

南京信息工程大学

本科生毕业论文（设计）



题 目 MySQL 管理优化工具的设计与实现

学生姓名	<u>邵震</u>
学 号	<u>20161307022</u>
学 院	<u>管理工程学院</u>
专 业	<u>信息管理与信息系统</u>
指导教师	<u>李敏</u>

二〇二〇 年 五 月 十 七 日

声 明

本人郑重声明：

- 1、 以“求实、创新”的科学精神从事科学研究工作。
- 2、 本论文中除引文外，所有测试、数据和相关材料均为真实有效的。
- 3、 本论文是我个人在指导教师的指导下进行的研究工作和取得的研究成果，请勿用于非法用途。
- 4、 本论文中除引文和致谢的内容外，并未抄袭其他人或其他机构已经发表或撰写过的研究成果。
- 5、 关于其他同志对本研究所做的贡献均已在论文中作了声明并表示了谢意。

作者签名：邵震

日期：2020年4月25日

目 录

1. 概述	1
1.1 选题背景	1
1.2 发展现状	2
1.2.1 国外发展现状	2
1.2.2 国内发展现状	2
1.2.3 存在问题	2
1.3 目的及意义	3
2. MySQL 管理优化系统规划	3
2.1 系统开发环境	3
2.2 MySQL 管理优化工具开发可行性分析	3
2.2.1 管理可行性	3
2.2.2 技术可行性	3
2.2.3 经济可行性	4
3. MySQL 管理优化系统分析	4
3.1 系统业务流程分析	4
3.1.1 数据库连接流程图	4
3.1.2 数据库操作流程图中	5
3.1.3 性能评估流程	5
3.1.4 压力测试流程图	6
3.1.5 sql 优化流程图	7
3.2 系统数据流分析	7
3.2.1 顶层数据流图	7
3.2.2 数据库连接数据流图	8
3.2.3 数据库操作数据流图	9
3.2.4 性能评估数据流图	9
3.2.5 压力测试数据流图	10
3.2.6 sql 优化数据流图	11
3.3 系统数据字典	11
4. MySQL 管理优化系统设计	14
4.1 结构设计	14
4.2 总体功能结构设计	14
4.2.1 功能结构图	14
4.2.2 主要模块说明书	15
4.3 界面设计	17
4.3.1 主界面	17
4.3.2 自定义 sql 界面	17
4.3.3 日志控制台界面	18

4.3.4 数据表管理界面	18
4.3.5 参数管理界面	19
4.3.6 性能评估界面	20
4.3.7 压力测试界面	21
4.3.8 sql 优化界面	22
5. MySQL 管理优化系统实现	22
5.1 程序模块实现	22
5.1.1 数据库连接模块	23
5.1.2 数据管理模块	23
5.1.3 性能评估模块	25
5.1.4 压力测试模块	26
5.1.5 sql 优化模块	27
6. MySQL 管理优化系统测试	28
6.1 运行测试	28
6.2 优化测试	28
7. 结论	30
7.1 创新与不足	30
7.2 未来展望	30
参考文献	31
致谢	1

MySQL 管理优化工具的设计与实现

邵震

南京信息工程大学管理工程学院，江苏 南京 210044

摘要： 数据库的优化往往需要专业的数据库管理人员（DBA）进行人工操作。这大大增加了数据库管理人员的工作量。同时，开发人员或非专业的系统管理人员可能缺乏数据库优化的相关知识，雇佣专业的数据库管理人员，既提高了成本，又增加了工作环节从而增加了协作的复杂度。本工具的开发是为了能简化数据管理人员的工作量，同时对于开发人员，以及非专业的管理人员能够获取正确的数据库优化建议。实现在没有数据库管理人员的情况下，也能够通过图形化界面完成基本的数据库优化工作。本工具针对的是当前最流行的关系型数据库系统 MySQL 进行管理与优化。

关键词： 数据库优化；MySQL；SQL 优化

The Design and Implementation of MySQL Management Optimization Tool

Shao Zhen

School of Management Science and Engineering, NUIST, Nanjing 210044, China

Absrtact: Database optimization usually requires manual operation by professional database administrator (DBA). This greatly increases the workload of the database administrator. At the same time, developers or non-professional system managers may lack relevant knowledge about database optimization, and hiring professional database managers not only increases costs, but also increases the number of work steps and thus increases the complexity of collaboration. The development of this tool is to simplify the workload of data managers, and at the same time, developers and non-professional managers can obtain correct database optimization recommendations. To achieve the basic database optimization work through the graphical interface without the database management staff. This tool is aimed at managing and optimizing MySQL, the most popular relational database system.

Key words: Database Optimization; MySQL; SQL Optimization;

1. 概述

数据库优化，是对于数据库表现所依据的因素，例如数据表、查询和参数配置进行调整从而提高数据库运行的效率。数据库优化工作是运维工作中的重要一环，数据库管理人员在运维数据库时往往需要借助一些图形化工具，但是现有的工具仍然有许多可以完善的部分。在本节中主要阐述本系统的开发背景以及研究现状的概述。

1.1 选题背景

数据库优化是一项非常繁杂的工作，很多时候仍然需要根据实际情况进行人工调优。但是当前的时代数据库的应用越来越广泛。并不只有系统的开发人员或运维人员才会面对数据库。在各行各业中都应用信息系统的今天，普通的系统管理人员，例如企业的 ERP 管理人员等也需要接触到数据库的使用。他们可能缺乏相应的数据库专业知识，如果再去聘请专业的 DBA，无论是经济成本还是时间成本无疑会增加。同时，很多情况下普通的系统管理人员需要的仅仅是轻量化的维护工作，面对太过复杂的问题可以通过申请软件厂商的系统技术支持来解决。在这种业务场景下 DBA 显然也是不需要的。

通常 DBA 在优化工作中，需要自行设计测试环节以量化优化指标，这增加了工作量，也让运维工作更为复杂。同时也频繁的修改数据库参数，分析日志等操作又对 DBA 的要求较高，增加了需要记忆的内容，这大大增加了数据库管理人员的工作量。与此同时，很多情况下并没有专业 DBA 参与运维工作，需要开发人员甚至是非专业的管理人员去完成优化工作。非专业的系统管理人员可能缺乏数据库优化的相关知识。无法完成数据库性能的评估与优化工作。例如企业的 ERP 运维人员经常需要对数据库中的数据进行整合，设计查询脚本，这时候如果不经优化直接将效率低下的语句输入业务数据库，可能会导致整个系统的运行缓慢，很多时候这些运维人员并不具备优化查询语句的相应知识。如果专门雇佣专业的数据库管理人员，既提高了成本，又增加了工作环节从而增加了协作的复杂度。

