlled by DCL are: CONNECT, SELECT, INSERT, UPDATE, DELETE, EXECUTE, USAGE, REFERENCES. According to different DBMS and different security entities, the permission control supported is also different. For example, some databases such as SQLite do not support DCL, but control the access rights of the database through the file system.

2.5 Database Transactions

Database transaction refers to a series of operations performed as a single logical unit of work, either completely executed or not executed at all. A logical unit of work need to guarantee four attributes called ACID (atomicity, consistency, isolation, and durability) attributes, only in this way can it become a transaction. In order to ensure the correct execution of transactions, different database management systems have different implementations.

2.5.1 Transaction properties

The guarantees provided by the transaction is often described as ACID. That is, atomicity, consistency, isolation level, and durability.
In the ACID attributes of transaction processing, consistency is the basic attribute and the other three attributes exist to ensure consistency. Consistency refers to the transition of the system from one state that satisfies a predetermined constraints to another state that satisfies the constraints. These constraints include uniqueness constraint, integrity constraints and the correctness guaranteed by the isolation level, etc. Usually, when a transaction fails, the database will roll back the transaction as a whole to ensure that the data is always consistent.
Atomicity, simply means "all or nothing", all operations in a transaction are either done or not done. So when an error occurs in the middle of the transaction execution, the database needs to discard the previously executed write and roll back to the state before the transaction is executed. The main purpose of this property is to prevent the failure of failed transaction operations from affecting the data in the database. A classic example is bank transfer. If the transfer is successful, then the account of customer A will be reduced by 100 euros, and the account of customer B will be increased by 100 euros. Databases usually use journaling to guarantee atomicity. Modifications to the data will be written in the journal first, and when the transaction is committed, a commit record will be generated. When a crash occurs, if the commit record exists, it means that the transaction has been completed. Thus the database should recover the data according to the records in the journal. Otherwise, the transaction should be skipped.

However, atomicity does not completely guarantee consistency. In the case of multiple transactions in parallel, even if the atomicity of each transaction is guaranteed, it may still result in inconsistent data. For example, transaction 1 needs to transfer 100 euro to account A: first read the value of account A, and then add 100 to this value. However, between these two operations, another transaction 2 modified the value of account A, adding 100 euro to it. Then the final result should be that A has increased by 200 euro. But in fact, after transaction 1 was finally completed, account A only increased by 100 euro, because the modification result of transaction 2 was overwritten by transaction. In order to ensure consistency in the case of concurrent, isolation is introduced, that is, to ensure that the data that each transaction can see is always consistent, just as if other concurrent transactions do not exist. In terminology, the state after multiple transactions are executed concurrently is equivalent to the state after their serial execution. The database uses a variety of mechanisms to ensure isolation.
Durability ensures that after the transaction is successfully committed, the change operation of the data will be persisted to the disk, and there will be no data loss due to failure. In a computer system, data is usually written to the memory first, and then written to the hard disk at an appropriate time. Without durability, data that was not written to disk would be lost in the event of a crash. Some database management systems represented by MySQL use logs to guarantee durability. The redo log is divided into two parts: buffer and file. When the transaction is committed, the redo log buffer will be flushed and recorded in the redo log file. In this way, when the database crashes and restarts, it can read the data in the redo log file and restore the database.

2.5.2 Multiversion concurrency control

Modern database management systems allow read and write operations to be executed concurrently. However, the concurrency of multiple transactions may cause dirty reads, non-repeatable reads and phantom reads and affect the consistency of the database. To solve this problem, some databases have their own concurrency control mechanism. Multiversion concurrency control is an advanced and widely used concurrency control method[7] . In short, it assigns a unidirectionally increasing timestamp to a transaction and keeps a version associated with the transaction timestamp for each modification. Read operations read only the snapshot of the database before the transaction started. In this way, it achieves a lock-free solution to the problem of read-write concurrency conflicts

2.5.3 Lock

Lock is a very important concept in data network transmission. It is is mainly to ensure the integrity and consistency of the database data in the case of multiple users. In a multi-user environment, because multiple transactions may read and update a record at the same time, data inconsistency may occur. Locking a table, row, or other object in the database will ensure that only the transaction holding the lock can operate on the object at a time. Other transactions that want to read or modify the object will be blocked and wait for the lock to be lifted. Locks make the execution order of concurrent transactions linear and in this way avoid possible problems with concurrency.
For databases, there are many types of locks. A popular classification method is to divide locks into shared locks and exclusive locks according to whether the locks are allowed to be shared. For shared locks, when the transaction locks the data, other transactions can only read the data and cannot do any modification operations. Only when the shared lock on

the data is released, other transactions can modify it. For exclusive locks, when the lock is not released, read operations are also prohibited compared to shared locks. At the same time, locks can also be divided into database locks, table locks, row locks, etc according to different granularities.
Because the lock will block other operations except the current operation and operations need to detect whether the object currently needs to be accessed has been locked, the use of the lock is a very expensive operation. Some databases have also optimized locks. For example, MySQL proposed and used an intention lock. When a record in the database is locked by a row lock, the table where the record is located will be locked by the intention lock at the same time. This method avoids the disadvantages of needing to scan the full table to detect the row locks when there is a table lock that wants to lock the table to detect the shortcomings of row locks, and reduces the impact of locks on database performance.
The lock mechanism is also an important reason that affects schema migration. For many databases, when using DDL statements for schema migration, the database management system will add table locks for the affected tables and the table lock will block DML statements that want to query or modify the data.

2.6 Schema Migration

A database schema is a definition of the structure of a database[42] . It can be simply understood as a logical definition of database related objects, but also a collection of objects in the database. This collection contains various objects such as: tables, views, stored procedures, indexes, etc. Figure 3 shows the schema of the TPC-C benchmark. Generally, developers and researchers classify databases as traditional relational databases and non-relational databases. They are characterized by the need to define the database and table structures in advance before using them. In this thesis, when talking about schema migration, we are only talking about traditional relational databases like MySQL and Oracle. Non-relational databases such as MongoDB and Redis usually do not have a fixed table schema[48] , so they are out of the scope of this thesis.

In addition, some databases that form a schema when reading data are also not within the scope of this thesis, such as Hive. In traditional relational databases, the schema of tables are forcibly determined when the data is loaded. If it is found that the data does not conform to the schema during loading, it will be refused to load the data. Because the data is checked against the schema when it is written to the database, this design is sometimes referred to as the "schema on write"[38] . Hive takes a different time to form a schema than traditional relational databases. In order to reduce the database loading time, they choose to verify the data during query and set the value that does not meet the query conditions to null. This mode is called "schema on read"[30] . Because this kind of database does not determine the schema of the data when storing it, but only stores the data, and adopts the method of setting the value that does not meet the query conditions to null, it is also out of the scope of this thesis.
Statements that cause database schema changes are commonly referred to as Data Definition languages (DDL), while statements that manipulate the database, such as INSERT and UPDATE, are called Data Manipulation Language (DML). For some databases, DDL can cause table rebuilding and data transfer, resulting in blocking behavior. It is temporarily impossible to query the data in the table being rebuilt, and the queryer needs to wait for the DDL operation to complete before getting the correct result. Michael de Jong run an experiment to compare the blocking behavior of MySQL and PostgreSQL when performing different DDL[28] . In his example, PostgreSQL blocks all queries when adding a column and specifying it as non-nullable. However, this experiment also found

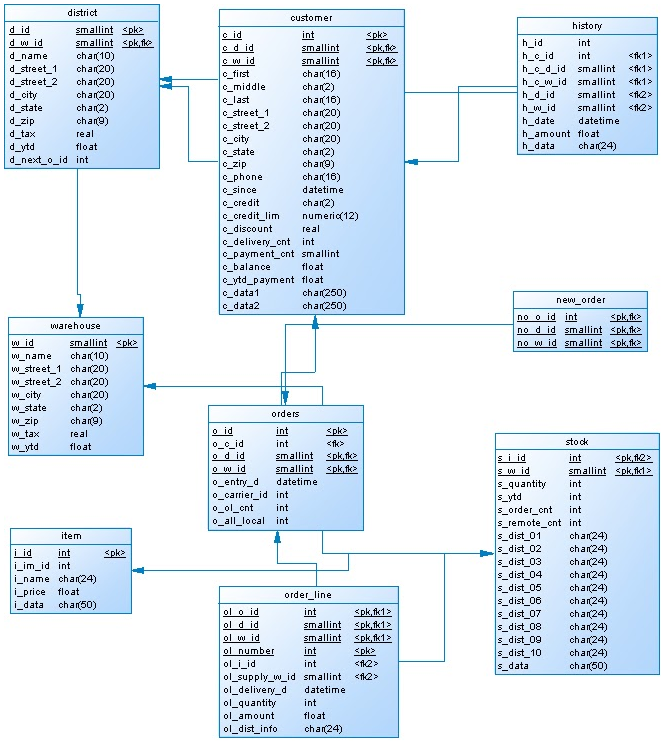


FIGURE 3: Schema of TPC-C benchmark.
found at:http://maheshgadgilsblog.blogspot.com/2011/11/more-on-constrained-tree-schema.html

that MySQL have a different behavior and it did not block the SELECT statement while performing this operation. De Jong did not probe further into the reasons for the difference. In fact, MySQL introduced online DDL in version 5.5. This feature is based on fast index creation feature and allows SELECT statements to be executed during DDL operations. However, this feature is disabled for DML that only works on older tables. So it doesn't meet Michael de Jong's requirement that the tools must support multi-version data. From his experiments, we can conclude that when schema change occurs, databases will block the current DML statement by applying a lock, which makes schema change usually an expensive and complex operation.

2.7 Zero Downtime Schema Migration

MySQL's reference manual reveals why schema migration blocks DML statements. For the ALTER TABLE statement, MySQL will rebuild the table according to the specific content of the statement, and insert the data of the original table into the rebuilt table. In addition, MySQL treats each DDL statement as a transaction. For DDL statements, the database uses table locks to lock the original table and block concurrent DML statements. This is the reason that schema migration causes the performance of the database to drop or even stop the service[1]. The domain that solves this problem is called zero-downtime schema migration.
Zero downtime schema migration refers to changing the database schema under the con-

[1]https://dev.MySQL.com/doc/refman/5.7/en/

dition that the database can continue to provide services. In the past ten years, many researchers have proposed their own solutions to the problem of continuous deployment. we will discuss four approaches that have been proposed.

1. **Approach 1** The first approach was proposed in the book *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*[37] . It copies the database and records operations that affect the database data such as update operations and delete operations during schema evolution. When the database copy is complete, the recorded operations will be replayed until the old and new database until the old and new versions of the database are in sync. And there are two options at this time: switch the database to the new version or continue to record the operation.

2. **Approach 2** This approach involves rewriting the SQL so that it will be executed in the new table as well as in the old table. Researchers have also developed a variety of tools based on this approach. PRISM+ uses ICMOS and SMOS to rewrite the query, so that the rewritten query can be executed on both the old and new data tables[26] .

3. **Approach 3** Some other tools such as Openark kit[47] and QuantumDB[29] use the method of creating a new ghost table to achieve zero downtime schema migration. When the DDL statement is applied, they will create a data table called the ghost table that conforms to the changed schema, and transfer the data from the original table to the ghost table. For data consistency, different tools have different implementations. For example, QuantumDB uses triggers, while Gh-ost uses subscription database logs. Although a part of them also involves the rewriting of SQL, there is a clear difference compared with Approach 2. Finally, when the data migration is completed and the two tables reach the synchronized state, the old table can be deleted.

4. **Approach 4** This approach is called "Blue-Green deployment" and it is mainly used in scenarios when database clusters are used. This method marks the database that has not been changed in blue, and mark the database that has been changed and reached the synchronization state in green. Like show in figure 4, When schema changes are applied, we apply the changes to some database each time and the other database still provides services normally. When the change is complete, the color of the changed database is switched to green, and then some databases that are still blue are selected and changed. Eventually, all databases will be converted to green and the deployment is finished.

2.8 Database Test

Scientists believe that testing is an important part of the software development life cycle[39] . Through sufficient and complete testing, we can reduce errors and find some special cases that were not considered during the development process, and prevent these problems from appearing in the production environment to further avoid huge losses. In this paper[39] , they compared three automated testing tools which are available in the market in some aspect like usability, effectiveness and budget.
Many existing database testing tools usually test the following aspects of the database to measure the performance of the database[58, 25, 36, 54] . First of all, the throughput of the database is a very important metric for benchmarks. The TPC-C benchmark test uses

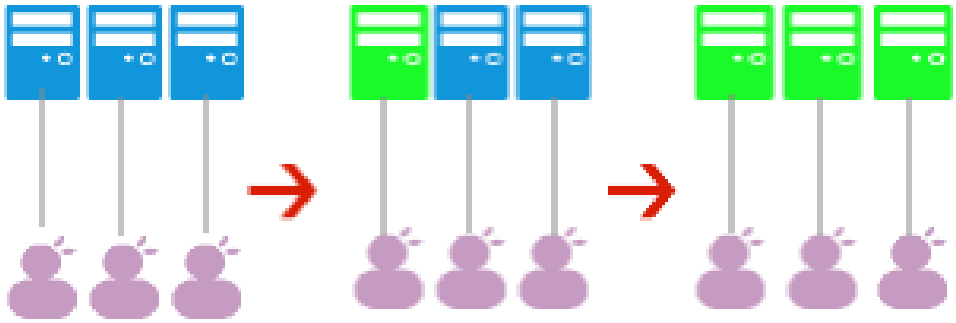


FIGURE 4: Blue-green deployment

tpcm as an indicator to measure the throughput capacity of the database. Tpcm describes how many new order transactions the system can process every minute while performing payment operations, order status query, shipping and inventory status query these four transactions. Latency is another very important indicator of database performance. It refers to the time it takes for the database to process the query. High latency can be caused by various reasons, such as distance, propagation delay and internet connection type. When the network delay is large, even if the database has a large throughput, the user will suffer a high latency due to the network delay. High latency will cause the user to get a response long after the request is initiated, resulting in a poor user experience.
Queries per second(QPS) and Transactions per second(TPS) are other two factors related to the throughput. The average time consumed to execute a sql is also an indicator to measure the database. In addition, the integrity, consistency and correctness of the data is also very important. A good database must ensure that the data in the database will not be easily changed or lost. Finally, some other indicators, such as lock-related indicators and hit rate, can also measure the performance of the database.
In general, the types of database testing can be divided into five types: load test, benchmark test, stress test and stability test.Different test types focus on different aspects of the test object.
Load testing is used to check the system's ability to work correctly when using a large amount of data. And benchmarking can observe the behavior of the system under different pressures, evaluate the capacity of the system, grasp what are the important changes and observe how the system handles different data. The main task of the stress test is to obtain the limit of the correct operation of the system and to check the ability of the system to perform correctly under the instantaneous peak load. Finally for the stability test, it is to run the system for a period of time under the condition of a certain business pressure on the system to detect whether the system is stable.

2.9 Benchmark

Benchmark refers to a standardized test which is composed of an application scenario, a data generator and a set of queries. The data generator is capable for generating the data suitable for the scenario and can adjust the size of the data amount according to the settings. And the queries should be representative for the scenario. In the database bench-

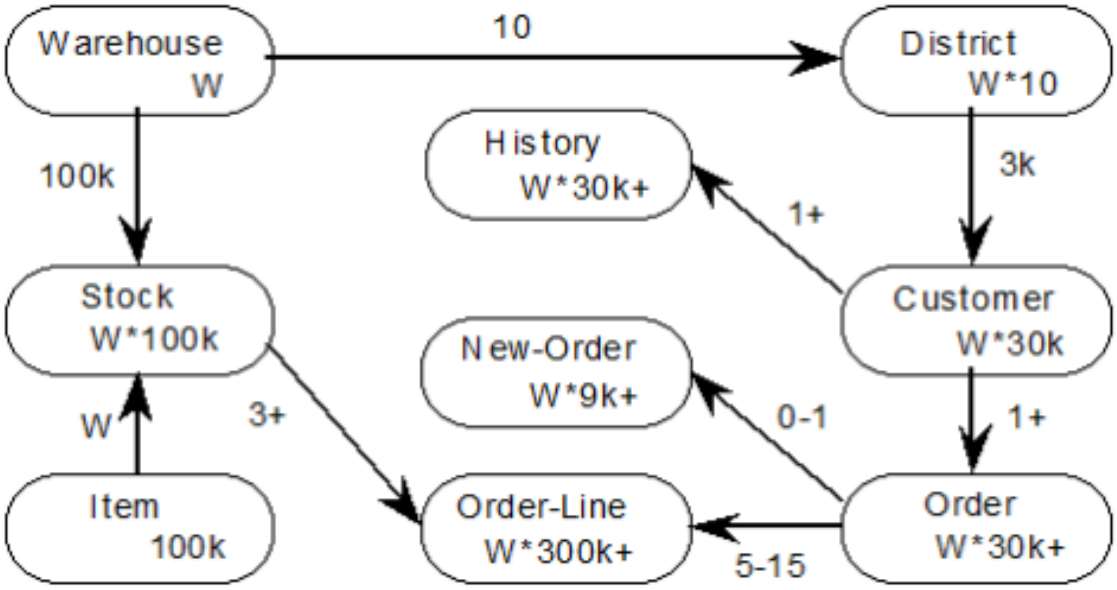


FIGURE 5: TPC-C test process
found at: https://commons.wikimedia.org/wiki/File:Sch%C3%A9ma_Datab%C3%A1ze_metody_TPC-C.png

mark test, in order to test the performance of different databases in different scenarios, researchers have proposed and implemented different benchmarks.
According to different application areas, the existing benchmarks can be roughly divided into three categories: OLTP, OLAP and OLAP+OLTP. OLTP is the main application of traditional relational databases, mainly for basic and daily transaction processing, such as bank transactions. OLAP is the main application of the data warehouse system. It supports complex analysis operations, focuses on decision support, and provides intuitive and easy-to-understand query results. The first type of benchmarks include TPC-C, TPC-E, AS3AP, etc. The second type of benchmarks are represented by SetQuery, TPC-H, and TPC-DS. And the third type of benchmark is CH-benchmark, InMemBench and so on.

2.9.1 TPC-C

The model used in the TPC-C test is a large-scale commodity wholesale sales company that has several commodity warehouses distributed in different regions. When the business expands, the company will add new warehouses. Each warehouse is responsible for supplying 10 points of sale, and each point of sale provides services to 3000 customers. In the orders submitted by each customer, there are on average 10 products in each order. About 1% of the products in all orders are There is no inventory in the warehouse to which it directly belongs and it must be supplied by warehouses in other regions. At the same time, each warehouse must maintain inventory records of 100,000 products sold by the company. Figure **??** shows the TPC-C test process.

TPC-C is mainly divided into five types of transactions, order creation, order payment, order query, order delivery and inventory query. These five transactions occur in a certain proportion. The test finally measures the number of executions of order creation transactions per minute. The unit is tpmC, tpm is the abbreviation of transactions per minute.

2.9.2 TPC-H

The TPC-H benchmark simulates a commercial procurement application. Its database model contains 8 tables, representing the objects or behaviors participating in the procurement and ordering in the commercial field. The size of the data volume has a direct impact on the query speed. The Scale Factor (SF) is used to describe the data volume in TPC-H. 1 SF corresponds to 1 GB unit, and SF is 1, 10, 30, 100, 300, 1 from low to high. 000, 3 000, 10 000. The data volume corresponding to 1SF is only the total data volume of 8 tables, excluding the space occupation such as indexes, and more space needs to be reserved when preparing data. The performance test benchmark defines 22 complex query (SELECT) statements and 2 update data statements (including INSERT and DELETE operations). The size of the database is determined by Scale Facto, and the total data size can range from 1GB to 100TB. The TPC-H benchmark uses the number of queries executed per hour (QphH@size) as the metric.

2.9.3 Other benchmarks

In addition to these two classic benchmarks mentioned above, many other benchmarks have been built for different databases or different system architectures. For example, TATP is a benchmark used to test the performance of an in-memory database[65] . It simulates the system of a telecommunication providers and it consists of 4 tables and 7 transactions. Most of its queries are fairly simple and are read-only. This is also the same as our usual use of in-memory databases: caching frequently read hotspot data. Besides, other benchmarks, such as the Yahoo Cloud Services Benchmark designed for distributed cloud service systems also have their own characteristics[25] .

3 Literature review

In this section, we will review the literature that has been conducted. First, we will discuss some researches related to schema migration and solutions to the problem of zero downtime schema migration. Then, this section will review some researches that are significant to the benchmarking field, concludes with a comparison of the different benchmarks and discussing their advantages and disadvantages.
For the papers in the field of schema migration, some of them conduct theoretical research on online schema transformation, while some papers proposes a solution strategy for online schema transformation. There are also papers that implement their own tools based on different resolution strategies.
For papers related to the benchmarking of schema migration, a paper by Moller provides a review of researches in this area[44] . In addition to this, in some researches related to solutions for zero downtime deployment, the researchers also describe the benchmarks they used. Finally, benchmark design guidelines presented in some papers and books for database testing also contribute to a better understanding of this subject.

3.1 Schema Evolution

There are many synonyms for schema migration. A similar word for "schema" is "database". "migration" has richer synonyms, such as "reorganization", "evolution", "transformation", "change" and "refactor". For some specific kinds of databases, some specific words also have the same meaning as schema migration. For example, for object-oriented databases, “objects change” also means schema migration.
The concept of schema migration was proposed very early. As mentioned in the GUIDE/SHARE Report[32] in 1970 and the CODASYL Data Base Task Group Report in 1971[20] , an important responsibility of the database administrator is to select the appropriate point for database reorganization and users should feel that the database is reorganized as little as possible. In 1974, Fry from the University of Michigan conducted research on database reorganization from the logical to the physical level[16] . He identified the main reasons for government or commercial users to perform database reorganization and discussed the complexities of database reorganization.
Sockut’s paper in 1979 summarized previous research[56] . The contribution of this paper is mainly to define the database reorganization in more detail, classify the types of reorganization according to different levels, introduce the strategy of reorganization and summarize the semantics and languages for specification of logical reorganization proposed by different researchers. In 1992, a bibliography of schema evolution research listed more than fifty papers in this field, the vast majority of which were published after 1987[51] . Among the fifty papers, some are related to relational databases and some are related to object-oriented databases. In 1987, Dadam proposed a method to reduce the time and space overhead of schema changes by not updating all data instances immediately when the schema changes[27] . In 1990, McKenzie extended the conventional relational algebra to support the evolution of a database’s schema[43] . In 1992, Roddick introduced an SQL extension called SQL/SE which can handle schema changes in relational database and in 1993, he described an approach based on historical schema to achieve schema evolution[51] .
In 1996, A paper by Hainaut described a common framework to cope with database evolution[35] . In this paper, he analyzed the process of database evolution and proposed that a good schema evolution framework should solve four problems. By addressing

these four problems, the framework can ensure data integrity and consistency, guarantee data recovery in the event of a failure and at the same time ensure the framework's ability to handle different requirements. Based on this four problem, four strategies are propose and they are showed in the following list.

1. The new schema and the old schema should be as similar as possible except for the structure that must change. And after the new schema is created, the contents of the old database should be transferred.

2. This strategy is backward database maintenance. To implement this strategy, the authors proposed that a feasible approach is to map the evolution of the database to changes in physical files and database structure.

3. This strategy is called database reverse engineering and it is consists of two phases called data structure extraction and data structure conceptualization. In these two phases, the engine restores different contents of the database respectively.

4. Anticipating design strategies require careful consideration of software stability.

Besides, Ronstrom's paper lists a variety of schema evolution tools in the related work chapter[52] . Kim implemented a way to address simple schema changes[40] . Lerner took a different approach to this problem by implementing a special language that propagates data from the old schema to the new schema[41] . There are also researchers who deal with this problem by using user-defined conversion functions and migration functions[33] .

3.2 Online Schema Migration

In the 1970s, the UIDE/SHARE Report[32] in 1970 and the CODASYL Data Base Task Group Report[20] mentioned that choosing the appropriate time for schema migration was one of the important responsibilities of the database administrator. It is also mentioned that for database administrators, the best time for schema migration is on weekends. In 1979 Sockut summed up four strategies for schema migration, two of which need to prevent users from accessing the database, and the other two do not need to take the database offline[56] . Since the 20th century, as computers and the internet have played an increasingly important role in human social activities, people have begun to require computer systems to be online 7*24 hours a day. As one of the conditions for realizing continuous deployment, how to complete the migration of database schema without suspending the production environment has gradually attracted attention. This research field is also called online schema migration. In this thesis, not leaving the production environment (online), specifically, means that read and write operations can be performed correctly when schema migration occurs. Today, the results of related research on schema evolution have been widely used in relational, object and document database products, among which the online schema change capability has become a standard feature of OLTP databases and mainstream relational databases have their own implementations. However, since these implementations still have various shortcomings, some organizations and individuals also developed their own online schema change tools.

3.2.1 Third-party solutions

Ronstrom's 2003 paper is well known to researchers in the field of online schema migration[52] . In this paper, he presents five steps to solve online schema migration problem which and their variants are also adopted by several modern migration tools. The five-step process is

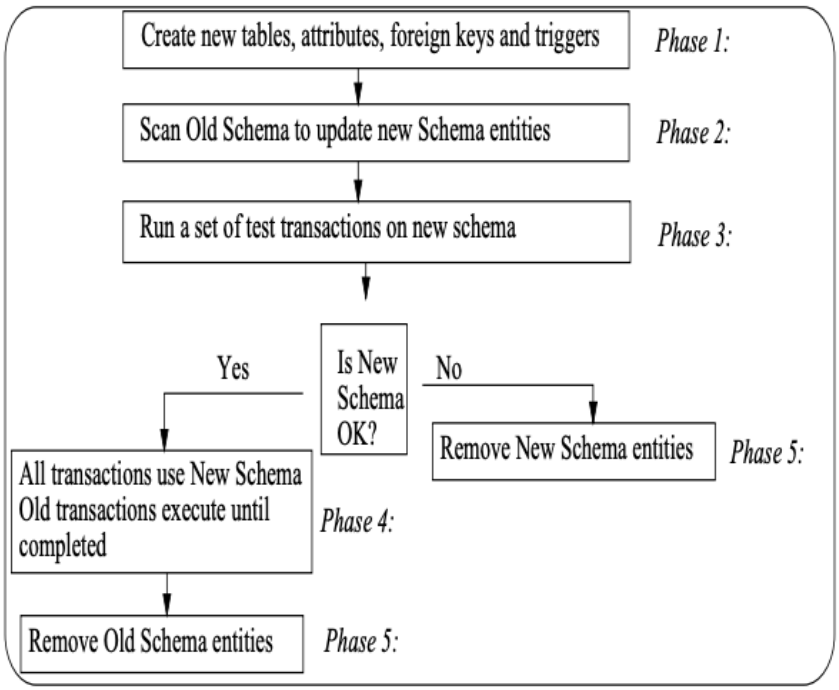


FIGURE 6: Steps of complex schema change[52]

shown in Figure 6 . As shown in Figure 6, in the first step, the DDL causing the schema change is executed and the new table is created. And some triggers and foreign key constraints are added to the new table to ensure the data is up to date. And the second step is the transformation of data. In this step, the data in the old table is scanned and copied to the new table. After this step is completed, the new table can be accessed by the database. The third step is to test the new table, and some test cases are executed on the new schema. And the results of the test will affect the next step. If the test passes, then in the fourth and fifth steps, all transactions will gradually run on the new schema, and the old schema will be gradually discarded. If the test fails, the new schema will be removed and rolled back to the old version. At the same time, in this paper, Ronstrom also pointed out that due to the use of triggers to ensure data synchronization between the old and new tables, when the new table and the old table are working at the same time, users should pay attention to the problem of circular triggers and a possible solution is to label the source of the data change.

Many other online schema migration tools have opted for the same approach as Ronstrom, using triggers to keep data in sync between old and new schemas. And these tools also take the multi-step solution proposed by Ronstrom. This kind of tools including pt-online-schema-change[4] , Facebook OSC[1] , LHM[3] , oak-online-alter-table[2] . When using these tools for online schema migration, after the DDL command is executed, a table with the new schema will be created instead of changing the schema of the old table. This table is generally called a shadow table or a ghost table. Since there is no data in this table at this time, it is temporarily unavailable for access. After this step, the data will be migrated from the old table to the new table. During the data transfer, the write operations performed on the old table will also be synchronized to the shadow table through triggers. After all the data has been transferred and the data has been synchronized, these tools will switch the tables by renaming the shadow tables and bring the tables online.

QuantumDB takes the same approach, but improves upon other tools by adding support for foreign key constraints[28] . When faced with foreign keys, other tools make foreign key constraints temporarily unavailable or refuse to work, QuantumDB will ensure that the constraints are not violated. Compared with other solutions to ensure data consistency, Zhu not only uses triggers, but also uses PostgreSQL's materialized views to ensure data synchronization between old and new tables[66] .
However, some of the previously mentioned tools designed from multi-step solutions still have some problems. First, since an additional ghost table needs to be created, the required system resources will be doubled. The second is that using these tools requires waiting for the database synchronization to complete before updating the front-end application, which causes a delay in application update. To solve this problem, some tools rewrite the new version queries to work on the old schema. However, this approach imposes limitations on schema changes as both old and new version transactions must be both readable and writable on the new schema. On the one hand, this limitation will lead developers to compromise to meet this condition and adopt a schema that is not perfect. On the other hand, there will be a gradual accumulation of technical burden, as this constraint is taken into account with each update. For these reasons, some companies turn to NoSQL to solve this problem.
There are also some problems caused by the use of triggers to ensure data synchronization. The developers from Github listed six problems caused by triggers[2]. They are

1. Triggers will cause overhead, because in some databases, triggers are essentially a stored procedure, and the stored procedure is only interpreted and not compiled. When using ghost tables, every update to the data causes the stored procedure to be interpreted, which is a big overhead on a busy table.

2. Using triggers can cause table locks and even database locks. When concurrent queries compete for resources via locks, triggers also need to compete for their own locks on ghost tables. The developers from Github have shown that this can lead to near or full lock down.

3. Triggers cannot be paused, and there is no way to save resources by pausing execution when the server is busy.

4. The ability to use triggers for concurrent migrations on multiple tables has not been proven.

5. For databases with master-slave architecture, because row-based replication is generally performed between the master database and the slave database instead of statement-based replication, triggers are not triggered by row-based replication.

6. Most current trigger-based tools cannot be used in a multi-server environment.

Out of consideration for these problems, the developers from Github chose to implement their own solution based on logs rather than triggers.

3.2.2 Database vendor's solution

To meet market demand, many databases also provide native support for online schema migration. For example, by using Oracle's DBMS REDEFINITION package, developers can rename, add and delete columns in Oracle database tables without downtime. However,

[2]https://github.com/github/gh-ost/blob/master/doc/why-triggerless.md

unlike some previously mentioned solutions, they do not support the coexistence of multiple schema versions.
Compared with Oracle, MySQL supports more kinds of DDL. MySQL has provided the online schema change (Online DDL) feature since version 5.6, which allows most DDL statements to be executed in parallel with DML. DDL execution is divided into three stages: initialization, execution and commit table definition. This functionality is achieved by using a new type of lock called a metadata lock(MDL). Taking creating a index as an example, in the initialization phase, the database mainly analyzes the DDL statement and determines the execution strategy, and will hold the shared MDL on the target table during the process. In the execution stage after this, the exclusive MDL will be obtained, and the preparations such as creating the rowlog will be completed after the transaction on the current table ends. After that, the original exclusive MDL will be downgraded to shared MDL to allow read and write operations to execute in parallel. Changes involving indexes in new transactions are written to the end of the rowlog. The index building process will also start at the same time. In the final commit table definition stage, the exclusive MDL is obtained again, and the MDL is released after the metadata update is completed. At this time, the process ends and the index is visible to the query.
However, the DDL process of applying a stand-alone database on a distributed database is not feasible. Engineers from Google found that a huge problem was that distributed databases needed to store multiple copies of the schema[49] . If the DDL process of a stand-alone database continued, distributed locks were needed to ensure that no read or write operations were performed during the loading of the new version schema, which was extremely costly. And when the nodes in the cluster cannot sense each other, it becomes an unachievable task. In response to this problem, at VLDB 2013, Google engineers gave a new schema change process to solve this problem by adding two intermediate states which are called delete only and write only. Only delete operations are performed on objects in the delete only state and writes are supported on objects in the write only state and reads are not allowed.
The plan of Google F1 contains two key points. Adding two intermediate states to the first critical point ensures that the cluster has at most two recent metadata versions. The second key point is to add the concept of lease to ensure that within a lease period, nodes that do not get the latest version of the schema cannot submit transactions. As an initial release, many other commercial databases have also improved upon Google's proposal. For examole, TiDB uses a component named PD to determine in real time whether all nodes have completed the schema version update. If some nodes fail, they need to wait for a lease period. OceanBase introduces ObServer, each ObServer has both storage and computing functions and saves a copy of metadata, which can determine whether the metadata versions of both ends are consistent during the task forwarding process.

3.2.3 Complex/Simple schema changes

Another major contribution of Ronstrom's research is the concept of complex and simple schema changes[52] . This change was also accepted and adopted by many researchers [28, 66, 9] . A simple schema change refers to a schema change that can be executed as a transaction, while a complex schema change refers to a long-running transaction. For how to perform a complex schema change, Ronstrom believes that the SAGA model can be used to solve this problem. According to this model, a complex schema change is executed as a set of simple schema changes, or in other words, a set of transactions. Since all schema changes can be classified into add and drop, any schema changes can be rolled back by performing a reverse transaction. Also, because a complex schema change

consists of multiple add or delete transactions, undoing this change is also a long-running transaction.
Zhu proposed a similar concept to Ronstrom in his paper. In this paper, a basic logic operation unit called SMO is define[66] . This basic unit of operation is wrapped by the ALTER command. Similarly, SMOs are divided into simple SMOs and complex SMOs. More than half of SMOs are simple, such as renaming tables and attributes. Complex SMO involves execution, rollback, etc. Unlike simple and complex schema changes, SMO involves not only schema changes, but also data migration. By using SMO instead of using DDL commands directly, Zhu believes that it is more convenient to execute complex commands, and it will improve software reusability and extensibility, make rollback more convenient, and divide complex operations into multiple simple steps.

3.3 Criteria

In this subsection, we review the criteria for tools used to evaluate online schema migration tools. Richter provides a detailed summary of the guidelines proposed by other researchers[50] . These criteria proposed by multiple researchers are further divided into four categories, which are functional, performance, correctness, and tolerance of errors.
According to Richter's paper, there are five papers presenting criteria for implementing online schema transfer tools. The authors of the first paper are Wevers et al[63] , in which these researchers first formulated performance and functional criteria for the transfer mechanism. Then, based on these requirements, they tested two popular databases and a migration tool available on the market. The test results prove that none of the existing tools can well maintain the ACID properties of the database and satisfy their proposed criteria. Functionally, this requirement require that transactions be executed concurrently with as little impact on latency and throughput as possible. Besides, the time of schema migration, whether interruptible and recoverable and memory usage are also important criteria.

Wevers et al were not the first researchers to propose criteria. In 2009, Sockut and Iyer investigated and studied this issue[57] . They have separate requirements for online migration applications and online migration strategies. They point out that even a short period of downtime can cause huge financial losses for a company, so availability is one of the important requirements for an application. For transfer strategies, similar to Wevers et al, they argue that correctness, performance and ability to handle errors are important aspects for evaluating strategies.

There are also some researches that offer their own implementations that also come up with their own criteria. Sheng made demands on the correctness and reliability of the migration tool, and he tested his own tool using the dataset provided by Curino[55] . This dataset contains 170 pattern changes, 168 of which their tool can perform correctly. Zhu pays more attention to the performance of the tool[66] . He designed different data query and data update statements, including random selection, sequential selection, insert, update, delete and scan. Finally, for ease of use, the tool should integrate with the system with as few source code changes as possible.
Richter summarizes and reclassifies the criteria proposed by previous researchers[50] . He divided all criteria into four categories, namely functional criteria, correctness criteria, performance criteria, tolerance of errors criteria. The criteria contained in these four categories and their descriptions are shown in Table 1 .