MySQL 作为开源的关系型数据库管理系统，在软件生态上自然有着无与伦比的优势。但是由于其开源的性质，在技术支持方面有所欠缺。与其他的大型数据库相比仍有一些不足之处，MySQL Cluster 的功能和效率都相对比较差^[1]。所以在使用 MySQL 时如果相应的专业能力不足，不能进行必要的优化，则会造成工作效率的降低。甚至在没有进行测试的情况下，不能很好的评估当前数据库的使用效率。MySQL 虽然自带压测工具 `mysqlslap`，但是命令行的操作不如图形化操作界面便捷，易于上手。同样的，现在已有的 sql 优化工具，大多以命令程序为主，各个软件的命令标准不一，需要记忆的命令较多，这无疑增加了 DBA 的工作量。

本工具的开发是为了能简化数据管理人员的工作量，同时对于开发人员，以及非专业的管理人员能够获取正确的数据库优化建议。参照当前现有的相关工具，利用图形化界面实现三个优化模块：索引/参数管理模块，性能评估模块，sql 优化模块。让数据库用户能够简单便捷对数据库日志，索引以及参数进行管理。同时能够利用图形化界面，快捷的进行压力测试，sql 优化分析等功能。为适应不同的工作环境，开发多个版本，分别支持 C/S，B/S 架构，以及 Linux 和 Windows 两种操作系统。本工具旨在让 MySQL 的优化工作更为简便，易于上手，从而提高数据库的运行效率。

1.2 发展现状

1.2.1 国外发展现状

在 2017 年的国际计算机学会 (ACM) 国际会议中发表的使用机器学习来进行数据自动调优的研究成果: OtterTune^[2]。OtterTune 是由卡耐基梅隆大学开发的自动化数据库调优工具。完全开源, 其开发目的是让任何人都能够部署数据库管理系统。OtterTune 会利用之前的对于数据库管理系统的优化部署来优化新的部署, 在这点上与其他数据库管理系统配置工具不同。这种特性非常明显的减少了部署新的数据库管理系统的时间以及资源。OtterTune 建立一个存储之前的优化过程中收集的数据的库。并且利用这个库来构建机器学习模型, 这是为了获取数据库管理系统对不同配置的响应方式。OtterTune 使用这些模型来指导新应用程序的试验, 进而推荐最佳的优化设置。

此外商用数据库一般也拥有自己的优化方案。例如 Oracle 查询优化技术, 部署在 Oracle 数据库服务器上的自管理系统。但是 Oracle 这一功能属于商用产品的功能, 不符合降低成本的要求, 同时很多情况下不需要使用如此复杂的优化功能^[3]。

除了优化系统的开发, 大多数的相关研究更多的是关注于优化算法方面, 更多的是为开发者提供理论依据。例如, 线性回归算法在 sql 查询优化中应用^[4], 机器学习, 神经网络以及大数据框架例如 Spark 的算法研究^[5]。

1.2.2 国内发展现状

国内的目前还没有具有普适性的数据库调优方案的论文发表, 但是已经有一些互联网公司的数据库团队开发出了数据库调优工具。比如由小米人工智能与云平台的数据库团队开发与维护的 sql 优化与重写工具 SOAR。以及由美团点评公司的 DBA 团队开发的 MySQL 索引优化工具 SQLAdvisor, 这些软件的缺点主要是它们都是命令行工具, 服务于专业的数据库管理人员, 不利于非专业的信息系统管理人员使用。

同时和国外的商用数据的优化方案相同, 国内的阿里云等云服务供应商, 将自己商用的数据库优化方案都划分为技术支持的一部分, 属于闭源的优化方案。

除了企业开发的具有普适性的优化系统以外, 针对不同特定的业务场景下的系统设计是国内发展的一大趋势, 例如针对医院数据库^[6]、高校境的外访问数据库^[7]以及网络安全方面^{[8][9]}的专业化数据库调优方法的研究。对于优化理论的研究的一大重点是针对 sql 的语句逻辑的优化^{[10][11]}, 而优化的方法主要集中于视图优化、索引优化、连接优化、sql 复用这几个方面。

1.2.3 存在问题

从国内外的解决方案技术层面来看, 主要存在以下两个问题:

一是许多的相关软件为了适应服务器需求, 多使用命令行程序, 部署以及操作对使用者有一定要求。也增加了数据库管理人员的工作负担。

二是优化工具往往不具备数据库管理工具的对不同数据库连接的管理特性。所有的日志, 以及优化结果是面向工具本身而非具体优化对象。当一台机器上部署了多个数据库, 或管理人员需要通过一个工具对多个数据库进行优化操作时无法针对不同的数据库优化信息进行分析工作。因此需要对两者进行整合。

从应用层面来看, 现有的数据库优化系统或者优化方案, 以及研究, 主要是针对特定领域的特定业务场景进行的, 缺乏灵活性。

1.3 目的及意义

针对如何提高工具的易使用性, 开发出相应的图形化处理界面, 同时考虑到业务场景, 需要适应 Windows 和 Linux 两种操作系统, 需要较强的跨平台性, 所以决定采用 swing 框架, 用 java 来构建一个桌面应用。为了解决远程操作的问题, 同时也需要开发出一个 B/S 架构的应用, 所以选用 WebSwing 框架在桌面版的基础上创建一个 Web 程序。使用户可以根据应用场景选用不同的版本。

一般来说数据库优化可以从三个方面进行考虑: 利用应用程序负责检查数据之间的约束条件、利用应用程序提高语句的执行效率、利用应用程序来实现通用语句无法实现的需求^[12]。也就是说从现实意义上来讲, 本工具作为数据库优化应用程序能够帮助数据库管理人员更加轻松的完成 MySQL 的基础优化任务。提供较为详尽的指导, 使得一般用户能够很快掌握并实施 MySQL 的基础优化工作。从理论意义来说, 能够弥补当前的 MySQL 优化方案复杂、不易上手、功能不够集中的缺点。