Category	Criteria	Description
Functional	Expressivity	All transformations can be performed online
	Declarativity	The user should not have to deal with specific implementation details
	Composability	The user can perform many transformations in one go
	Concurrently active schemas	Multiple schema versions can be active at the same time supporting the same data
	Non-invasiveness	The solution should not require major work to applications to support it
	Aborting	The solution should not abort running transactions
Correctness	Transformation of data	All data should be available in the new schema version
	Transactional guarantees	The solution should satisfy the ACID properties
	Application migration	A client should be able to continue querying the database with its current schema version
	Referential integrity	Foreign key constraints have to be enforced at all times
	Schema isolation	Clients should not be able to see any other schema version than their own
	Consecutive migrations	The solution verifies that no other transactions are updating the schema
Performance	Performance degradation	Performance degradation should be within limits depending on the use case
	Space consumption	The solution should not significantly fill up available storage space
	Finite completion time	The solution should perform the transformation in a finite time, preferably as fast as possible
	Time to commit	Data should not take too much time to transform after clients are already using the new version
	Memory usage	The solution should not take up more memory than is allocated to the system
Tolerance of errors	Data recoverability	The solution should have the means to reverse the transformation
	New data reverse transformation	New data should have a transformation back to the previous version
	Transformation resilience	Transformations can be aborted without inconsistencies, preferably restarting from a checkpoint
	Successfully online	The new schema version should only come online when it is successful

TABLE 1: Criteria summarized by Richter[50]

Functionality specifies the functions that the migration tool should have and the specific requirements for those functions. Correctness specifies the result or intermediate state that the data or system should achieve. Performance criteria evaluates the performance the software should achieve. The last criterion makes demands on the behavior of software in the face of systematic and human error.

3.4 Benchmark

3.4.1 Benchmark for databases

Database benchmarks help database administrators choose between different setups, hardware and software[5] . The first standard benchmark for performance testing databases is the Wisconsin benchmark. Before that, although some benchmarks can be used to test database performance, they are application-specific. After implementing the DIRECT database machine in 1981, researchers from the University of Wisconsin concluded that there is currently no universal benchmark. For a more comprehensive comparison of this data machine with INGRES, Oracle and other DBMS, the Wisconsin benchmark was designed and implemented.
However, the Wisconsin benchmark still has many shortcomings. According to Anon et al[31] , the Wisconsin benchmark does not perform well for datamation applications. Therefore, they implemented three benchmarks, scan, sort and DebitCredit. The first two benchmarks tested the input-output capabilities of the system. The last benchmark has a more profound impact on later benchmarks, in which a metric for transactions per second (TPS) is proposed. For the DebitCredit benchmark, TPS refers to the peak value of transactions executed per second, with 95% of transactions requiring response times within one second. Now, TPS is more widely considered as the number of transactions per second. Due to the concept of TPS, this benchmark also became one of the most popular benchmarks at that time.
Another major disadvantage of the Wisconsin benchmark is its inability to test database

performance in a multi-user environment[12] . This is a very serious disadvantage, because in the real world, applications are often multi-user. Benchmarking in a single-user environment means that the database's ability to process transactions and concurrency cannot be tested. The industry quickly recognized the problem and soon several multi-user benchmarks were implemented[11, 14]. Boral's paper pointed out that only four basic database operations are needed to test database performance and the main factors affecting the throughput of concurrent systems are level of multiprogramming, degree of data sharing and transaction mix[11] . Thus the vision proposed in 1983 was implemented, retaining the relational structure proposed in the paper and modifying the benchmark for multi-user scenarios. In addition, he divided metrics into three categories and described them in detail.
The boom in database benchmarks has sparked a vicious competition called the Benchmark Wars[34] . When a vendor publishes benchmark results for its own product, if it achieves better results than its competitors, the fairness of the test will be called into question. If the grades are published by a relatively objective third party, the benchmark may be discredited. The loser in the comparison may also make its product better than the competitors by releasing an enhanced version or through a special configuration. Likewise, winners will retaliate in the same way when losers post new results. This war has brought many unnecessary losses to many companies. In this background, Omri Serlin established the Transaction Processing Performance Council (TPC) in 1988 in consultation with 34 software and hardware suppliers. The main purpose of the TPC is to define benchmarks in various fields. In addition, they also describes a method for calculating system prices and reporting test results.
Since the establishment of TPC, the standards and procedures of benchmarking have been gradually standardized. There are also more and more benchmarks emerging for different segments. Since the establishment of TPC, the standards and procedures of benchmarking have been gradually standardized. There are also more and more benchmarks emerging for different segments. For example, there are TPC-C for OLTP systems and TPC-H for OLAP systems released by TPC. There are also many benchmarks to test the different capabilities of the database, such as SIBench[17] for testing snapshot isolation and The Yahoo! Cloud Serving Benchmark (YCSB) which requires high system scalability[25] . Finally, many companies and research institutes have also implemented multiple benchmarks for different application domains, such as LinkBench[5] and Twitter[19] for social applications, and Wikipedia[61] for knowledge websites.

3.4.2 Metircs

Metircs are used to compare the performance of various aspects of the database. Researchers from the University of Wisconsin designed the first standard relational database benchmark[10] , and in this paper, they also describe how to evaluate database performance. Since the benchmark was run in single-user mode, they designed a strict sequential mechanism. Under this mechanism, only one query is executed at a time, and this step will be repeated many times and the average value will be taken. Elapsed time is the main measure they finally selected. Elapsed time is the same as response time. For a backend machine, it refers to the time it takes for the machine to receive a query and send a response. The Wisconsin benchmark compares the performance of different databases by calculating the total elapsed time spent by 100 queries. In 1993, David J. DeWitt introduced three methods to measure the speedup, scaleup and sizeup characteristics of the database. They require that when the number of processors increases, the database's

query processing capacity should also increase proportionally.
The metrics presented in the paper by Anon et al. are extremely significant for how to use benchmarks to evaluate the performance of databases[31] . In this paper, the metric transactions per second is presented for the first time, which is still used by many OLTP benchmarks to this day. In addition, the author of this paper also points out that the cost performance is also very important for measuring a system. This concept has also been adopted by the TPC committee and has become one of the important criteria for the committee to evaluate the system. However, they also point out that the identification of transactions is difficult because the benchmarks are not performing real transactions. At the same time, there was no way to price the system at that time, so it was difficult to really evaluate the cost performance of a system. Based on this situation, this paper also defines in detail what spending should contain. Their view is that cost should be in units of five years, including hardware and software purchase, installation, and maintenance costs, excluding terminal costs, application development costs and operations costs.
Boral use three performance metric including queries-per-second, illustrative and response time as system indicator[14] . Bitton gave a more detailed summary and classification of the evaluation indicators[11] . She divides metrics into two categories: speed metrics and utilization metrics. The first category of metrics measures the speed of the database, which includes the database response time for stand-alone queries and throughput. The second categories including the utilization of cpu, disk and communication lines.
The opinions of the above researchers have a profound impact on the metrics of the current benchmark. TPC-C benchmark mainly have two indicators, namely the traffic indicator (throughput, referred to as tpmC) and the price/performance (price/performance, referred to as price/tpmC). TpmC describes how many new order transactions per minute the system can process while executing other transactions. Armstrong used the same method as Boral to classify metrics[5] . When designing Linkbench, he argued that the two most important indicators used to measure the speed of the system are latency and throughput. At the same time, for commercial systems, price/performance is also an indicator that must be considered. In addition, he also listed the utilization metrics in detail, including the following measurements:

1. CPU usage of user, system, idle and wait.
2. Number of I/O operations per second.
3. Bytes read and written per second.
4. Memory usage.
5. Persistent storage usage.

3.4.3 Guideline

Researchers from the University of Wisconsin proposed that a good benchmark must have a sufficient amount of data. Benchmarks with too small amounts of data do not reflect how real-world database management systems work. And data that is insufficient makes it difficult to systematically test the database. For example, when the amount of data generated by the join query is insufficient, it becomes difficult to analyze which factor affects the response time. Designing database relationships is another important task of designing benchmarks, which helps users better understand and expand benchmarks. At the same time, before designing attributes, the designer must consider which aspects of the database should be tested for performance. Another work by researchers from the

University of Wisconsin is to develop a collection of queries for testing relational databases, which are:

1. Selection: The select queries must have various selectivity factors.
2. Projection: This kind of queries must have different percentages of duplicate attribute.
3. Join query: Including simple join query and multi-join query.
4. Aggregate function: Benchmarks should test aggregate functions.
5. Update query: The update query here refers not only to the update of data, it includes addition, deletion and modification.

In addition, since the use of indexes and the type of indexes can also have a significant impact on performance, indexes should also be considered when conducting design benchmarks.
The engineering database benchmark follows the design guidelines of the Wisconsin benchmark, uses a different database structure and is tested against an object-oriented database and a relational database[18] . Nelson proposed a benchmark that can also test the bulk function, batch processing capability and locking mechanism of the database[46] . In 1993, Carolyn Turbyfill argued that existing benchmarks are not able to perform different workloads, so a scalable benchmark is necessary[60] . Thus, AS^{3}AP is developed and it can benchmarking a broad range of system.
Sawyer gave detailed guidance on the design and implementation of benchmarks. He believed that a benchmark mainly consists of load specifications and test specifications[53] . Achieving the benchmark specification requires designers to analyze the requirements of the benchmark. The first step is to determine the logic of the application, including the proportion of each type of query in each transaction. He also introduced a method to record the tps rate and it is shown in Figure 7.

Transaction	tps	Table 1	Table 2	Table 3	Table 4	Total
Tran 1	6	2R 1U	6F	-	1R	3R 1U 6F
Tran 2	12	-	3F	1R	1R	2R 3F
Tran 3	4	2R	3F	-	1R	3R 3F
Tran 4	1	1R	1U	9F	1R	2R 1U 9F
R = Random read			U = Update		F = Sequential Fetch	

FIGURE 7: A method used to record tps

In addition, he mentioned that batch work and error checking should be considered in most benchmarks. For the other specification, Sawyer introduced the definition of each measurement in detail and recommended the transaction response time percentile.
Gray summarize previous research on benchmarks in a book *Database and Transaction Processing Performance Handbook*[34] . His point is that due to the complexity of the real world, there is no easy way to benchmark all systems, and therefore, domain-specific benchmarks are needed. For a domain-specific benchmark, it must satisfy the following four criteria. The first is relevance, the operation it performs must be a classic operation

in the related domain. The second is portable, benchmarks must be easy to understand and be able to implement and execute on a variety of hardware and systems. Then there is the scalability, which requires that the load size of the benchmark can be adjusted according to the needs of users. Finally, the benchmark should be simple and easy to understand.

3.4.4 Benchmark for online schema migration

Benchmarks for online schema change are a relatively new area of research, where so far no benchmarks have emerged that satisfy all the criteria proposed by researchers. When testing online migration tools, the vast majority of researchers chose to extend the TPC-C benchmark, such as renaming some attributes or adding entirely new attributes. Some researchers choose to expand on the Wikipedia benchmark, because this benchmark saves multiple versions of the schema, so the data and schema changes come from real production environments.

In 2020, Moller reviewed and summarized the benchmarks in the field of schema migration and pointed out the shortcomings of these benchmarks[44] . The aspects of comparison include data model, workload, data generator, data availability and metrics. The benchmarks they found are Pantha Rei, Twente, STBenchmark, BigBench and Unibench, where the first three of these five benchmarks are applied in the field of schema evolution and the remaining two are just related to schema evolution. According to their observations, only Pantha Rei of the five benchmarks both modeled real-world systems and used real-world data, while other benchmark use data generator to generate distributed data. In this paper, data are divided into three categories: structured, semi-structured and unstructured. The comparison found that only BigBench and Unibench have all three categories of data and the other three benchmarks in the field of schema migration have only one category of data.

Moller also compared the statement types of the benchmarks. Among them, STBenchmark has a special application field and is mainly used in mapping scenarios, while other benchmarks belong to OLTP or OLAP. The last aspect of comparison is metrics, where Pantha Rei tests the system's robustness and adaptability and other benchmarks focus on system performanceNo benchmark can test both aspects at the same time. Based on their observations, they also made their own requirements for benchmarking tools, which are summarized derived from the four assessment aspects above.

Another hitherto unsolved problem in this area is an implementation called "mixed state". The definition of mixed state is given by de Jong, but it has been mentioned by other researchers before. As mentioned in de Jong's paper, supporting multiple versions of data is one of the basic requirements of online migration tools[28] . This is because the actual application is often multi-user, and the update of the application by the user will not be completed at the same time in an instant. Therefore, during schema migration, there will be a state in which the old and new version SQL access the database at the same time, which is called a mixed state. Many tools cannot test their ability to handle multiple versions of data because there are currently no benchmarks and benchmarking tools that implement mixed states. This is also a major problem in this field of research.

In addition, because some databases have their own special DDL implementations, there is no researcher to summarize and enumerate all DDLs. Researchers tend to design benchmarks based on commonly used DDLs. For example, Wevers enumerated and classified common DDLs and they divided all DDLs into five categories [63] . Curino et al also enumerated the schema modification operators they used[**?**] . By comparison, it can be found that there are similarities in the schema migration used in the two papers. And these

common DDLs are also frequently used when researchers are testing their online migration tools.

3.5 Summary

The literature review section begins with a review of researches in the field of schema migration which provides a more comprehensive understanding of schema migration. On the other hand, the difficulties encountered in designing schema migration tools mentioned in some researches, as well as the deficiencies found by researchers when testing their migration tools, are also instructive for the design of our tools. For example, some researchers mention that there is no tool to achieve mixed states. Therefore they cannot verify that the tool supports the coexistence of multi-version data[28, 50, 9] . These papers help us understand the needs of potential users.
The review of researches in the benchmark field is helpful for benchmark design. First, a review of database benchmarks helps us realize how a benchmark should become popular. Benchmark scenarios, metrics and some extra features all help here. For example, TPS is an important reason for the popularity of Debit Credit. And the reason some benchmarks are so popular in a particular domain is because they have fitting scenarios. Subsequent reviews of metrics, guidelines and benchmarks in the field of schema migration describe the design of the benchmarks in more detail. The first two tell us how to think about benchmark scenarios, queries, data, and metrics. While the last part presents some of the issues that benchmarks in the field of schema transfer should be aware of.

4 Treatment design

Designing a benchmark is actually designing two components of the benchmark: the workload component and the metrics component. For a benchmark for online schema migration, the first components include the schema before and after the migration, the transactions executed, and the queries contained in the transactions. Designing the metrics component requires us to consider the environment in which the tests are performed. It mainly includes the indicators used, the time points when the indicators should be collected and how to report the results. This chapter provides an overview of the benchmark design process.

4.1 Workload

4.1.1 Requirement

This subsection summarizes the guidelines for design benchmarks proposed by previous researchers. There are three main sources of these guidelines. The first source is the documents for the different benchmarks, which document the designing process of benchmarks, requirements, models and how to report results, etc. The second type of source is the researches in the benchmark domain. Many of them detail the design requirements for benchmarks. Finally, there are also some requirements from researches in the field of online schema migration, which also mention the requirements for benchmarks because benchmarking the migration tool is an important part of proving the feasibility of the tool. Some requirements have been briefly introduced in the literature review chapter of these researches. This chapter will summarize and define these criteria based on the previous contents.

The first requirement is that the benchmark must model multi-user online sessions. This requirement was first proposed by Anon et al[31] , and is also listed as one of the essential requirements in the documentation of many benchmarks designed by the TPC committee. Single-user and serialized sessions are not suitable for today's computer systems, so benchmarks that only support single-users are no longer popular in the market. Support for simulating multiple users has become an important requirement for database benchmarks.

Another requirement is that the system modeled by the benchmark must be a real-world system or a simulation of a real-world system. A survey by Wang revealed a gap between manual-based benchmarks and real-world benchmarks[62] . It is also mentioned in the documentation of the TPC-C benchmark that the simulated transactions should occur in a real OLTP environment.

The third requirement is that the variety of SQL statements used by the benchmark must be complete. In Boral's paper, benchmarking a database can be done with four basic database operations[15] . SO a benchmark that initially meets this requirement should include four types of query: select, delete, update and add. A more stringent requirement is that these query statements should use different conditions, use keywords such as join, group by, use nested queries, use aggregation functions and sorting, etc.

Another requirement is the variety of data types. Unlike Mark Lukas Moller who divide data according to the degree of structure, we take a different view. In relational databases, semi-structured data is usually stored with delimiters or a specific structure. The database system usually only accesses and modifies it. There is no obvious difference between manipulating this kind of data and manipulating attributes such as text. It is the back-end program that processes it further. Therefore, the definition of data variety here is that the benchmark should include various types of attributes, including numbers, text, time, boolean, etc.

Category	Requirement	Description
Scenario Requirements	multi-user	Benchmarks can simulate multiple users
	real-word system	Benchmarks must be modeled on real systems
	SQL varity	The types of SQL should be comprehensive
Data requirement	data availability	The original data must be available
	data type varity	The types of data should be comprehensive
Functional Requirements	generality	Benchmarks can be used for a variety of DBMS
	independence	Results are not affected by factors outside the test subjects.
	intelligibility	The benchmark should be easy to understand
	error checking	Benchmarks can check and count erroneous results

Table 2: Requirements for benchmarks

The fifth requirement is that the attributes and query statements of the benchmark must be understandable. In Sawyer's book , it is mentioned that the attributes and query statements of benchmarks must be easy to understand, so as to help users better understand and expand benchmarks[53] .
A requirement called data availability is also from Moller et al. According to them, the data used by the benchmark is either real-world data that can be modified by the user, or a data generator that can be set by the user. This helps users to further understand the test data and benchmarks.
The next requirement came from Turbyfill, who argued that the benchmarks had to be runnable on systems of all sizes[60] . This requires that the workload of the benchmark is adjustable. This adjustment should not only be reflected in the total number of executed transactions, the number of users, the amount of data and the upper limit of transactions requested per second should be adjustable.
The portability requirements proposed by Jim Gary enable benchmarks to be executed on systems with different hardware, different operating systems and different configurations. To achieve this, benchmarks should be implemented taking into account the differences between different systems, and adopt some solutions that can be cross-platform and cross-database.
The next requirement is the independence of the results. According to the documentation of the TPC-C benchmark, the results of the benchmark test should only be affected by the system under test and the state of the system at that time, including the hardware, software and configuration files of the system. This requirements ensures that benchmarks can objectively report on the performance of the system.
Another requirement from the TPC-C document and Sawyer's book is correctness or error checking capability. This capability requires that database queries against benchmarks return correct results and modification operations allow correct updates. In addition, the benchmark should be able to judge the wrong operation of the database, record the error when it occurs and report it after the test is completed.
In order to present the benchmark requirements we have summarized in a more intuitive way, we categorize them and show them in Table 2. As shown in the figure, we divide all the requirements into three categories.

4.2 Choosing Tool and Benchmark

Secondary development, in simple terms, is to make specific modifications and functional extensions on the existing software to achieve the desired new functions. Generally, this

kind of development will not cause huge changes to the original architecture and the original code can be reused, so it can greatly save manpower and time resources. As mentioned earlier, the purpose of this thesis is to design a benchmark and a benchmarking tool for the scenario of online schema migration. In this chapter, we will first choose the tools that will be modified and then the benchmarks that will be based on. In the previous sub-chapter, we have introduced the requirements of benchmarks, in this chapter we will also selecting benchmarks base on the requirements listed in Table 2.

4.2.1 Tool

The selection of benchmarking tools will mainly refer to the following aspects:

1. The first aspect is document integrity. The secondary development of software requires a good understanding of the software. Detailed documentation and comments can help developers do this quickly. Conversely, a software that is not well-documented will result in a lot of time spent understanding the source code, and may even cause some bugs that are difficult to fix.

2. The second aspect is language friendliness. Developers often get better results when they program software in languages they are familiar with. Some projects written in niche programming languages will first increase the learning cost of developers, resulting in an increase in development difficulty and development time. After the development is completed, it is also not conducive to further modification of the software because there are fewer developers familiar with the language.

3. The third aspect is functionality, which means how many functions the software provides. For example, some tools can only test the throughput of the databases, and some software can also perform benchmark tests in addition to testing the throughput. More features means more comprehensive database testing and maybe more code that can be reused, so this is a very important aspect.

4. The forth aspect is generality. Since there are many relational databases, such as Oracle, MySQL and PostgreSQL and there are some differences between them, generality is used to indicate how many different databases the tool support.

5. The fifth aspect is software extensibility. The secondary development of software that does not have good extensibility may involve substantial modification of the source code, which should be avoided from the perspective of software engineering.

6. The last aspect is benchmark diversity, which depends on how many benchmarks the tool provided. For some tools that have already implemented some benchmarks, it is easier to make modifications based on their original benchmark implementation than to design and implement the benchmark from scratch. The more benchmark implementations the tool has, the more options there are in choosing which benchmark to base on.

According to the above requirements, we selected a number of existing open source software for comparison. Some software itself has the function of benchmarking, and some software can perform performance test on the database. In particular, if a piece of software satisfies a requirement, but the way it does so is too complex, we'll point it out and give it a evaluation.

The basis of selection is visually displayed in Table 3 and 4. We rate each feature of each tool on a scale of one to five. The evaluation of the first four abilities mainly comes from their GitHub homepage or official website documentation. In order to better evaluate the latter two capabilities, we conducted a source code level inspection of the software.
Almost all tools provide detailed manuals, but none of them go into detail on how to add a benchmark or extend functionality. Among them, only OLTPBenchmark briefly introduced that it is possible to support more databases and add benchmarks by referring to the existing code, which is the reason for the higher score of OLTPBenchmark.
The language-friendliness rating is based on *The State of Developer Ecosystem 2021* published by Jetbrain. In this report, Jetbrain announced the proportion of users of each programming language. Programming languages ranked 1-5 in this report will receive five points, and those ranked 6-12 will receive four points. Three points will be awarded to other programming languages appearing in Jetbrain‘s report. Other languages are classified as "Others" usually receive a maximum of two points. A special case is TCL used by HammerDB, because the language is popular in the database testing space, we give it three points instead of two.
Since the purpose of this secondary development is to implement a benchmarking tool, a important aspect of functionality is whether the software already has the ability to perform benchmarking. If the software can perform both benchmarking and stress testing or stability testing, it will get five points, however none of the software does that. We give three points to software that can perform database tests but cannot perform benchmark tests. This is the reason why Jmeter only gets three points although it provides many features to test the performance of the server.
The generality score depends primarily on whether multiple relational databases are supported. If multiple databases are supported, we will evaluate how they provide this capability and deduct points for less convenient ones. HammerDB is one example of a deduction. Although HammerDB supports a variety of databases, its generality is achieved by implementing each benchmark for each database separately. This inconvenient implementation increases the effort when adding new benchmarks.
For secondary development, software extensibility is a very important aspect, so when the final score is summarized, the score of this ability will be multiplied by two and then counted into the total score. We will also conduct a analysis of the capabilities of each software. OLTPBenchmark achieves the highest score in this capability because its architecture allows developers to easily add benchmarks to software through inheritance. The way of extending H-Store is similar and also requires the implementation of abstract classes. But it is slightly more troublesome than OLTPBenchmark because the latter has a richer level of abstraction. The way of adding benchmarks for HammerDB and DatabaseBenchmark is similar. The way of adding benchmarks to HammerDB and DBenchmark is similar. Developers need to implement benchmarks and write specific files for different databases. Jmeter itself does not provide benchmarking functions. If developers want to add new testing functions, they need to implement subclasses that inherit from the abstract class 'AbstractJDBCTestElement'. Sysbench can add benchmarks by adding Lua scripts. The remaining tools have less reusable code when adding benchmarks because of their relatively simple architecture.
Benchmark diversity is scored based on a internal comparison among the nine tools, with the tool supporting the highest number of benchmarks being awarded five points. HammerDB received four points for providing multiple benchmarks, and Sysbench received three points for providing the ability to perform benchmarks via user-supplied Lua scripts and having ready-made scripts. The remaining tools that support only one benchmark

Tools	document integrity	language friendliness	functionality
Benchbase(OLTPBenchmark)	4	5	4
Sysbench	3	4	4
HammerDB	3	3	4
dbbench	3	4	4
pgbench-tools	3	5	4
sql-bench	3	2	4
Database benchmark	3	4	4
Jmeter	3	5	3
H-Store	4	5	4

Table 3: score sheet 2

Tools	generality	software extensibility	benchmark diversity
Benchbase(OLTPBenchmark)	5	5	5
Sysbench	3	3	3
HammerDB	4	4	4
dbbench	5	3	2
pgbench-tools	3	3	2
sql-bench	3	3	2
Database benchmark	5	4	3
Jmeter	5	3	1
H-Store	5	4	5

Table 4: score sheet 2

received two points.

The final results are shown in Table 5. We can see that OLTPBenchmark achieves the highest overall score, and in addition to that, its score is also very prominent in every ability. It provides very good software extensibility so that developers can expand the functions of the software with less modification to the source code, which is in line with the requirements of software engineering.

Tools	Total score
Benchbase(OLTPBenchmark)	33
Sysbench	23
HammerDB	26
dbbench	24
pgbench-tools	23
sql-bench	20
Database benchmark	27
Jmeter	23
H-Store	31

Table 5: Total scores

Benchmarks	Description
TPC-C	The model used in the TPC-C test is a large wholesale commodity company.
Wikipedia	The data for the Wikipedia benchmark is derived from real-world data and maintains a record of schema changes.
Twitter	A benchmark for social networking applications.
AuctionMark	An OLTP workload modeling an online auction site.
SEATS	The Stomebraker Electronic Airline Ticketing System benchmark models an online ticketing service.
CH-benCHmark	CH-benCHmark is a hybrid workload, which is a mix of TPC-C for OLTP and TPC-H for OLAP.
LinkBench	LinkBench is another social network database whose data source is Facebook's social graph data.

Table 6: Introduction to benchmarks

4.2.2 benchmark

OLTPBenchmark itself provides the implementation of fifteen benchmarks, covering lots of current popular benchmarks. This advantage of OLTPBenchmark greatly facilitates our choice of basic benchmarks. Modifications to existing benchmarks can save development time on the one hand. On the other hand, because the original benchmark has been tested by many users, this method can also increase the user's confidence. Therefore, what follows is an evaluation of some benchmarks selected from these fifteen benchmarks against the requirements presented earlier. Some benchmarks were not considered because they were not designed for relational databases. A brief description of each benchmark is given in Table 6.

Since these seven benchmarks are all very popular benchmarks, Almost all of them meet the ten requirements for benchmarks summarized above. However, we still remove some benchmarks from candidates through comparison. The first benchmark to be abandoned was Twitter, because it is similar to LinkBench in modeling social networking applications. However, Twitter has relatively little data compared to LinkBench, making it harder to gain a better understanding of the benchmark and make changes. However, LinkBench also does not meet our requirements, because its data model is relatively simple, and it has fewer types of data types and SQL statement types. The subsequent selection of benchmarks is mainly based on the following criteria. The first is whether the benchmark's model is widely used and representative in the real world. The second is the popularity of the benchmark. The last criteria is whether the benchmark can be easily changed and applied in the field of online schema migration. We then divided the remaining benchmarks into two categories based on their models. The first category is only Wikipedia, because it models a knowledge system. The rest of the other benchmarks are based on some kind of online trading system. In the second category of benchmarks, we choose TPC-C as their representative. Firstly, the online transaction scenario simulated by the benchmark is very classic, and e-commerce is one of the hottest areas of the internet. Secondly, this benchmark is a famous database benchmark in the world, and lots of database vendor have published its own TPC-C benchmark score. Finally, the benchmark is also easy to be modified due to the large number of tables, data types and query types it has.

The choice between Wikipedia and TPC-C is mainly based on the following considerations. The TPC-C benchmark has the upper hand in the three criteria presented above. However, Wikipedia maintains multiple schema versions and their data and the schema migrations

that cause these schemas to differ have also occurred in production environments. Finally, after careful consideration, we decided to combine the strengths of the two and modify the TPC-C benchmark with reference to the schema migration that happened in Wikipedia.

4.2.3 New requirements for online schema migration benchmark

Compared with traditional benchmarks, benchmarks applied in the field of online schema migration have some new requirements. However, no researchers have summarized these requirements and some requirements are scattered in researches related to online migration tools. Although Moller et al. compared and summarized five benchmarks in the field of online schema migration and present their requirements for benchmarks in this field[44] . The evaluation method they propose is not significantly different from the traditional method of evaluating benchmarks in the database field. Jim Gary mentioned that due to the wide variety of applications, it is difficult for general benchmarks to test the system well and benchmarks for specific fields are becoming more and more popular. To fill this gap, this subsection will identify some new requirements that apply to benchmarks in the domain of online schema migration.

The first requirement is called continuity. In general, schema migration can be divided into two categories. In some cases, schema migration will cause changes to the corresponding DML statements. For example, after renaming a column, it is necessary to use the changed name of this attribute to enable the query to be executed correctly. For this type of schema migration, the query statements before and after the change must be executed continuously or after waiting for a time set by the user. Currently, a popular implementation is to use the functionality provided by benchmarking tools to execute multiple benchmarks consecutively. However, this inevitably leads to very brief pauses. The tool should minimize or eliminate the pause time.

The second requirement is inconsistency. This is one of the basic requirements of benchmarks in this field. Inconsistency requires that the benchmark must change from one schema to a different schema. The inconsistency also includes that when the schema would cause the DML statement to change, the query before and after schema migration may also change. This requires the ability of the migration tool to handle multiple versions of SQL.

The third requirement is that the benchmark must implement the mixed state. For real-world applications, the requests made by the user do not all change at once. Therefore, the database sometimes receives multiple versions of the query during schema migration. Implementing the mixed state can simulate this scenario and test the ability of the database to process multiple versions of data query at the same time. There are currently no benchmarks and benchmarking tools that implement this requirement.

The fourth requirement is completeness. Databases can undergo a variety of schema migrations. Taking MySQL as an example, it provides twenty-seven different online ddl operations. To ensure completeness, benchmarks should provide the ability to test multiple schema migrations.

The last requirement is that the designer should consider the units and rate at which the query changes. Also taking TPC-C as an example, since it initiates requests to the database in units of terminals, the version of queries issued by a terminal should be changed at the same time. At the same time, the rate at which the terminal changes should also be considered. For example, for a mobile application that has released a new version, the number of users updated every day should generally follow a trend of rising first and then falling.

4.3 Metrics

4.3.1 Differences with database benchmark

Metric is another important component of benchmark. In the literature review section we have summarized the metrics used by traditional database benchmarks. These metrics measure database utilization (including performance, stability, etc.) and price. However, the situation is different when these metrics need to be applied to evaluate databases at the time of schema migration.
The main reason for this difference is that the indicators are designed for different purposes. In 5.3 we mentioned that inconsistency is one of the requirements for benchmarking in the field of online schema migration. Therefore, benchmarks in the field should also be able to measure the impact of such changes compared to traditional benchmark indicators. This requires us to make certain modifications to the indicator to meet this requirement.
Another difference is the definition of "system" being tested by the benchmark. For traditional benchmarks, the system here should refer to the database and the hardware and software that support the database. But for benchmarks in the field of online schema migration, the "system" should be based on the original definition plus online migration tools. Although some databases already provide the online schema migration function, it is not considered as a basic function that must be provided. Therefore, database administrators often use schema migration tools when schema migration is required. Simply put, testing online migration tools is also the goal of benchmarks in the field.

4.3.2 Designing process

The following content mainly introduces how we process the indicators according to the above mentioned differences so that they can be better applied to the field of online schema migration.
From the metrics, we can find that traditional database benchmarks focus on measuring the overall performance and extreme performance of the database. Words such as "average," "maximum," and "minimum" are often used as adjectives for traditional benchmark metrics. In contrast, benchmarks in the field of schema migration focus more on the process of testing. Many researchers use line graphs to illustrate that their designs have little impact on database performance. Words such as "average" rarely appear in researches in this field. Therefore, this benchmark will also use line graph to present the changes in the indicators. In addition to this, identifying the stages into which the whole process should be divided is also of great help in analyzing changes. To achieve this, several important time points are identified, which are the time when the online migration tool starts and completes the migration and the time when the mix state starts and ends. The importance of the latter has already been explained before. For the former, some online migration tools will copy data in the background during migration, and this non-in-place copy will affect database performance. Therefore, recording these two points in time has a unique meaning: comparing the time when the migration tool completed the migration and observing the impact of the tool on the metrics.
The criteria for online schema migration tools summarized by Nick Geral Richter can help us design benchmarks for this field[50] . And the criteria is presented in Table 1. In the previous content we mentioned that online migration tools used by users can have an impact on the results of benchmark tests. Some of these requirements should be guaranteed by the software itself, and they are not the benchmark's responsibility to test whether the requirements are correctly met. For the remaining requirements, Table 7 describes the

Criteria	Strategies
Concurrency active schema	In the mixed state, requests made by the terminal should return correct results and not be blocked.
Transformation of data	Querying the data after the migration is complete must return correct results
Application migration	For each terminal, subsequent requests should always return correct results.
Performance degradation	The performance change of the migration process will be observed through the performance change curve.
Space consumption	Monitor changes in storage space.
Finite completion time	Observe the point in time when the schema migration tool starts and completes the migration.
Meomory usage	Monitor changes in memory

Table 7: Strategies

strategies for testing them.

Based on the strategies shown in Table 7, we made the following modifications to the indicators. In order to measure whether the migration tool meets the first three requirements, the number of transactions completed per second must be counted to see if blocking occurs. At the same time, the result returned by the database should also be checked to verify that it was executed correctly. These indicators will be presented in the form of line graphs. For the last four requirements, the changes of the corresponding indicators should be recorded, and at the same time, important time points should be presented on the line chart. These metrics include memory usage, storage space usage, transactions per second (throughput), and average response time per second.

Although it doesn't make much sense to average over the entire process of performing the benchmark. This does not mean that calculating the average of the metric at each stage is meaningless. Averaging the metrics for each stage can measure the performance of the system at that stage, and comparing it with other stages can also observe the effect of pattern transfer on performance. Based on this comparison, we design two metrics. The first metric is called the schema migration process loss rate (P), which is used to measure the performance degradation during the mode transfer process. The second, called the schema migration result loss rate (R), measures the performance change before and after schema migration. They are defined as follows:

$$P = \frac{avbsm - avdsm}{avbsm}$$

$$R = \frac{avbsm - avasm}{avbsm}$$

where avbsm is equal to average value of metrics before the schema migration, avdsm is equal to average value of metrics during the schema migration and avasm is equal to average value of metrics after the schema migration.

However, the above two metrics cannot be used for comparisons between migration tools because they do not take into account the impact of migration time. A test tool can improve its performance on these two metrics by limiting the data transfer rate, which will increase the time for data migration to complete. However, this practice violates the finite completion time criteria. We will use the following metrics when comparing the

throughput impact of different migration tools.

$$Q = \frac{(tpsbsm - tpsdsm) * mt}{C * tpsdsm}$$

where tpsbsm= tps before schema migration, tpsdsm= tps during schema migration, mt=migration time and C is a constant value.

5 Treatment implementation

The next step after designing the treatment is to implement it according to the design. In this chapter, we will first analyze the architecture and source code of OLTPBenchmark, explore how it should be modified and then introduce how we can achieve the requirements mentioned above.

5.1 Analysis of OLTPBenchmark

OLTPBenchmark is a benchmark suite developed in Java and users can use it to load and execute multiple database benchmarks. Users use OLTPBenchmark for benchmarking through the command line. The required parameters of the command include the benchmark to be used and the operation to be performed. Subsequently, OLTPBenchmark will set the "workloadconfiguration" object according to the input parameters and the content of the configuration file, including the database user name, password, benchmark, number of terminals and batch size, etc. This settings object will be used later to generate the "benchmarkmodule" object. At the same time, in order to provide the ability to execute multiple benchmarks successively, OLTPBenchmark uses a list to store “benchmarkmodule” objects. Finally, the tool will choose what to do next based on the "-b” parameter provided in the command. OLTPBenchmark divides the execution of the benchmark into four stages, which are database creation, data loading, workload execution and clearing in order. Since the online schema migration benchmark does basically the same thing as the database benchmark during the create database phase and load data phase. Next, we will skip the above two stages and analysis how OLTPBenchmark performs the workload.
During the workload execution phase, OLTPBenchmark will complete the creation of workers. The type of worker varies according to the selected benchmark, for example using the TPC-C benchmark will create a “TPCCWorker”. The executework function in the Worker object specifies how the benchmark executes a single transaction.
OLTPBenchmark uses scripts to create the database. Therefore, when the original schema of the new benchmark differs from the one on which it is based, the developer needs to modify or rewrite the script. Loader and worker objects are responsible for data loading and workload execution respectively. Rewriting these two objects can change the operation of the corresponding stage of the benchmark.

5.2 Modifications of OLTPBenchmark

In the previous chapters, we presented five requirements for the benchmark, which are continuity, inconsistency, mixed state, completeness and rate control. This subsection begins with an overview of how we modified the OLTPBenchmark to enable it to perform benchmarks that meet these five requirements. It then introduces some additional modifications that make it better applicable to the field of schema migration. Fig 8 shows the class diagram of the modified core components of OLTPBenchmark.