2. MySQL 管理优化系统规划

系统规划主要包括了开发环境的描述以及可行性分析两个部分。本节将会详细说明系统开发所使用的基础环境, 包括编译环境以及测试环境, 以及系统开发的管理可行性、技术可行性、经济可行性。

2.1 系统开发环境

开发语言为 Java, 发行版本为 OpenJDK 13.0.2。JRE:OpenJDK Runtime Environment (build 13.0.2+8)。JVM:OpenJDK 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing) 集成开发环境为 IntelliJ IDEA 2019.3.1。开发测试所使用的 RDBMS 为 MySQL8.0.18 / MySQL5.7.19。对 MySQL 8 以上版本和 8 以下版本都进行了测试。Java 程序的 Web 打包框架为 WebSwing 2.7.5。线上测试框架所依赖的 java 发行版本为 openJDK 11。

2.2 MySQL 管理优化工具开发可行性分析

2.2.1 管理可行性

整个项目采用 GPL3.0 (GNU General Public License) 许可, 无管理压力, 维护压力。完全自由开源, 任何更改、二次发布都必须沿用 GP3.0 许可, 原作者对修改不负任何责任^[13]。利用 Copyleft 而不是 Copyright 去管理软件。Copyleft 是指所有人都可以对使用它去重新分发软件, 不管有没有对软件进行修改, 但是相应的, 必须保留软件所具有的自由特性。相当于放弃了软件的版权, 能够使得软件进入公有领域, 但是同时约束任何修改软件的人, 要求他们必须保留软件的自由特性。这样实际上并不需要花费精力在软件管理上, 仅需要按照当前的情况更新软件的主干即可。

2.2.2 技术可行性

利用 Java 的良好跨平台性，能够很好的适应 Windows 和 Linux 两种操作系统，满足多平台的 MySQL 管理。Java 可以利用多线程使得程序具有更好的交互性和实时性。同时 Java 在多线程处理上的性能表现良好，因此在 Java 语言中进行多线程处理也很简单，从技术上来说，java 语言能够降低开发的难度。

考虑到 java 在连接时的性能问题，同时兼容 mysql8.0 以前和 8.0 以前的版本，选择 com.mysql.cj.jdbc.Driver 作为 JDBC 的驱动^[14]。

同时利用 WebSwing 框架可以同时实现 B/S, C/S 框架，满足不同业务场景。Webswing 是一款可以让 java 应用程序在浏览器中运行的 web 服务。它只需要在 web 界面简单设置就可以将本地的 applet 程序或基于 swing 的 java 应用程序部署在浏览器中^[15]。

2.2.3 经济可行性

开发工具 IDEA 为 JetBrains 教育许可，免费使用。其余开发环境皆为开源免费。用户运行环境需要 JRE 也是完全免费。本工具为 GUN 软件，完全遵守 GPL3.0 许可，这也意味着开发者不需要对于依据本软件进行二次发布的内容负责，对于软件的维护来说是零成本的，同时作为一款完全开源工具，也不期望任何收益。综上所述，本软件的开发在经济层面上是完全可行的。

3. MySQL 管理优化系统分析

系统分析主要包含针对各个模块的业务流程进行分析、对各个模块的数据流动进行说明以及数据流图中出现的数据的数据结构。所有数据的数据结构都将以数据字典的形式展示。

3.1 系统业务流程分析

3.1.1 数据库连接流程图

用户输入连接需要的相应信息来连接数据库。如果选择测试连接，那么仅测试是否可以成功连接至数据库而不进行实际连接。如果选择保存连接信息，则会将用户信息保存至外部数据中。如果用户在连接时出现异常，系统会对异常进行分析，反馈给用户准确的错误原因，使用户能够快速解决异常。数据库连接流程如图 1 所示：

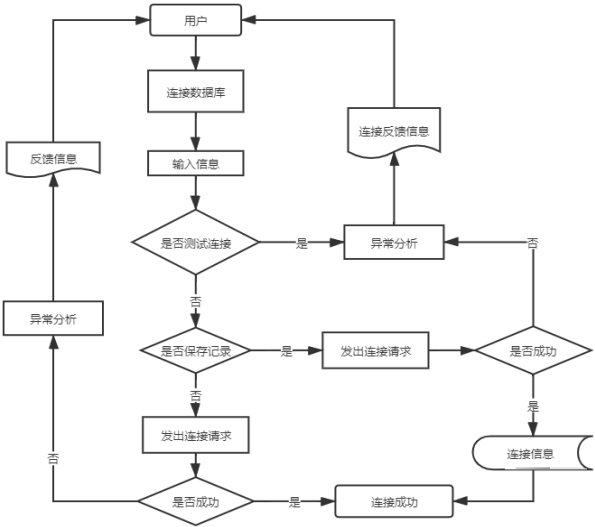


图 1 数据库连接流程图

3.1.2 数据库操作流程

用户可以通过图形化界面来管理数据库，也可以使用自定义的 SQL 或脚本进行操作。如果通过固定的图形化界面操作出现异常，系统会对数据库反馈的异常进行异常分析，经过处理后反馈具体错误分析，便于用户快速解决异常。如果使用自定义的形式，则会将执行信息直接返回控制台，包括时间戳、执行时长、错误信息等。数据库操作流程如图 2 所示：