5.2.1 Achieving continuity and inconsistency

The method we use to achieve inconsistency and continuity is called "replacement of prepared statements". Originally, OLTPbenchmark used prepared statements to access the database. Since traditional benchmarks do not involve DML changes, the string objects used to generate the prepared statements are set to constants. However, simply setting the prepared statement as a variable and substituting it when it needs to be changed doesn’t

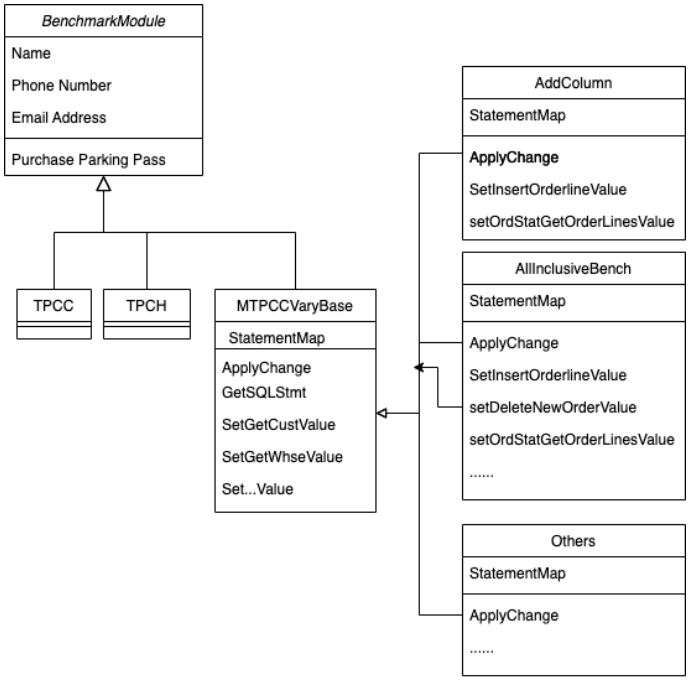


FIGURE 8: Class diagram for the core modification

work well. This is because sometimes prepared statements are executed differently before and after a change. For example, when adding a new column, the prepared statement needs to set a different number of parameters. Handling this situation with lots of if/else statements would make the code hard to read and hard to extend. We solve this problem by centrally managing each worker's prepared statements. All prepared statements used by a worker are created, executed and changed in a class called "VaryBase" . This class will determine the stage the worker is in to choose the appropriate execution method. In this way, we achieve continuity and inconsistency of query statements.

5.2.2 Mixed state

Mixed state is implemented with the help of "VaryBase" class. Each worker has its own unique "VaryBase" object, and the execution of prepared statements in a transaction is changed from transaction control to managed by this object. Calling the "applyChange" method of the object will complete the replacement of the prepared statement and change the internal state, thereby changing the execution process of the prepared statement. Since each worker has its own "VaryBase" object, and these objects do not affect each other, the mixed state is achieved.

5.2.3 Completeness

There are two ways to accomplish the completeness requirement. The first is to design a very complex benchmark that includes almost all types of schema migrations. However, this is a complex, difficult and unsatisfactory work. We will explain the reasons in detail in the later section. In view of the impracticality of including all types of schema migrations and the variety of user needs, we decided to achieve this requirement in a way that provides good extensibility for the tool. Therefore, users can add benchmarks according to their needs. We believe that with our designed architecture, users can simply add their own

designed benchmarks applied in the field of online schema migration. Users can add new benchmarks in the following ways. First, the new subclass of “VaryBase" class should be implemented, and the user should implement the ”applyChange“ abstract method in it to replace the prepared statement. Second, how to execute prepared statements affected by schema migration should also be rewritten. Next, the tool will automatically call and count the new methods designed by the user.

5.2.4 Rate control

OLTPBenchmark originally only provided rate control for a single worker, such as limiting the number of transactions performed by a worker per minute, it did not provide any form of overall control. Therefore, there is no way for the software to have overall control over the rate at which the terminal changes. To solve this problem, the "ControlCenter" class was designed and implemented. This class can query the state the worker should be in and notify the worker when a query version change should occur. It is also responsible for providing a schedule of terminal changes based on parameters given by the user.

5.2.5 Extended configuration file

The original configuration file of OLTPBenchmark can not meet our requirements, so it has also been modified. First, since it does not contain some settings that apply to the schema migration domain benchmark, we have expanded the content of the file. Some settings such as the client's change rate, schema migration type selection and schema migration start conditions are added. This change also affected the configuration file loading process, so we've optimized this process as well.

5.2.6 Portability

Portability is achieved through the use of custom scripts and design patterns. By writing different scripts and files for different databases and using JDBC, the benchmark can complete database creation, data loading and workload execution for multiple databases. For example, for different databases, because they will be different in syntax, we prepare the creation scripts for different databases and call them according to the selected database type.

5.2.7 Metrics

OLTPBenchmark has provided a variety of functions to display the results, and we have further implemented the software's result analysis function based on these functions. OLTP-Benchmark can report throughput and latency in milliseconds and dump the results. Based on these results, we can perform statistics on the indicators according to the ideas and formulas proposed above.

5.3 Benchmark Design and Implementation

This subsection first describes why an all-encompassing benchmark is not suitable for schema migration scenarios. The multiple sub-benchmarks provided by the tool are then introduced, including the schema migrations they perform and their purpose. We divide schema changes into simple changes, which are usually caused by a single DDL statement, and complex changes, which are caused by multiple. At the same time, since we believe that an all-inclusive benchmark is not well suited for schema migration scenarios. We

used a number of sub-benchmarks, each of which simulated a kind of schema migration. These sub-benchmarks are called benchmark cases. In addition, the transaction of new orders plays a major transaction role in the TPC-C benchmark. An additional benefit of modifying the "Orderline" table is that operations on this table include updates and inserts, as well as selects. Therefore, making changes to this table provides a more comprehensive measure of the performance impact of schema migration. We also modified the original schema of the TPC-C benchmark, and the changed schema is shown in Figure 9.

5.3.1 Why extensibility

In previous chapters we mentioned that we achieve completeness by providing convenient extension methods. The reasons for not using a benchmark that includes all kinds of schema migrations are as follows:

1. An all-inclusive benchmark is difficult to implement and it can lead to very complex situations. Usually a schema migration does not use a lot of DDL, so it is not necessary to have a benchmark that tests all the DDLS simultaneously

2. Not all databases support simultaneous execution of multiple DDLs. An all-inclusive benchmark may cause compatibility issues.

3. A large benchmark may result in poor testing of individual DDL. If a schema of an infrequently used table is migrated, then the TPS will not change significantly because the service is used less frequently. And if there's a performance hit, it's hard to know which DDL caused it.

4. Good extensibility is necessary. On the one hand, only the user knows what they want best, and on the other hand, changing the all-inclusive benchmark can be cumbersome, assuming there are new DDLS in the future.

5.3.2 Simple changes

In this chapter, we present all benchmark cases that simulate simple changes. Since not all schema migrations will be used in production, we design benchmarks only for frequently used schema migrations. The principle for determining frequent use comes from Wikipedia, which statistics schema migrations that have occurred in the database[61] . The statistical result can be seen in Table 8. In addition, we follow MySQL's classification method and divide all schema migrations into six categories. However, since the last category of Partitioning Operations is rarely used, we do not implement it.

5.3.3 Complex changes

In this chapter, some complex changes are introduced. Although according to the Mediawiki dataset and statistics on ING Bank's Changeset provided by Richter[50] , the vast majority of schema migrations that occur are simple. Some complex changes still happen in production. The main content of this chapter covers the scenarios in which these complex changes may occur and the changes in the schemas they cause.

Complex change 1

The first complex change scenario considers the structuring of semi-structured data. Initially, the shipping information is semi-structured, consisting of multiple types of information and now it is divided into two columns for courier name and contact information.And the delivery information column is dropped.

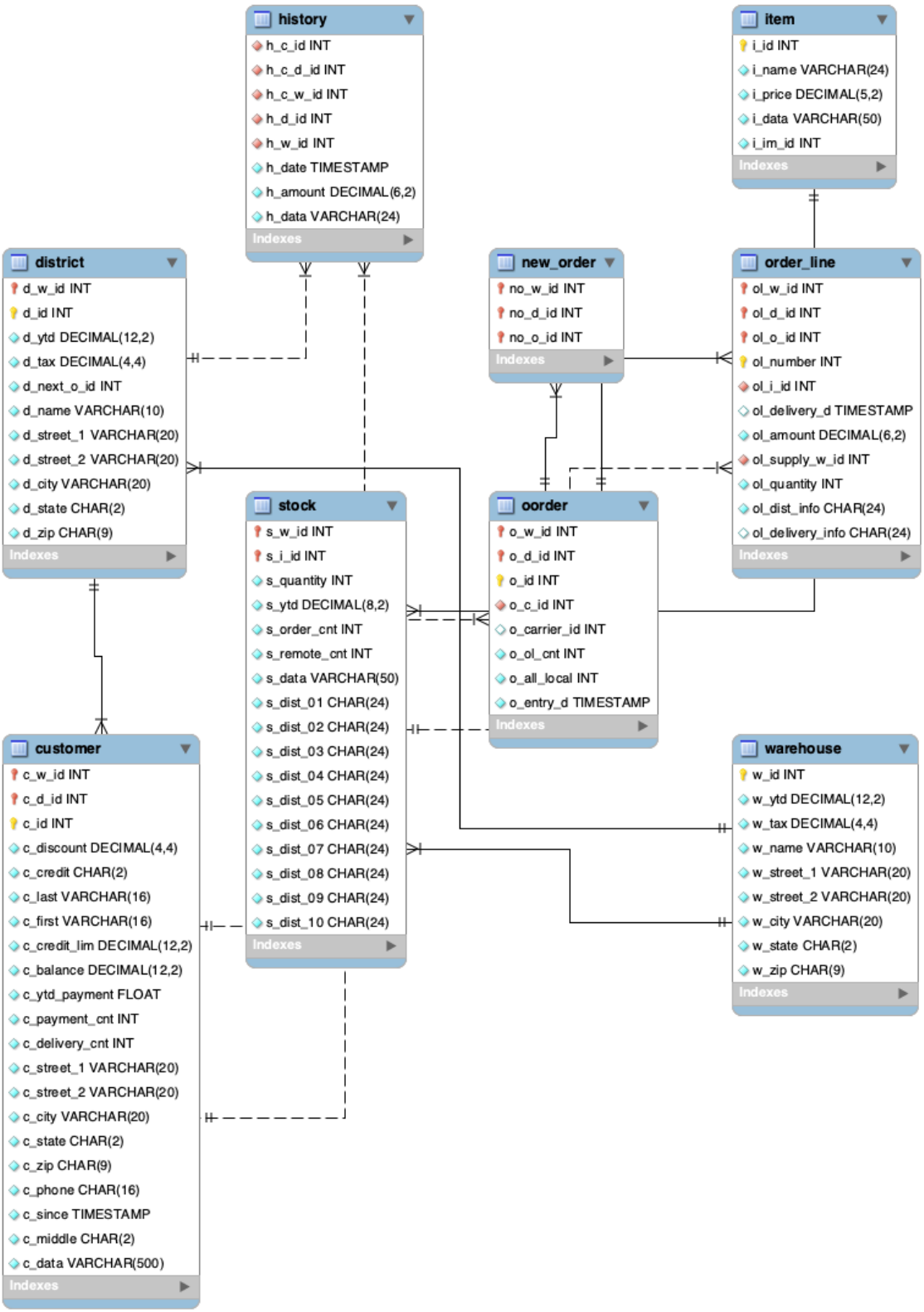


FIGURE 9: Original schema of the new benchmark

DDL	number of usage	percentage of usage
create table	24	8.9%
drop table	9	3.3%
rename table	3	1.1%
distribute table	0	0.0%
merge table	4	1.5%
copy table	6	2.2%
add column	104	38.7%
drop column	71	26.4%
rename column	43	16.0%
move column	1	0.4%
copy column	4	1.5%

Table 8: Statistical result of different DDL[61]

	Index operation
creating an index	Create a composite index on the three columns OL_id, OL_item and OL_order.
dropping an index	Drop the composite index (S_warehouse, S_item) on the stock table.
adding a fulltext index	Create a full-text index on the data column of the history table
	Column operations
adding a column	Create a new column in the orderline table called tax. The type of this column is float and the value can be null.
dropping a column	Delete the delivery_info column in the orderline table.
renaming a column	Rename the amount column in the orderline table to the size column.
changing the column type	Change the type of number column in orderline column to use a greater range of integers.
adding default values	Set default value for delivery info column.
	Foreign key operation
adding a foreign key constraint	Add a foreign key constraint to the orderline table, which requires that no constraints be established for this table when the database is initialized.
dropping a foreign key constraint	Remove the foreign key constraint on the orderline table.
	Table operation
renaming a table	Rename orderline to orderitem.

Complex change 2
The scenario that this complex change takes into account is to change the primary key. The primary key is an important attribute in a data table and changing it is a costly operation that results in reindexing, data transfer and blocking. Therefore, in practice, changing the primary key happens rarely. However, changing the primary key is a classic schema migration and this thesis is still designed for it. Mostly composite primary keys are used in TPC-C. Create a new column named item_new_id and use it to replace the i_id column in the item table.
Complex change3
Add a new column called tax_rate to the orderline table. After that set the default value of the tax column to 0.21. This scenario takes into account that after adding a property, the default value of the property is sometimes set.
Complex change4
This complex scenario takes into account that all items in the same order are not necessarily delivered at once by the same courier (like Amazon). Therefore, the courier information is recorded in the orderline table. First create a column called carrier_id in the orderline table, and then set the value of this column according to the data in the order table. Finally, the new shipping information will also be updated to the orderline table instead of the order table. Finally, the courier id column in the order table needs to be removed.
Complex change5
Sometimes, with the increase of the company's business volume, some data are required to be pre-calculated and stored in the database so that users can quickly obtain or analyze users when querying. Based on this, the average cost per order and the highest cost two columns are created in the consumer table and index them. This change is mainly for testing adding multiple columns at once.

5.4 Summary

In this section, we first analyzed the source code of OLTPBenchmark and figured out the execution process of OLTPBenchmark. This further analysis is helpful for subsequent modifications. Next, this section describes how the tool modifies OLTPBenchmark to meet the new requirements presented in Section 4. The core modification to meet these requirements is that the string used to create the "preparedStatement" is no longer defined as a constant, but is obtained from a object named "varyBase". This modification enables "preparedStatement" to be replaced during benchmark execution. What follows is an introduction to why the tool provides extensibility rather than implementing an all-encompassing variety. Benchmarks for schema migration. Finally, this section lists and describes the schema migration "benchcases" the tool has.

6 Treatment evaluation

The final step in the design science cycle is the evaluation of the treatment. In this chapter, as one of the keys to verifying software functionality, unit testing will be performed to test whether the tool behaves as expected and to detect possible bugs early. On this basis, the software will be further analyzed and tested whether it meets the requirements mentioned above. After the evaluation of the software is completed, we will also test different databases and migration tools to evaluate them and explore the mechanisms of different databases in the face of schema migration. Finally, the problems found during testing will be listed and introduced.

6.1 Unit Testing

The developers of OLTPBenchmark have tested the software quite comprehensivly with JUnit. However, the execution flow of some classes is modified during further development. Thus, it is necessary to unit test the software with more cases. The main objects of unit testing are the modified classes as well as the new classes.
Some use cases are designed to test the newly added classes. For example, the "Change-Controller" class is mainly responsible for setting the time of worker change, determining whether the worker needs to be changed, and recording some statistics. The main purpose of unit testing the class is whether the functions of the class can get the correct result. The worker class has also been modified, and now it needs to determine whether it needs to be changed before executing the transaction. In addition to that, it needs to be able to get the correct information about the changes. Therefore, the new test case needs to provide more information than the previous test case. These similar changes occurred in unit tests for some other classes. Some new test cases are also added for some of the classes that have not been modified. The unit tests found no errors and all test cases passed without a hitch.

6.2 Performance Testing

Performance testing of benchmarking tools focuses on their memory usage as well as their CPU usage. In addition, it is also worth paying attention to whether it can guarantee the same amount of workload under the premise of running for a long time. This is called the consistency of benchmarking tools. In order to measure the performance of the above aspects of the tool, several long-term experiments were designed.
To test the tool, the database management system used is MySQL and the version is 8.0.13. The machine running the software is a machine with 8G RAM and and a dual-core CPU. The maximum heap memory of the Java virtual machine is 2048MB, and the minimum value of the new generation size is 1228MB. And the garbage collector used is G1. For the benchmark, the key settings include that the number of terminals is 20 and the number of warehouses is 100, and the benchmark run time is 6 hours. More specific settings are shown in Figure 21. During this time, Jconsole will be used to monitor the performance of the tool.

During the loading phase, the tool determines the number of loading threads based on the number of repositories. Assuming the number of warehouses is n, the tool will use n+1 threads to load the data. The number of working processes is equal to the maximum number of threads available at that time, and other threads will be added to the queue. During the work phase, the number of threads is equal to the number of terminals.
The changes in heap and CPU usage are shown in Figure 10 and Figure 11, respectively.