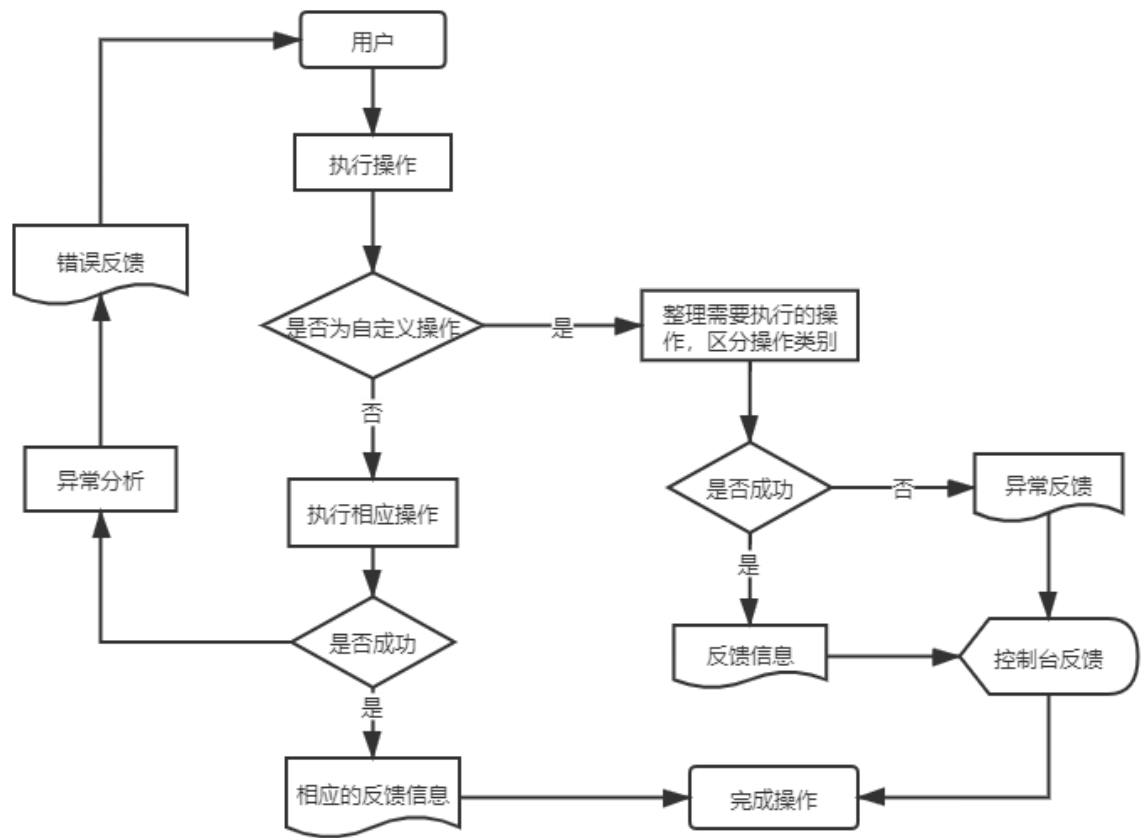


图 2 数据库操作流程

3.1.3 性能评估流程

常见的 MySQL 数据库设计方法有两种，分别是单线程操作 MySQL 和多线程同时操作 MySQL，因为多线程操作相较于单线程操作响应速度更快^[16]，所以在进行数据库的并发测试时，采用多线程的操作方法。在记录评估标准时，同时记录数据库参数状态用于性能比较。自定义操作支持针对常用的四种 MySQL 优化参数进行评估。四种参数分别是索引缓冲区（key_buffer_size）默认为 8MB，全连接缓冲区（join_buffer_size）默认为 128KB，全表扫描缓冲区（read_buffer_size）默认为 60KB，排序缓冲区（sort_buffer_size）默认为 256KB^[3]。用户可以根据评估结果对参数进行合理的调整。性能评估流程图如图 3 所示：

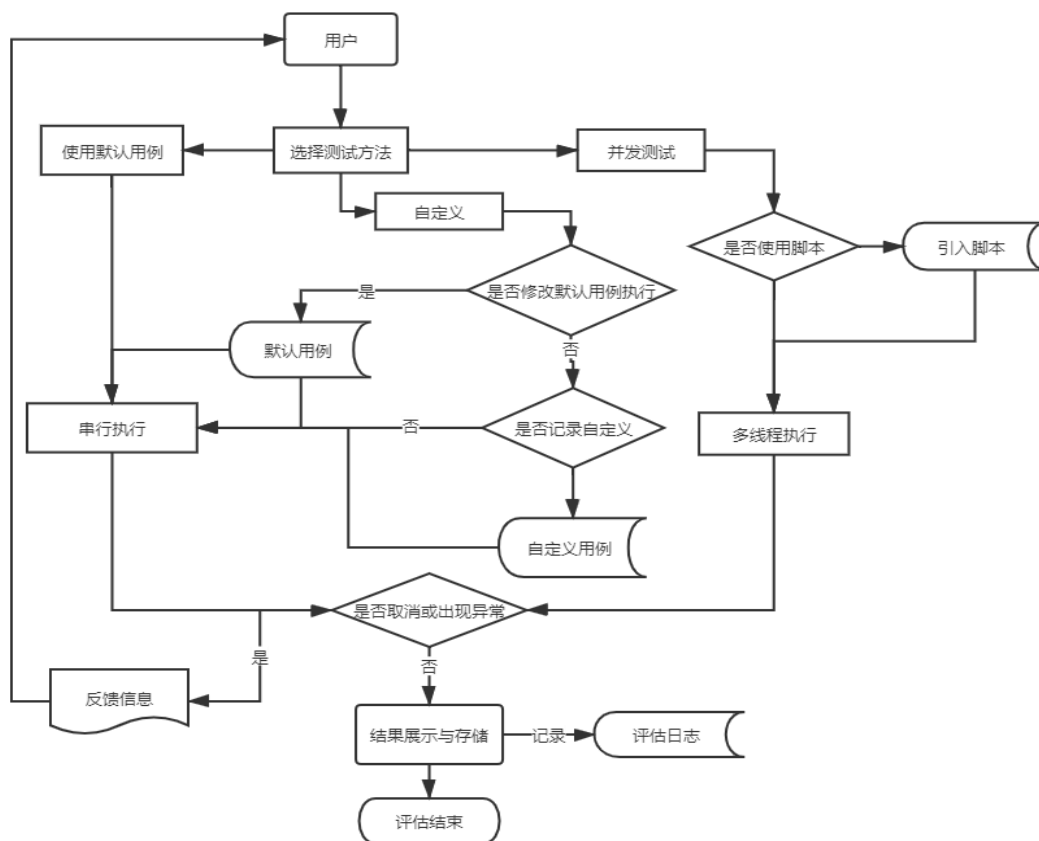


图 3 性能评估流程图

3.1.4 压力测试流程图

利用 mysql 自带的 mysqlslap 进行压力测试。依据 mysql 参考手册第四章第五节第八条^[17]对其的功能描述,对于 mysqlslap 这一命令行工具,设计图形化界面,让用户使用该工具。用户可以选择之前保存的数据库信息,或自定义测试库。测试库设定完成后,输入用例,选择相应的参数提交执行,执行日志会保存在本地。压力测试流程如图 4 所示:

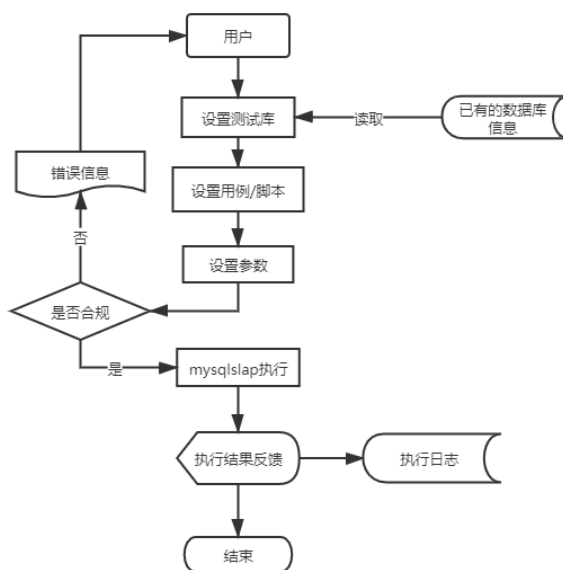


图 4 压力测试流程图

3.1.5 sql 优化流程图

主要基于开源工具 SOAR 来进行优化操作。设计图形化界面，支持 sql 语句优化、语法检测、sql 美化、ALTER 语句合并、EXPLAIN 语句分析等功能。记录相应的操作日志。用户可以选择是否使用测试数据库，如果使用，那么可以通过从已保存的信息中选取或自定义的形式进行配置。结束后填写用例，选择功能，系统自动生成命令进行执行。执行结果在控制台进行反馈，并记入日志。sql 优化流程如图 5 所示

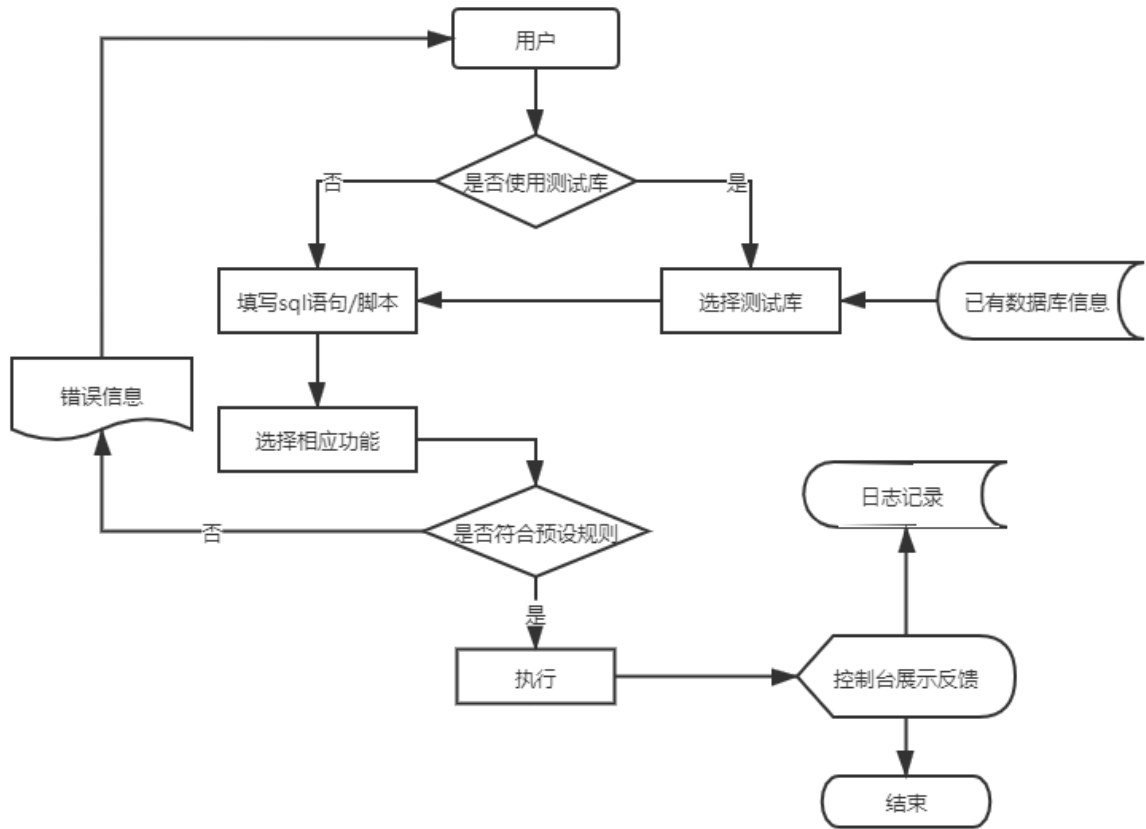


图 5 sql 优化流程图

3.2 系统数据流分析

3.2.1 顶层数据流图

顶层的数据流动在五个顶层数据模块进行，主要涉及各个模块的日志数据的存储。用户 在未连接数据库的状态下只能够，访问连接模块、sql 优化模块和压力测试模块。而数据库 管理与性能评估模块需要连接数据库之后才可以访问。系统的顶层数据流如图 6 所示：