The loading phase took a total of 2111 seconds, during which the heap memory usage was between 50 and 100MB and the CPU usage was around 7 percent. During the execution phase, the size of the heap memory increases significantly, however, the maximum value is still less than 1000MB, which does not exceed the maximum heap size of the Java virtual machine. And at this stage, the CPU usage fluctuates around 12%. These two results show that for this machine, memory size and CPU are not the bottlenecks limiting the software. In addition, during the software run, the garbage collector performed a total of 13248 young generation garbage collections, which took a total of 26.06 seconds. These 13248 garbage collections caused fluctuations in the number of requests in Figure 12. However, since the number of old generation garbage collections is 0, garbage collection does not have a serious impact on consistency. In addition, Figure 12 shows the number of transactions per minute during the six hours. In this table, the throughput in the final phase of the test dropped by about ten percent compared to the beginning. However, since the Java virtual machine has never performed full gc, the reason for the decrease in the number of transactions is the performance degradation caused by the computer running under high stress for a long time. This drop is inevitable and not caused by the software itself

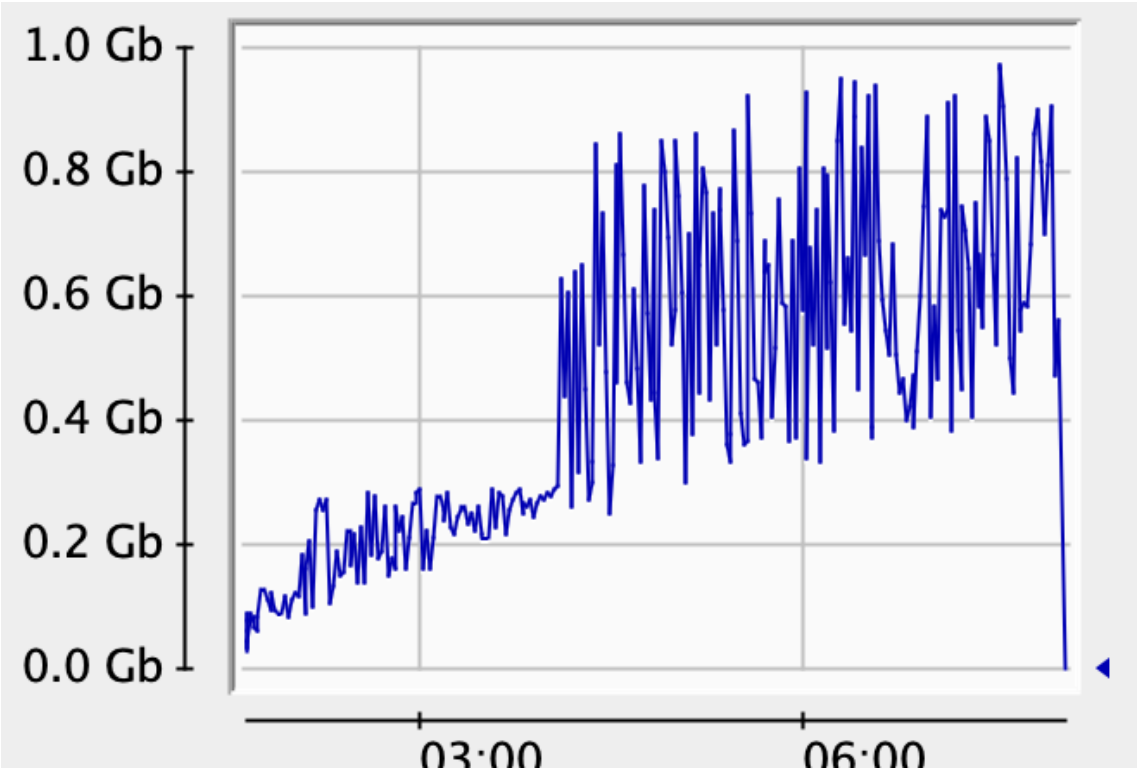


FIGURE 10: Heap space usage

The results of this experiment show that CPU and memory are not the bottlenecks that limit software performance in this experiment. And each performance metrics is relatively stable during operation and does not change significantly over time, so it meets our consistency requirements.

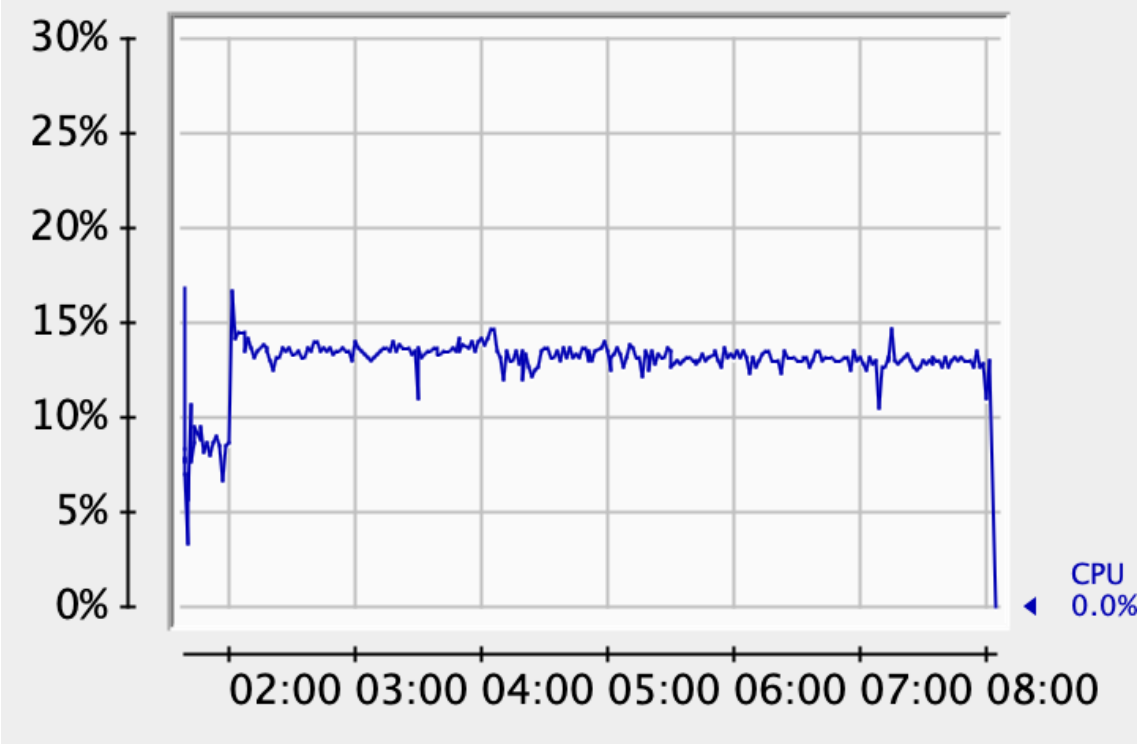


FIGURE 11: Cpu usage

6.3 Requirements Implementation

In the foregoing, five new basic requirements for schema migration benchmarks are presented. These five requirements must be considered when designing the schema migration domain benchmark, however, their implementation depends on the testing tools. Test tools must have the ability to help designers achieve these five requirements. In this chapter, whether the test tool has this capability will be examined.

6.3.1 Continuity

Continuity requires DML requests before and after schema migration to be issued continuously. To verify this requirement, a small experiment was designed. At the 20th second of the experiment, a new column was inserted into the order line table of the database. At the 40th second, both clients of the benchmark were changed at the same time and the DML was changed to the new version. Figure 13 shows the throughput of the database, the blue line and orange line show transactions per second with add a column and no change respectively. The results show that when the worker changes, the number of requests processed by the database does not drop significantly compared to no change. The key to the tool's ability to do this is that it doesn't create new workers and replace old ones. It just replaces the preparedstatements of the transaction executed by the worker that should change. Since the preparedstatements object should have been created when the transaction was executed, changing the DML statement is not disruptive compared to the original benchmark.

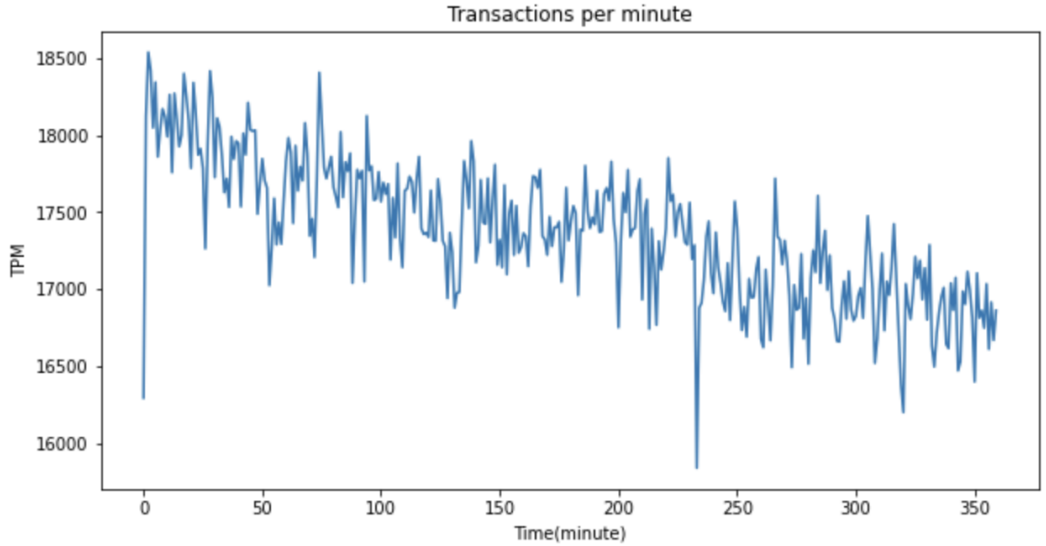


FIGURE 12: Transactions per minute

6.3.2 Inconsistency

Inconsistency requires that sometimes the tool must be able to issue different versions of DML. In the previous chapter, we have explained how tools have this capability. In this chapter, an experiment similar to the previous chapter is designed. Unlike the previous experiments, no schema migration occurred at the twentieth minutes of this experiment. Therefore, the tool will attempt to insert an unknown column of data into the order line table. Since the database will check whether the SQL conforms to the schema, this SQL request that does not conform to the schema will result in a SQLException. After the experiment ended, the software's log showed that no exceptions were recorded during the first forty minutes of the run. After the 40th minute, SQLExceptions are thrown continuously, these exceptions indicate that the tool is trying to insert unknown data named ol_tax into the order line table of the database. A comparison of logs before and after forty minutes of this experiment shows that the tool meets the requirements for inconsistency.

6.3.3 Mixed state

Mixed state means that at some point the database will receive both the DML before and after the schema migration. This ability can be verified by modifying the configuration files used in the experiments in the previous chapter and logging the SQL statements. Now, the two workers are not changed at the 40th second at the same time, but one at the 40th second and one at the 60th second. By observing the logs, we found that there are two different SQLs in the records from 40th seconds to 60th seconds. Therefore, the tool has the ability to achieve mixed states. The reason for this capability is that the tool centrally manages the strings used to generate preparedstatements. Each different worker has an independent such center, so the worker has the ability to generate different preparedstatements.

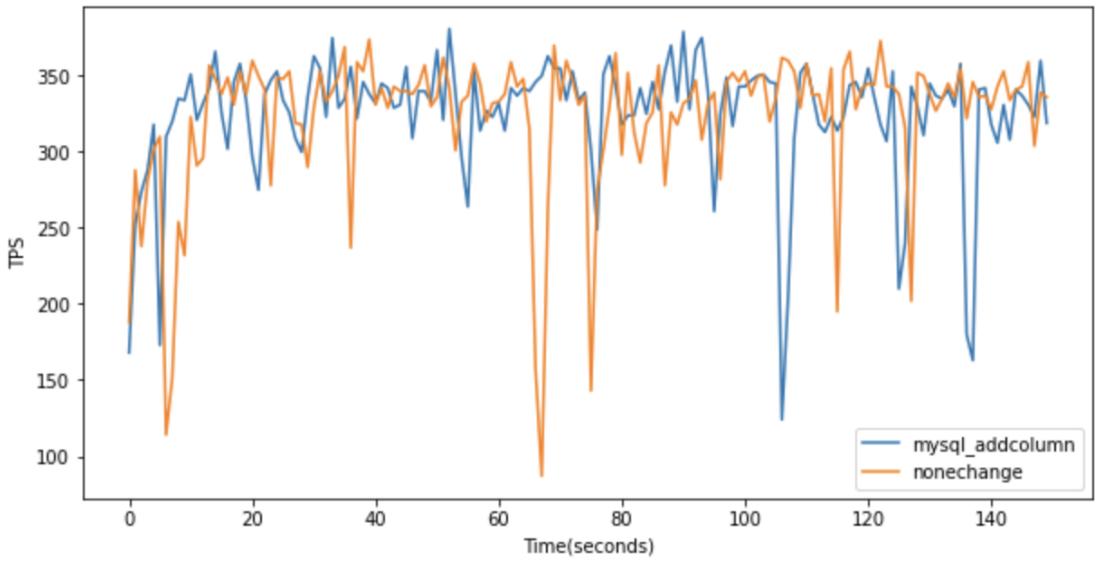


FIGURE 13: Transactions per second with add a column(blue) and no change(orange). Schema migration happens at 20 seconds and workers change at 40 seconds.

6.3.4 Completeness

Currently the tool supports almost all simple changes, although it does not implement many complex changes for the time being. For schema migration that does not affect DML, users can use the "original" benchmark case. Other simple changes also have corresponding base cases. Given the variety of components of complex changes, the tool addresses completeness by providing extensibility. The tool not only makes it easy to add benchmark cases, it also makes it easy to add new benchmarks. Chapter 6.2.3 has given a detailed introduction on how to extend this tool.

6.3.5 Unit

This requirement requires that the requests issued by the same terminal are consistent respectively before and after the change. Experiments in Section 7.3.2 have demonstrated consistency before worker changes. Consistency after worker change can be verified by performing a benchmark case of dropping a column. Execute the script to delete the column before executing the workload, then the SQLException will not be thrown after the 40th second as before the 40th second. The logs after the experiment demonstrate the consistency.

6.4 Testing of Database and Migration Tools

The performance of transfer tools has been showed and compared several times by previous researchers. It does not make much sense to repeat a study for a problem many times, so only simple results are presented in this regard. Since this tool is the first to implement mixed state, experiments will focus on the impact of mixed state on database schema migration and the ability of the migration tool to handle mixed state.
The purpose of the first experiment was to test the ability of the database itself to handle

schema migration. MySQL and Oracle explicitly state in their documentation that support for online schema migration is provided natively or via plugins. As one of the most popular open source database at present, MySQL was chosen as the experimental object of this experiment. Starting from version 5.6, MySQL provides the function of Online DDL. Before that, DDL operations would cause table locks. In this experiment, MySQL 8.0.28 was used. The MySQL instance run on a virtual machine with 128G of NVMe SSD storage and 2G of memory. And the test tool is run on a computer with 8G memory.
First, three experiments that did not involve any types of schema migration were performed, and the experimental results of these experiments will be used as the basis for comparison. The key configuration of the configuration files in these three experiments are shown in Figure 22, 23 and 24. The ratio of the number of warehouses to the number of terminals is usually 4:1 or 5:1, ratios higher than this can sometimes result in a lock table. In order to test the online DDL function of MySQL, the test script we used is shown in Figure 26. In the resulting plots, the time at which the schema transition started is uniformly marked with a red vertical line. The time when the schema migration of different tools ends is marked with a vertical line of different colors.
The orange lines in Figure14, 15 and 16 show the experimental results of onlineDDL. In the experiment with only 1 warehouses and with 10 warehouses , we can find that after the schema migration is completed, the throughput of the database drops to zero and starts to recover gradually until the worker change is completed (black line in 14 and pink line in 15). The reason for this is that the new schema is not able to handle the old version SQL. During the execution of the experiment, the console kept printing SQLException, including the unknown columns or can't find column in field list, etc. After the worker was changed, because the new SQL conformed to the new schema, they could be processed correctly and the throughput gradually recovered. The following experiments with 100 warehouses further verified this conclusion. Due to the increase in the amount of data, the time of schema migration cost increased and finished after the first worker changed, so the situation becomes that the old version schema cannot process the new version SQL, which causes the throughput to drop near the black line. The console also started to continuously print SQLException. However, since there are still workers constantly sending old version SQL, the throughput does not drop to zero as in the previous experiments. It can be concluded from this experiment that MySQL's Online DDL function does not cause the table to be locked and SQL that conform to the schema can still be executed correctly at this time.
The other three experiments were used to test a third-party migration tool designed for MySQL. The migration tool of choice was gh-ost and the results are showed in Figure 14, 15 and 16 with green lines. In this and previous experiments, since Gh-ost does not support foreign key constraints, we used the modified benchmark without foreign key constraints. The original composite primary key is deleted, the new primary key is an auto-incrementing column and the original primary key is substitute with a composite index to increase query speed. The changes in throughput in these graphs show the same trend as the experiments of onlineDDL, SQLException is constantly being printed to the console during execution. The result of the experiment with 100 warehouses was slightly different because Gh-ost did not complete the migration during the 1 hour experiment. Therefore, all SQL is new version that cannot be processed by the old schema. During the experiment, Gh-ost showed that it was expected to take more than 12 hours to complete the migration. In the end, the experiment was not completed. At the same time, the constantly printed SQLException also shows that Gh-ost also does not have the ability to handle multi-version SQL. Finally, the time taken by each migration tool to complete the migration is shown in Table 9.

util	online ddl			gh-ost		
number of warehouses	1	10	100	1	10	100
number of terminals	1	2	20	1	2	20
migration time(seconds)	4	51	2202	72	284	not finished in 1 hour

TABLE 9: Migration time

In order to test the performance of Online DDL and Gh-ost without exceptions, we designed another experiment using the add a column benchcase and make the schema migration complete before the worker is changed, so that the old and new version SQL conform to the schema before and after migration. Figure 17 shows the results of this experiment. The green and orange lines have remained stable and close to the blue line throughout the experiment. By comparison, it can be found that MySQL's Online DDL function does not have a great impact on performance and can migrate the schema in a pretty short time. Gh-ost took four times as long as OnlineDDL to complete the schema migration. And Gh-ost also did not have a significant impact on the throughput of the database after the begin of schema migration until the workers started changing. Based on the above experimental results, the MySQL database itself has a good ability to complete the online schema migration and the impact on the throughput is small when the online schema migration occurs. However, the multi-version SQL caused by the mixed state may generate exceptions and cause the transaction to temporarily fail to complete, affecting the ability of the database to process transactions.

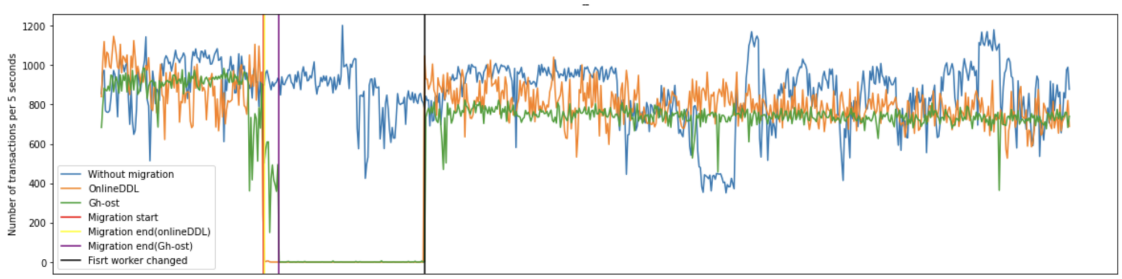


FIGURE 14: Number of transactions per 5 seconds with migration using OnlineDDL, using gh-ost and without any migration with 1 warehouse and 1 terminal

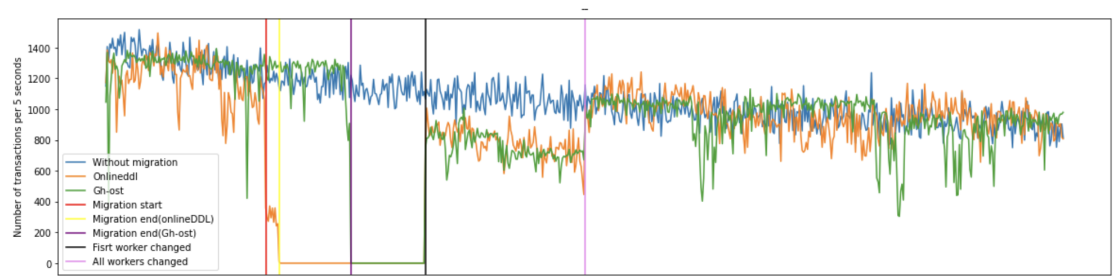


FIGURE 15: Number of transactions per 5 seconds with migration using OnlineDDL, using gh-ost and without any migration with 10 warehouses and 2 terminals

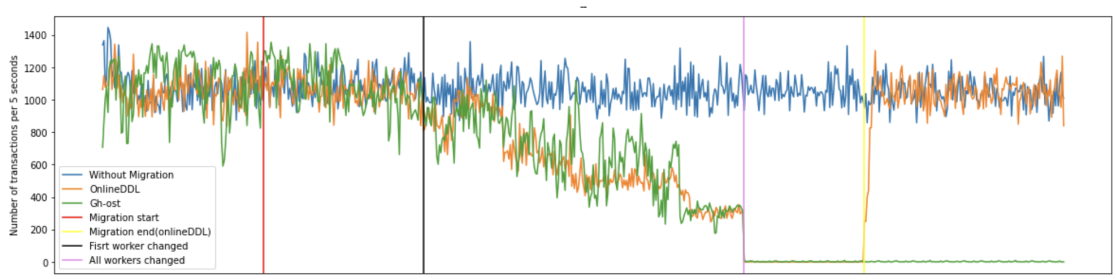


FIGURE 16: Number of transactions per 5 seconds with migration using OnlineDDL, using gh-ost and without any migration with 100 warehouses and 20 terminals

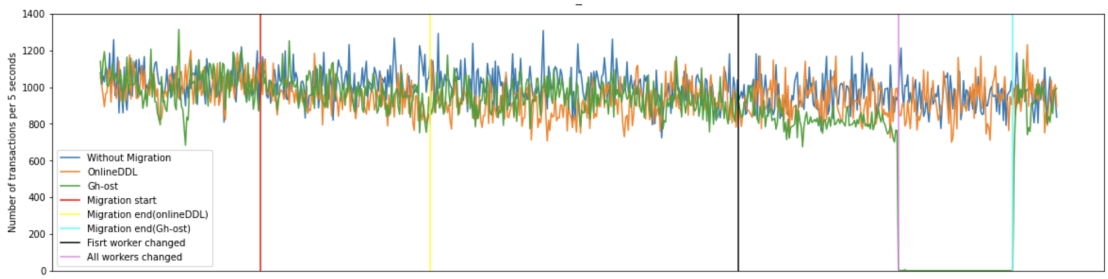


FIGURE 17: Result of add a column experiment with 50 warehouses and 10 terminals

There are not many online migration tools that claim to be capable of handling multiple versions of SQL, Quantumdb is one of them. Quantumdb is an online schema migration tool designed for PostgreSQL. It ensures data synchronization through triggers, supports foreign key constraints, and has the ability to handle multiple versions of SQL and mixed state. The version it was developed with is PostgreSQL 11. The version used in this experiment was 13, but this did not cause any exceptions or errors. Three experiments with similar configurations to the previous experiments were used for comparison. But the maximum number of terminals is limited to 10, and the warehouse is 40. The changesets used in the experiment are also shown in Figure 25. Since this testing tool was not originally designed for any migration tools, testing Quantumdb resulted in some minor modifications to the software. The principle of Quantumdb dealing with mixed state is to rewrite the SQL through the version field in the parameter by using a custom database driver. This requires that when the worker changes and reconnects to the database, the url used to generate the database connection should also change. Another change is that Quantumdb uses a custom driver which has not been published to the maven center, so users must add it to dependencies and load the driver themselves. For the experimental results, we marked the time when the migration started and ended with the red and green lines respectively. At the same time, in order to better show the experimental results, we use the number of transactions every five seconds as the y-axis.
The experimental results are presented in Figure 18 , 19 and 20. The orange line shows the throughput with schema migration using QuamtumDB, and the blue line is the database throughput without schema migration. During the experiment, the orange line never dropped to 0 but was below the blue line most of the time. The experimental results show that the changes in database throughput are relatively stable during the schema migration process and no exceptions are printed in the console, although the throughput is

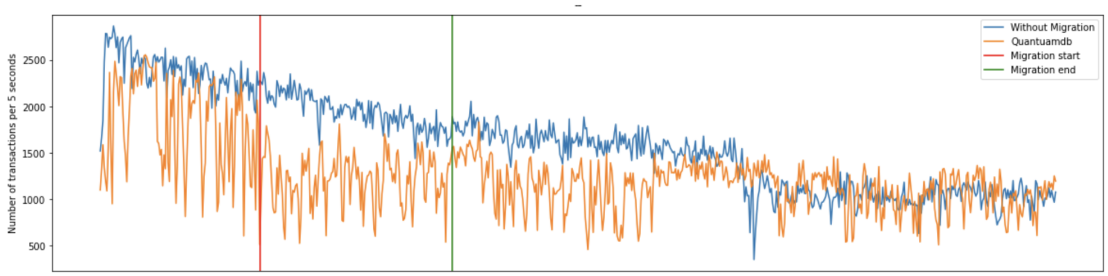


FIGURE 18: Number of transactions per 5 seconds with migration using Quantumdb and without any migration with 1 warehouse and 1 terminal

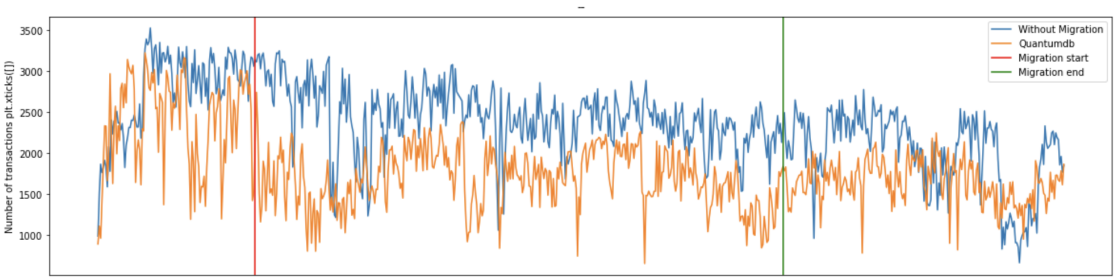


FIGURE 19: Number of transactions per 5 seconds with migration using Quantumdb and without any migration with 10 warehouse and 2 terminal

reduced compared to not performing any schema migration. The results of this experiment demonstrate that Quantumdb can maintain multiple versions of data and can handle mixed states, but this capability comes with a performance penalty. In addition, compared to Online DDL, Quantumdb still takes a long time to complete the migration.

The above content mainly analyzes the throughput during the experiment. However, in addition to recording the throughput of the database, this tool can also record the latency, the number of different transactions and some other indicators of the database. Since these metrics are less affected by pattern transfer, they are not analyzed.

6.5 Unsolved Problems

During the experiment, some unresolved problems also occurred, but these problems did not have a decisive impact on the results of the experiment. In this chapter, we will introduce these issues. The cause of the first problem is MySQL or gh-ost. In the process

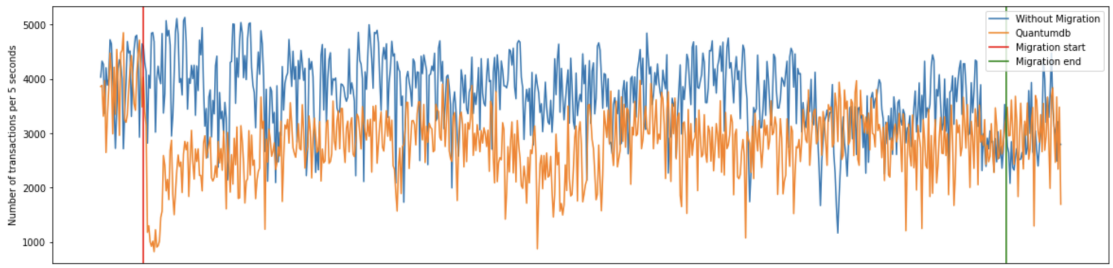


FIGURE 20: Number of transactions per 5 seconds with migration using Quantumdb and without any migration with 40 warehouse and 10 terminal

of a experiment, the machine running the database prompts that the storage space is full. However, it should have been able to store this amount of data. During the inspection process, it was not possible to locate which file caused this situation by using the scandisk tool. The issue was resolved after uninstalling MySQL and reinstalling. This problem should have arisen by accident as there is no previous description of this problem on the web and it has not recurred. The cause of this problem remains to be explored.
The other two problems are caused by Quantumdb. The Quantumdb-provided driver causes a reduction in throughput compared to using the Maven Center-provided driver even without performing any schema migration. This will cause a roughly 20% performance drop. During the migration process using Quantumdb, we observed that sometimes the test tool will stay in the stage of waiting for the terminal to end for a long time and the migration process of Quantumdb will also last for a long time, or even never end. This problem did not occur when testing other schema migration tools. At the same time, this problem also does not arise when the driver provided by Quantumdb is not used when it does no need to deal with mixed state. Therefore, it is initially inferred that the problem is caused by the driver. However, further diagnosis of the problem could not proceed due to lack of more detailed information.

7 Discussion

7.1 Result Analysis

In this chapter, we will further analyze the results of previous experiments and experiments and conduct further analysis of this benchmarking tool.
The results of the unit tests show that the tool's functions and modules function as expected. However, the specific performance of the tool still requires more realistic and persistent testing. Continued testing of up to six hours has demonstrated that the tool's performance remains stable over extended periods of time, albeit with a drop in the number of requests due to the performance degradation of the machine running the software. During these six hours, the Java Virtual Machine did not experience full gc, nor did the heap memory usage exceed the settings. These experimental results show that the tool has the ability to benchmark over a longer period of time.
Subsequent experiments verified that the tool satisfies the new requirements. Compared to traditional benchmarking tools, this tool provides the ability to benchmarking schema migrations. Unlike traditional benchmarking tools, the query of this tool is variable during the execution of the workload. This capability allows it to be used for benchmarking schema migration. More than just comparing with traditional benchmarking tools, the tool also has unique advantages over other testing tools designed or used by researchers in the field of schema migration. Some previous researchers have used the ability of benchmarking tools to execute multiple benchmarks consecutively to simulate query changes during schema migration. However, this approach poses some problems. The first is that this method has poor extensibility, users need to implement at least two benchmarks. While parts of the code can be reused by copy-pasting, this again makes it difficult to modify the underlying benchmark. Users need to find what needs to be modified in each derived benchmark and modify them. This tool avoids this problem. On the one hand, code from the base benchmark can be reused by derived benchmarks. On the other hand, the user only needs to modify the code in the base benchmark instead of searching anywhere. In addition to this, the tool also provides the ability to gradually change the worker's query. The previous tools could only change all the workers at once, instead of making individual settings for each worker like this tool. Using this tool, users can make changes to workers gradually. For example, there are 3 workers in total, and the user can set them to change at 5, 10, and 15 minutes after the benchmark is executed. This capability also helps the tool achieve the mixed state, where multiple versions of a query can be issued at the same time. The tool is also currently the only benchmarking tool that implements the mixed state. Finally, the tool already provides multiple schema migration benchmarks. These benchmarks cover all simple changes and some commonly used complex changes. If these benchmarks cannot meet the needs of users, its good extensibility also provides convenience for users to customize their own schema migration benchmarks.
Finally, some experiments are also conducted to test some schema migration tools and observe the performance of the tool in actual testing. The result shows that Online DDL and Gh-ost do not have the ability to store multi-version data and handle multi-version SQL while Quantumdb has. For the former, although DML is not blocked during schema migration, the query is rejected and an exception is thrown because the new version SQL does not conform to the old schema. The practical application of the tool performed well, although some unsolved problems were found, but it was probably caused by the driver provided by Quantumdb.
In short, this tool is the first schema migration benchmarking tool that implements mixed state. Compared to other tools, it provides benchmarks covering multiple schema migra-

tions and provides good extensibility. In practical applications, the tool has also achieved good performance. The results of various tests verify that the tool meets the requirements.

7.2 Future Work

7.2.1 More schema migration benchmarks

This tool not only provides good extensibility at the benchmark case level, but also provides good extensibility for benchmarks. Users can easily add other benchmarks in the field of schema migration to the tool according to their own wishes. In the process of writing this tool, we also found some flaws in the TPC-C benchmark, such as the lack of statements to query data by non-primary keys, which led to insufficient testing of non-primary key indexes. Future work can further design benchmarks applied in the field of schema transfer, and the good extensibility of this tool facilitates the implementation of benchmarks.

7.2.2 More schema migration datasets

Currently, the only publicly available dataset that contains schema information for different versions of the database is wikipedia[61] . Richter also gave a statistics on changesets of ING Bank[50] . However, these are far from enough. Sufficient and detailed data on schema migrations that have occurred in production environments can help designers better design benchmarks in the schema migration field. During the design process, which kind of schema migration is commonly used becomes a difficult problem, especially for complex schema migration. Finally, we chose to solve this problem by providing good extensibility. A problem with all current datasets is that the descriptions are not detailed enough, they often only tell you which schema migration happened how many times over a period of time, but not which DDLs are executed together. If there are many companies in the future that can provide data organized in a format like Liquibase's changeset, it will greatly help the design of the benchmark.

7.2.3 Implementation of more complex changes

A major difficulty in implementing complex changes is the need to make changes to the original execution flow of TPC-C. The current architecture of the tool does not support this feature. However, there are two ways to achieve this with this tool. The first is further development of the tool, which abstracts the process of execution at a higher level. This implementation is difficult and requires deep thinking about the software architecture. The second implementation is to treat the benchmark with the new execution flow as a new benchmark and add it to the tool. This way is easier, but makes the software bloated with many such "new" benchmarks. We believe that the second solution is just a stopgap measure. The first solution is the ultimate solution to this problem. However, how to implement the first method is still under consideration.

7.2.4 Optimization of OLTPBenchmark

Although OLTPBenchmark meets our requirements quite well as a benchmarking tool, it still has some flaws. When testing databases that are not local, the loading process tends to take a long time. By comparison, we found that the time taken to perform the load locally is much less than that of the remote. Further modularization of the software might be a good solution, modularizing the loading process and executing it on the machine

that will be tested. In the actual test, we also execute the loading process on the tested machine, which greatly saves time.

7.2.5 Continuous Schema Migration

Currently the tool only supports one schema migration. The reason for this result is not a limitation of software functionality, but a limitation of the benchmark design. This tool provides the ability to execute multiple benchmarks consecutively. However, since the schema of the database has changed after the schema migration and all migrations are performed on the schema before the change, this tool is temporarily unable to perform benchmarks with more than two consecutive schema migrations. Future work could look at more sophisticated design of the benchmark so that it can be executed continuously.

7.2.6 Support for migration tools

Currently, the main function of this tool is to benchmark data and collect and analyze test results. As mentioned above, users often need to choose a schema migration tool when performing schema migration. This software can help users to choose by integrating some schema migration tools, which brings convenience to users. To achieve this functionality requires scripting or implementing configuration classes for different migration tools. In addition, we also hope to have a migration tool that can support multiple databases. Almost all migration tools currently only support a single database, which limits the software to a certain extent.

7.2.7 Real Multi-user test

OLTPBenchmark can simulate multiple users accessing the database, however, this is a simulation rather than real multiple users accessing the database through the network at the same time. Executing multi-user tests can better simulate actual application scenarios and make test results more accurate. A simple implementation of multi-user emulation is to compile and execute the tool on multiple machines, and then send requests to the database. A more rigorous multi-user simulation would require an additional coordination center, which would complicate the management of the machines sending the requests and the statistics of the test results. At present, the tool has not tried how to achieve this function.

7.2.8 Strict consistency testing

At present, the tool does not test strictly for database consistency.The tool judges whether the database has executed the query correctly by the return value of the execution result of the database. Strict consistency testing should check the results returned by database query statements to determine whether update and insert statements are executed correctly. This requires a detailed record of the current data in the database. A possible solution is to log the results returned by all database and query statements generated by the benchmark. And after the workload is executed, all the returned results are checked through a script. Checking in real-time can impact database performance, while deferred checking minimizes this impact.

8 Conclusion

This section summarizes the contribution to science and the main work of this thesis. And at the end of this section, the previously raised research questions will be listed and answered.

8.1 Contribution and Main Work

The main purpose of this thesis is to design a benchmark and a testing tool for database schema migration. In the process of designing and implementing this tool, this thesis also made other contributions besides the tool itself. First, in the literature review chapter, the researches in the field of schema migration are reviewed and summarized. In addition to this, the section also summarizes the benchmark's design guidelines, metrics and benchmarks for schema migration. In the design phase of the benchmark and the tool, we first summarizes the benchmark's evaluation criteria and proposes new requirements for the schema migration benchmark. This chapter also lists the differences between schema migration benchmarks and database benchmarks and proposes how to modify the traditional benchmark indicators accordingly. Finally, the tool is currently the only benchmarking tool that implements mixed state, and perhaps the first benchmarking tool for schema migration.

The design of the benchmark is divided into two phases, the task of the first phase is to design the workload. At this stage, multiple criteria from other benchmark documents or related researches are listed. Based on these criteria, we choose to design a new benchmark for schema migration based on the TPC-C benchmark. In addition, we also listed and analyzed multiple benchmarking tools and ranked these tools according to the needs to select the most suitable tools for secondary development. The main task of the second stage is to design the benchmark indicators. This thesis first summarizes the differences in schema migration benchmarks and points out that these different traditional metrics cannot measure performance well. Therefore, new evaluation methods are proposed.

During the implementation phase, the architecture of OLTPBenchmark is modified. This modified architecture allows the SQL to vary according to the settings. In addition to this, the mixed state mentioned in many papers has also been implemented for the first time. Finally, this architecture also provides another level of extensibility for the tool.

Through the analysis of the experimental results, we concluded that although these tools will affect the performance to a certain extent, there is no significant difference in the impact of these tools on the database performance. Also of concern is their ability to handle multiple versions of data, as this ability directly affects migration planning. For tools without this capability, the timing of schema migration and the timing of backend program updates should be carefully considered, and the SQL used should be carefully designed. For a tool with this capability, consider whether the way it implements this capability might cause other problems for the application. In addition to this, this thesis demonstrates experimentally the problems that mixed states can cause, and points out that designers of schema migration tools should consider these issues.