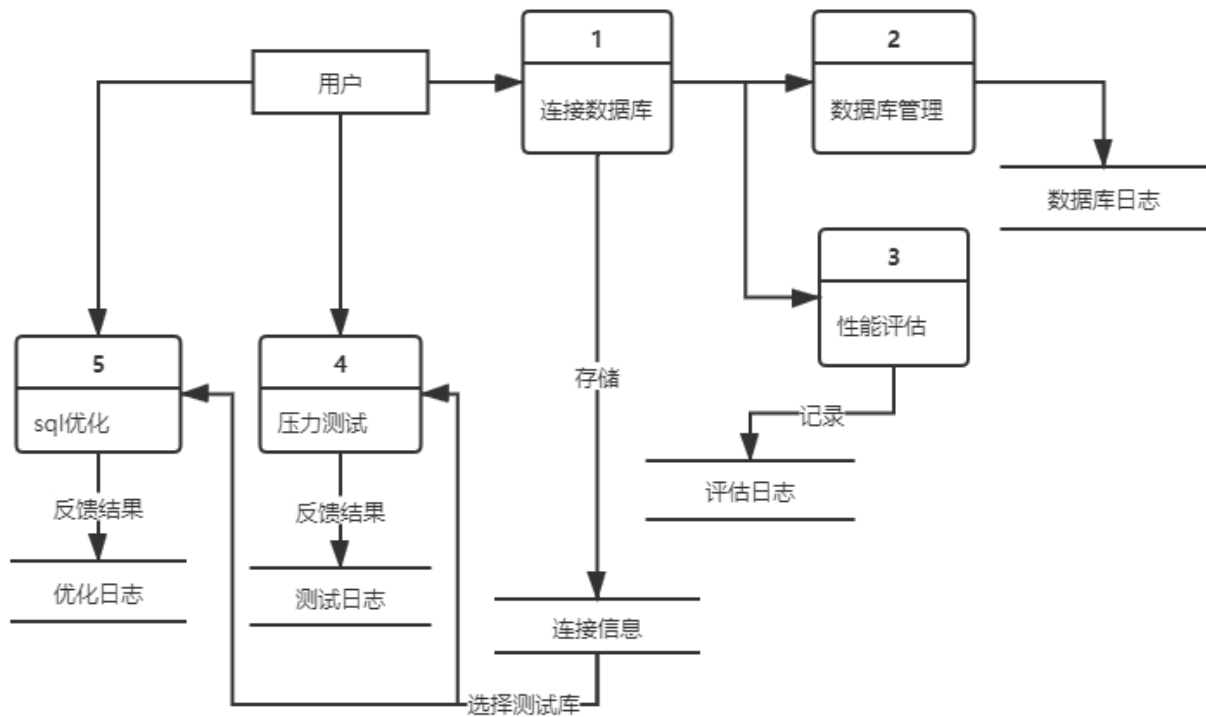


图6 顶层数据流图

3.2.2 数据库连接数据流图

数据库连接主要涉及连接信息的流动，存储连接的地址（包括端口）、用户名、密码，以便用户快速连接数据库。用户可以通过选择或自定义两种方式生成连接信息，这些信息通过连接功能连接至数据库。数据库连接模块的数据流如图7所示：

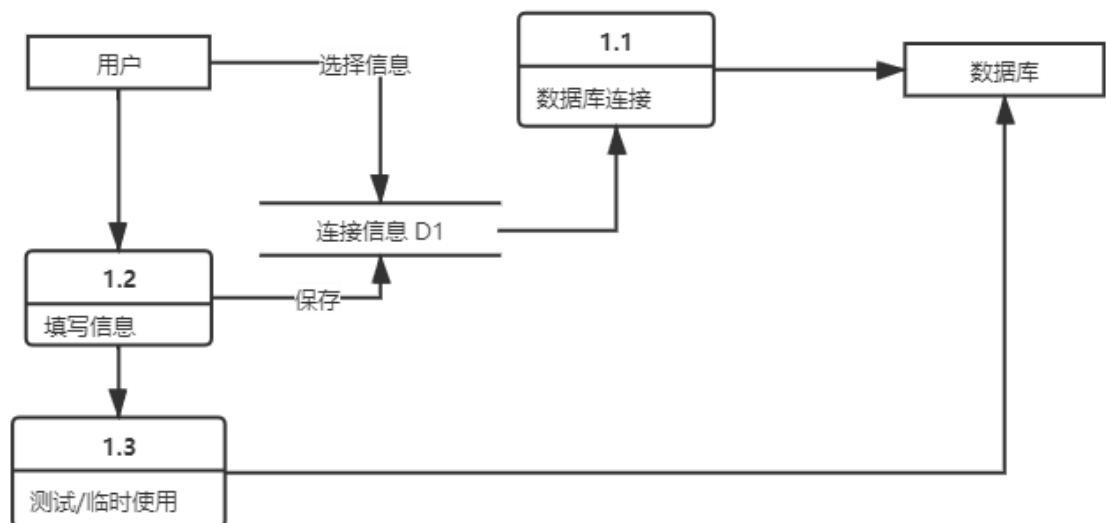


图7 数据库连接数据流图

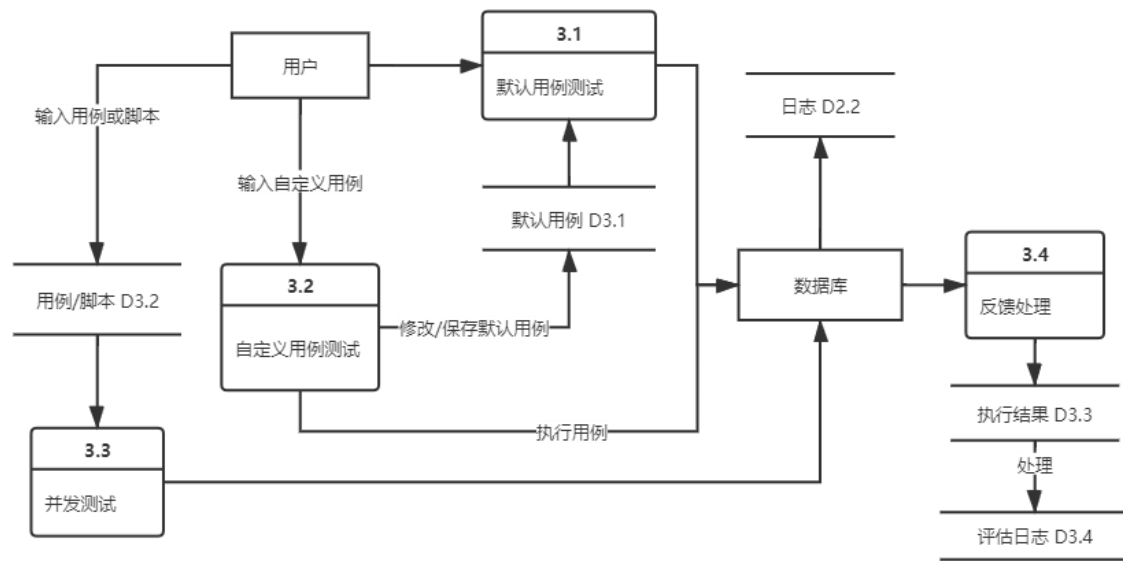


图 9 性能评估数据流程图

3.2.5 压力测试数据流程图

用户在使用压力测试时可以选择已有的连接信息，同时必须输入测试用例。选择相应的功能，系统自动生成命令，最终命令直接交由 `mysqlslap` 执行，反馈执行结果，执行结果主要由 3 部分组成平均执行时长、最大执行时长和最小执行时长。对于执行结果进行处理之后记录在 `slap` 日志中，`slap` 日志中的数据结构主要由时间戳、反馈结果、执行的命令构成，其中命令中的敏感信息会被隐去。压力测试数据流如图 10 所示：