Finally, this thesis also mentions the shortcomings of existing work and research in the future work chapter. The first is the inadequacy of this tool. It should further abstract the transaction to provide better extensibility and provide more complex changes. The second is the lack of data, more datasets of schema migration from the real world are very necessary.

8.2 Answers to the Research Questions

The research questions of this thesis are:

1. Which of the existing benchmarking tools can be more easily re-developed to adapt to the scenario of schema change?
    - What criteria should be met when selecting tools for secondary development?
    - How should the tool be re-developed to make it meet the demand?
2. In view of the database schema changes, what requirements must a benchmark and benchmarking tool meet?
    - What requirements can we get from the existing benchmarks and benchmarking tools?
    - What new requirements can be found in case of schema migration?
3. In case of schema migrations, what metric will allow us to better measure the performance of the database?
4. How the quality of a benchmark can be measured?

8.2.1 Research question1

There is currently no benchmarking tool designed for schema migration. Researchers in the field of schema migration tend to benchmark their designed tools by simply modifying existing benchmarking tools based on their own needs. Such modifications are usually relatively simple, not extensible and do not fully meet the requirements in Section 4.2.3. However, the analysis in Section 4.2.1 shows that some existing tools have the potential to be used to benchmark schema migration after further development. Compared with developing a tool from scratch, secondary development can save time and human resources. Therefore, this thesis designed an evaluation system to scores the existing tools and selects the most suitable tools for further development according to the scores. The evaluation system focuses on the software's documentation integrity, extensibility and the friendliness of the programming language used, etc. All these properties are helpful for further development of the software. The scores of each tool are shown in Table 5. In this comparison, OLTPBenchmark got the highest total score. And it also scored very well in each of the individual items. Thus, OLTPBenchmark was selected as the most suitable benchmarking tool for further development. However, it still requires some modifications before it can be used to benchmark schema migrations. For example, the SQL of the tool's benchmark is unchanged during the execution of the workload, which does not meet our requirements in Section 4.2.3. Other modifications to meet the requirements are listed in Sections 5.2 and 6.3. The fifth chapter mainly introduces the implementation details, and the seventh chapter analyzes whether the tool meets the requirements.

8.2.2 Research question2

Determining requirements is a very important step in the design science circle, and identifying the test objects are also an important part of designing benchmarks. This research question helped us determine how a benchmark and benchmarking tool should be measured and what modifications should be made to them before they can be used to test schema migrations. Some of the criteria is taken from the documentation of traditional database

benchmarks or from benchmark-related researches. These criteria are enumerated in Section 4.1.1 and later help us choose a benchmark on which to base. In addition to this, some new requirements for schema migration benchmarks are proposed, which are drawn from researches in the field of schema migration and summarized in Section 4.2.3. The new criteria such as continually, inconsistency, mixed state and completeness guide how we should modify the traditional database benchmarks and tell us some of the functions that the tool must have. In the implementation phase, due to the diversity of schema migrations, it is not realistic to design a scenario and benchmark that includes all schema migrations or to implement a benchmark for each schema migration. Since no one knows their needs better than themselves, we solve this problem by providing users with good extensibility. However, multiple benchmarks that model simple or complex changes are still provided.

8.2.3 Research question 3

Metrics are another important component of benchmark and different metrics are needed to test different systems. Section 2.4.2 summarizes and enumerates the metrics used by previous database benchmarks. However, as one of the subdivisions of database benchmarks, the metrics for schema migration benchmarks should be somewhat different from database benchmarks. In Section 4.3, we analyze this difference, stating that the metrics like throughout, latency and migration cost for schema migration benchmarks should focus more on demonstrating the process, and the line graph provides a better presentation. In addition, the criteria of the schema migration tool also have reference significance for the design of indicators and some indicators are also obtained from the criteria. Finally, in this chapter two important time points are identified and three formulas used to measure the performance penalty are defined.

8.2.4 Research question 4

In section 6, the quality of the design proposal is evaluated. The results of unit testing of the software demonstrate that the core components of the software behave as expected. And The results of performance tests on the software show that the software itself does not affect throughput when running for a long time. Logs of experiments conducted in Section 6.3 demonstrate that the software meets the five newly proposed requirements for schema migration benchmarking tools. In order to test the software more practically, we also tested the database and schema migration tools with the software. The results of the experiments are presented and discussed in Sections 6.4 and 7.1, respectively. During experiments, some unsolved problems were discovered and they are enumerated in Section 6.5.

References

[1] Facebook online schema change. `https://github.com/facebookincubator/OnlineSchemaChange`, 2010.

[2] Oak online alter table. `https://shlomi-noach.github.io/openarkkit/oak-online-alter-table.html`, 2010.

[3] Large hadron migrator. `https://github.com/soundcloud/lhml`, 2012.

[4] Pt online schema change. `https://www.percona.com/doc/percona-toolkit/3.0/pt-online-schema-change.html`, 2016.

[5] Timothy G Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. Linkbench: a database benchmark based on the facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1185–1196, 2013.

[6] Charles W Bachman. » integrated data store–the information processing machine that we need!«. *Charles W. Bachman Papers (CBI 125), Box*, 1, 1962.

[7] Philip A Bernstein and Nathan Goodman. Multiversion concurrency control—theory and algorithms. *ACM Transactions on Database Systems (TODS)*, 8(4):465–483, 1983.

[8] Paul Beynon-Davies. *Database systems*. Bloomsbury Publishing, 2017.

[9] Souvik Bhattacherjee, Gang Liao, Michael Hicks, and Daniel J Abadi. Bullfrog: Online schema evolution via lazy evaluation. In *Proceedings of the 2021 International Conference on Management of Data*, pages 194–206, 2021.

[10] Dina Bitton, David J DeWitt, and Carolyn Turbyfill. Benchmarking database systems-a systematic approach. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1983.

[11] Dina Bitton and Carolyn Turbyfill. Design and analysis of multi-user benchmarks for database systems. Technical report, Cornell University, 1984.

[12] Dina Bitton and Carolyn Turbyfill. *A Retrospective on the Wisconsin Benchmark*, page 422–441. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[13] Cristiana Bolchini, Fabio Salice, Fabio A Schreiber, and Letizia Tanca. Logical and physical design issues for smart card databases. *ACM Transactions on Information Systems (TOIS)*, 21(3):254–285, 2003.

[14] Haran Boral and David J DeWitt. A methodology for database system performance evaluation. *SIGMOD Rec.*, 14(2):176–185, jun 1984.

[15] Haran Boral and David J DeWitt. A methodology for database system performance evaluation. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, page 176–185, New York, NY, USA, 1984. Association for Computing Machinery.

[16] Alina Buzachis, Antonino Galletta, Antonio Celesti, Lorenzo Carnevale, and Massimo Villari. Towards osmotic computing: a blue-green strategy for the fast re-deployment of microservices. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2019.

[17] Michael J Cahill, Uwe Röhm, and Alan D Fekete. Serializable isolation for snapshot databases. *ACM Transactions on Database Systems (TODS)*, 34(4):1–42, 2009.

[18] RGG Cat Te Ll. The engineering database benchmark. 1991.

[19] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna Gummadi. Measuring user influence in twitter: The million follower fallacy. In *Proceedings of the international AAAI conference on web and social media*, volume 4, 2010.

[20] DBTG CODASYL. Codasyl data base task group report, conf. *Data Sys. Languages, ACM, New York*, 1971.

[21] Edgar F Codd. A data base sublanguage founded on the relational calculus. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) workshop on data description, access and control*, pages 35–68, 1971.

[22] Edgar F Codd. Further normalization of the data base relational model. *Data base systems*, 6:33–64, 1972.

[23] Edgar F Codd et al. *Relational completeness of data base sublanguages.* Citeseer, 1972.

[24] CODASYL Development Committee et al. An information algebra-phase i report. *Comm. ACM*, 5(4):190–204, 1962.

[25] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[26] Carlo A Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Update rewriting and integrity constraint maintenance in a schema evolution support system: Prism++. *Proceedings of the VLDB Endowment*, 4(2):117–128, 2010.

[27] Peter Dadam and Jukka Teuhola. Managing schema versions in a time-versioned non-first-normal-form relational database. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 161–179. Springer, 1987.

[28] Michael de Jong, Arie van Deursen, and Anthony Cleve. Zero-downtime sql database schema evolution for continuous deployment. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 143–152. IEEE, 2017.

[29] Michael de Jong, Arie van Deursen, and Anthony Cleve. Zero-downtime sql database schema evolution for continuous deployment. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 143–152, 2017.

[30] Xin Luna Dong and Divesh Srivastava. Big data integration. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 1245–1248. IEEE, 2013.

[31] Anon et al, Dina Bitton, Mark Brown, Rick Catell, Stefano Ceri, Tim Chou, Dave DeWitt, Dieter Gawlick, Hector Garcia-Molina, Bob Good, Jim Gray, Pete Homan, Bob Jolls, Tony Lukes, Ed Lazowska, John Nauman, Mike Pong, Alfred Spector, Kent Trieber, Harald Sammer, Omri Serlin, Mike Stonebraker, Andreas Reuter, and Peter Weinberger. A measure of transaction processing power. *Datamation*, 31(7):112–118, apr 1985.

[32] Gordon C Everest and Edgar H Sibley. Critique of the guide-share dbms requirements. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 93–112, 1971.

[33] Fabrizio Ferrandina, Thorsten Meyer, Roberto Zicari, Guy Ferran, and Joëlle Madec. Schema and database evolution in the o˜ 2 object database system. In *VLDB*, volume 95, pages 170–181. Citeseer, 1995.

[34] Jim Gray. Database and transaction processing performance handbook., 1993.

[35] J-L Hainaut, Vincent Englebert, Jean Henrard, J-M Hick, and Didier Roland. Database evolution: the db-main approach. In *International Conference on Conceptual Modeling*, pages 112–131. Springer, 1994.

[36] Emily H Halili. *Apache JMeter*. Packt Publishing Birmingham, 2008.

[37] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[38] Slađana Janković, Snežana Mladenović, Stefan Zdravković, and Ana Uzelac. Schema on read modeling approach implementation in big data analytics in traffic.

[39] Harpreet Kaur and Gagan Gupta. Comparative study of automated testing tools: selenium, quick test professional and testcomplete. *Int. Journal of Engineering Research and Applications*, 3(5):1739–1743, 2013.

[40] Won Kim and Hong-Tai Chou. Versions of schema for object-oriented databases. In *Proceedings of the 14th international conference on very large data bases*, pages 148–159, 1988.

[41] Barbara Staudt Lerner and A Nico Habermann. Beyond schema evolution to database reorganization. *ACM SIGPLAN Notices*, 25(10):67–76, 1990.

[42] Andy Maule, Wolfgang Emmerich, and David S Rosenblum. Impact analysis of database schema changes. In *Proceedings of the 30th international conference on Software engineering*, pages 451–460, 2008.

[43] Edwin McKenzie and Richard Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[44] Mark Lukas Möller, Stefanie Scherzinger, Meike Klettke, and Uta Störl. Why it is time for yet another schema evolution benchmark. In *International Conference on Advanced Information Systems Engineering*, pages 113–125. Springer, 2020.

[45] Iulian Neamtiu, J. Bardin, R. Uddin, Dien-Yen Lin, and P. Bhattacharya. Improving cloud availability with on-the-fly schema updates. In *COMAD*, 2013.

[46] Neal Nelson. The neal nelson database benchmarktm: A benchmark based on the realities of business., 1991.

[47] Shlomi Noach. Openark kit. *common utilities for MySQL*, 2015.

[48] Zachary Parker, Scott Poe, and Susan V Vrbsky. Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference*, pages 1–6, 2013.

[49] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek. Online, asynchronous schema change in f1. *Proceedings of the VLDB Endowment*, 6(11):1045–1056, 2013.

[50] Nick Geral Richter. Zero-downtime postgresql database schema migrations in a continuous deployment environment at ing. Master's thesis, University of Twente, 2021.

[51] John F. Roddick. Schema evolution in database systems - an annotated bibliography. *SIGMOD record*, 21(4):35–40, 1992.

[52] Mikael Ronstrom. On-line schema update for a telecom database. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*, pages 329–338. IEEE, 2000.

[53] Tom Sawyer. Doing your own benchmark. In *The Benchmark Handbook*, 1991.

[54] S Shaw. Hammerdb: the open source oracle load test tool, 2012.

[55] Yangjun Sheng. *Non-blocking Lazy Schema Changes in Multi-Version Database Management Systems*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2019.

[56] Gary H Sockut and Robert P Goldberg. Database reorganization—principles and practice. *ACM Computing Surveys (CSUR)*, 11(4):371–395, 1979.

[57] Gary H Sockut and Balakrishna R Iyer. Online reorganization of databases. *ACM Computing Surveys (CSUR)*, 41(3):1–136, 2009.

[58] Standard Specification and Transaction Processing Performance Council TPC. Tpc benchmark tm h. 1993.

[59] Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In *Proceedings of the 1974 annual conference-Volume 1*, pages 180–186, 1974.

[60] Carolyn Turbyfill, Cyril U Orji, and Dina Bitton. As3ap: An ansi sql standard scaleable and portable benchmark for relational database systems., 1993.

[61] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845, 2009.

[62] Yuepeng Wang, James Dong, Rushi Shah, and Isil Dillig. Synthesizing database programs for schema refactoring. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–300, 2019.

[63] Lesley Wevers, Marieke Huisman, and Maurice van Keulen. Lazy evaluation for concurrent oltp and bulk transactions. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, pages 115–124, 2016.

[64] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[65] A Wolski. Tatp benchmark description (version 1.0), 2009.

[66] Yu Zhu. *Towards Automated Online Schema Evolution*. University of California, Berkeley, 2017.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
  <isolation>TRANSACTION_SERIALIZABLE</isolation>
  <batchsize>128</batchsize>
  <scalefactor>100</scalefactor>
  <terminals>20</terminals>
  <works>
    <work>
      <time>21600</time>
      <rate>10000</rate>
      <weights>45</weights>
      <weights>43</weights>
      <weights>4</weights>
      <weights>4</weights>
      <weights>4</weights>
    </work>
  </works>
  <migration>
    <benchcase>original</benchcase>
    <util>none</util>
    <timepoint>60</timepoint>
    <number>20</number>
```

FIGURE 21: Configuration 1

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
  <isolation>TRANSACTION_SERIALIZABLE</isolation>
  <batchsize>128</batchsize>
  <scalefactor>1</scalefactor>
  <terminals>1</terminals>
  <works>
    <work>
      <time>3600</time>
      <rate>10000</rate>
      <weights>45</weights>
      <weights>43</weights>
      <weights>4</weights>
      <weights>4</weights>
      <weights>4</weights>
    </work>
  </works>
```

FIGURE 22: Configuration 2

```xml
<configuration>
  <isolation>TRANSACTION_SERIALIZABLE</isolation>
  <batchsize>128</batchsize>
  <scalefactor>10</scalefactor>
  <terminals>2</terminals>
  <works>
    <work>
      <time>3600</time>
      <rate>10000</rate>
      <weights>45</weights>
      <weights>43</weights>
      <weights>4</weights>
      <weights>4</weights>
      <weights>4</weights>
    </work>
  </works>
```

FIGURE 23: Configuration 3

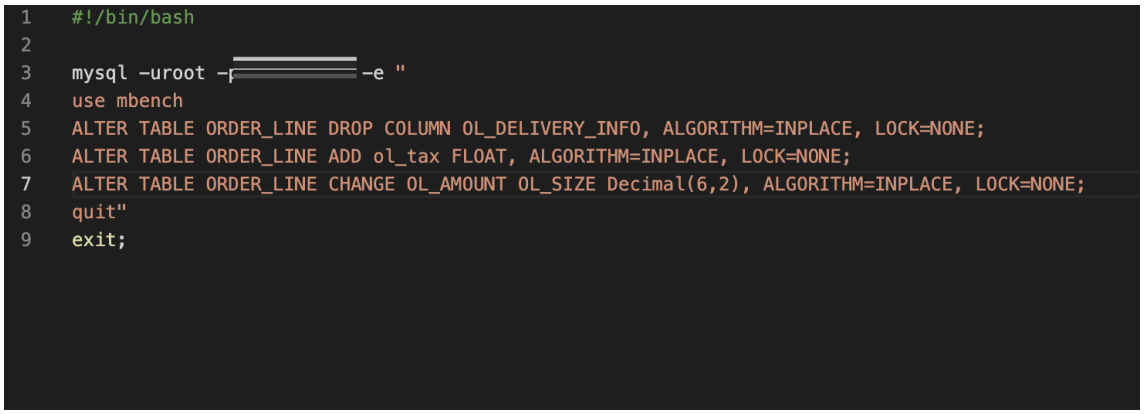
```
<configuration>
  <isolation>TRANSACTION_SERIALIZABLE</isolation>
  <batchsize>128</batchsize>
  <scalefactor>100</scalefactor>
  <terminals>20</terminals>
  <works>
    <work>
      <time>3600</time>
      <rate>10000</rate>
      <weights>45</weights>
      <weights>43</weights>
      <weights>4</weights>
      <weights>4</weights>
      <weights>4</weights>
    </work>
  </works>
```

FIGURE 24: Configuration 4

```
<?xml version="1.0" encoding="UTF-8"?>
<changelog xmlns="http://www.quantumdb.io/xml/ns/quantumdb-changelog"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.quantumdb.io/xml/ns/quantumdb-changelog-0.4.xsd">

  <changeset id="add_ol_amount" author="Shuhao Li">
    <description>add a column.</description>
    <operations>
      <addColumn tableName="order_line">
        <column name="ol_tax" type="real" nullable="true" />
      </addColumn>
      <alterColumn tableName="order_line" columnName="ol_amout" newColumnName="ol_size" nullable="true" />
      <dropColumn tableName="order_line" columnName="ol_delivery_info" />
    </operations>
  </changeset>
</changelog>
```

FIGURE 25: Changeset used by quantumdb



```
#!/bin/bash

mysql -uroot -p[redacted] -e "
use mbench
ALTER TABLE ORDER_LINE DROP COLUMN OL_DELIVERY_INFO, ALGORITHM=INPLACE, LOCK=NONE;
ALTER TABLE ORDER_LINE ADD ol_tax FLOAT, ALGORITHM=INPLACE, LOCK=NONE;
ALTER TABLE ORDER_LINE CHANGE OL_AMOUNT OL_SIZE Decimal(6,2), ALGORITHM=INPLACE, LOCK=NONE;
quit"
exit;
```

FIGURE 26: Script used for online ddl