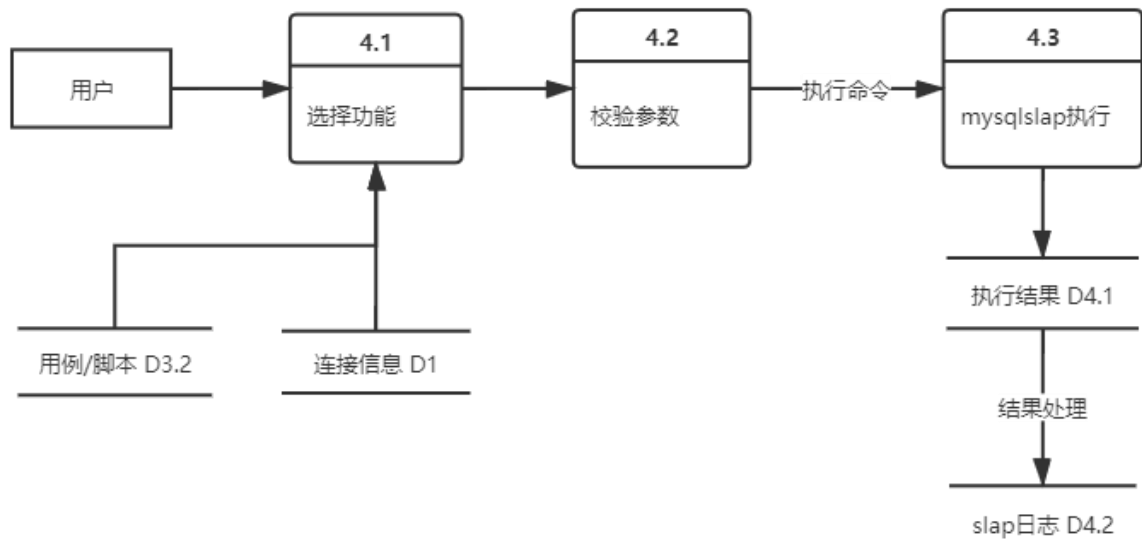


图 10 压力测试数据流程图

3.2.6 sql 优化数据流图

用户可以选择性的填写数据库信息，和压力测试相同需要输入外部用例，外部用例必须是 sql 语句或脚本的形式。针对用户选择的功能进行预处理，系统自动生成相应的命令，交由 SOAR 执行。获得反馈结果（优化建议），反馈的结果出现在反馈区中, 对结果进行处理，加上时间戳，隐去敏感信息，最后将优化日志写入优化日志中。sql 优化数据流如图 11 所示：

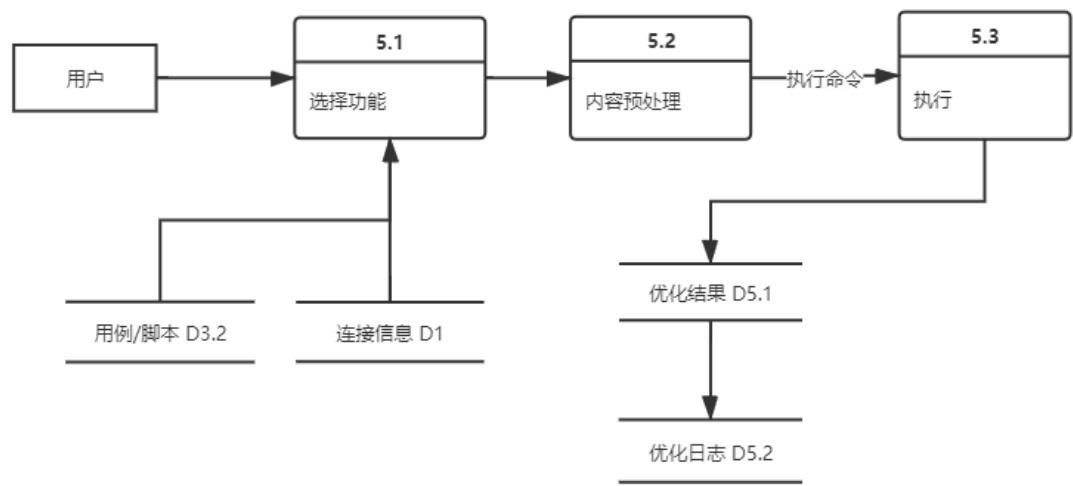


图 11 sql 优化数据流图

3.3 系统数据字典

连接信息的数据字典主要描述的是进入数据库连接模块的数据结构。存储于系统的外部数据中，占用系统部署位置的空间。连接信息数据如表 1 所示：

表 1 连接信息数据结构表

数据名称：连接信息		标识符：D1		
名称	数据类型	默认值	是否允许为空	说明
连接名称	varchar		否	PK
数据库地址	varchar		否	需要包含端口
用户名	varchar		否	
密码	varchar		否	

查询结果属于系统内部数据，临时的显示在系统的主界面上，仅仅占用系统内存，不占用任何硬件存储空间。查询结果信息如表 2 所示：

表 2 查询结果数据结构表

数据名称：查询结果		标识符：D2. 1
数据结构：	与查询出的数据表结构相同	

MySQL 日志分为查询日志、慢日志、错误日志三种。查询日志主要是记录数据操作的相关信息，慢日志记录超出时间限制的操作信息，错误日志记录异常信息。日志的有数据库处理，存储在本地，属于系统外部数据。数据字典分别如表 3、4、5 所示：

表 3 数据库查询日志数据结构表

数据名称：数据库日志		标识符：D2. 2		
数据类型：查询日志		启动信息：数据库位置，端口		
名称	数据类型	默认值	是否允许为空	说明
Time	DATETIME		否	PK

Id	int		否	线程 Id
Command	varchar		否	命令类型
Argument	TEXT		是	具体命令

表 4 数据库慢日志数据结构表

数据名称：数据库日志		标识符：D2.2		
数据类型：慢日志		启动信息：数据库位置，端口		
名称	数据类型	默认值	是否允许为空	说明
Time	DATETIME		否	PK
User@Host	varchar		否	用户@数据库地址
Id	int		否	线程 Id
Query_time	double		否	高于慢日志的设定值
Lock_time	double		否	锁定时长
Rows_sent	int		否	返回行数
Rows_examined	int		否	检索行数
Argument	TEXT		否	执行过慢的语句

表 5 数据库错误日志数据结构表

数据名称：数据库日志		标识符：D2.2		
数据类型：错误日志		启动信息：无		
名称	数据类型	默认值	是否允许为空	说明
时间	DATETIME		否	PK
错误类型	varchar		否	
错误内容	TEXT		否	

默认用例是用户需要默认执行的 sql 用例，属于系统外部数据，占用用户的本地内存。从用户处输入，最终流向外部存储。默认用例数据结构如表 6 所示：

表 6 默认用例数据结构表

数据名称：默认用例		标识符：D3.1		
名称	数据类型	默认值	是否允许为空	说明
sql 语句	TEXT		否	
执行次数	int		是	在 sql 语句直接用@连接

用例/脚本数据是用户自定义的 sql 语句或 sql 脚本，属于系统的外部数据，用户通过输入或选择输入该数据，流入性能评估模块进行处理。用例/脚本数据结构如表 7 所示

表 7 用例/脚本数据结构表

数据名称：用例/脚本		标识符：D3.2		
名称	数据类型	默认值	是否允许为空	说明
sql 语句	TEXT		否	
执行次数	int		是	在 sql 语句直接用@连接

性能测试执行结果是用户在执行性能测试功能后输出的结果，性能测试执行结果属于系统内部数据，仅在界面显示，它由 2 部分组成，一个是本次的执行结果，另一个是从属于外部数据的评估日志中读取的前次执行结果。性能测试执行结果数据结构如表 8 所示：

表 8 性能测试执行结果数据结构表

数据名称：性能测试执行结果			标识符：D3.3	
名称	数据类型	默认值	是否允许为空	说明
时间	DATETIME		否	
执行时长 1	int	0	否	join_buffer_size 对应（ms）
执行时长 2	int	0	否	read_buffer_size 对应（ms）
执行时长 3	int	0	否	sort_buffer_size 对应（ms）
执行时长 4	int	0	否	key_buffer_size 对应（ms）
前次执行时长 1	int	0	否	join_buffer_size 对应（ms）
前次执行时长 2	int	0	否	read_buffer_size 对应（ms）
前次执行时长 3	int	0	否	sort_buffer_size 对应（ms）
前次执行时长 4	int	0	否	key_buffer_size 对应（ms）

评估日志是在完成性能测试结果的处理后产生的数据，主要是记录评估操作，方便用户对历史数据进行查看、分析。评估日志以 csv 文件的格式存储在用户的本地存储中。具体数据结构见表 9：

表 9 评估日志数据结构表

数据名称：评估日志			标识符：D3.4	
名称	数据类型	默认值	是否允许为空	说明
时间	DATETIME		否	PK
执行时长 1	int	0	否	join_buffer_size 对应（ms）
执行时长 2	int	0	否	read_buffer_size 对应（ms）
执行时长 3	int	0	否	sort_buffer_size 对应（ms）
执行时长 4	int	0	否	key_buffer_size 对应（ms）
join_buffer_size	int		否	当前参数值（KB）
read_buffer_size	int		否	当前参数值（KB）
sort_buffer_size	int		否	当前参数值（KB）
key_buffer_size	int		否	当前参数值（KB）

slap 的执行结果来自于压力测试模块，是 mysqlslap 执行的反馈结果，属于系统的内部数据，仅在压力测试界面的反馈区进行显示，不占用用户的本地存储。slap 执行结果的数据结构见表 10：

表 10 slap 执行结果数据结构表

数据名称：slap 执行结果			标识符：D4.1	
名称	数据类型	默认值	是否允许为空	说明
平均执行时间	double		否	所有语句执行的平均时间
最大执行时间	double		否	所有语句执行的最大时间
最小执行时间	double		否	所有语句执行的最小时间

slap 日志是在完成压力测试后系统自动生成的数据，属于系统的外部数据，主要目的是便于用户查看历史测试结果，从而进行性能分析工作。存储位置为用户本地存储。slap 日志的数据结构如表 11 所示：

表 11 slap 日志数据结构表

数据名称：slap 日志			标识符：D4.2	
名称	数据类型	默认值	是否允许为空	说明
时间	DATETIME		否	
反馈结果	TEXT		否	执行结果
命令	TEXT		否	mysqlslap 执行的命令（密码隐藏）

优化结果从 sql 优化模块产生，是系统在调用 soar 工具之后，从上游返回的数据。优化结果属于系统内部数据，仅在 sql 优化界面的反馈区显示，暂时性的占用内存，窗口关闭后即释放存储空间。具体数据结构见表 12：

表 12 优化结果数据结构表

数据名称：优化结果	标识符：D5.1
数据结构：SOAR 的对应返回结果	

优化日志属于系统的外部日志，在优化操作执行完成后由系统自动生成，主要用于用户进行优化历史追溯。存储与用户的本地内存中。具体数据结构见表 13：

表 13 优化日志数据结构表

数据名称：优化日志			标识符：D5.2	
名称	数据类型	默认值	是否允许为空	说明
时间	DATETIME		否	
反馈结果	TEXT		否	执行结果
命令	TEXT		否	执行的命令（密码隐藏）

4. MySQL 管理优化系统设计

系统设计主要包括了结构设计，总体功能结构设计以及各模块的详细设计信息。结构设计涵盖了系统的架构特点，以及层次说明。功能结构设计主要涵盖了各个功能模块的输入、输出、内部数据、实现算法等信息。

4.1 结构设计

基于 Java Swing 技术，采用 3 层 C/S 架构的设计模式，参照 MVC 的架构模式，分别设置表示层、功能层、数据层。表示层负责处理用户交互。功能负责数据的控制，各项业务逻辑的处理。数据层负责访问数据库或本地数据。

4.2 总体功能结构设计

4.2.1 功能结构图

系统主要分为三个顶层功能模块，分别是数据库管理模块、数据库性能评估模块、数据库优化模块。

数据库管理模块：分为三个子功能模块，分别是基础管理、日志管理、参数管理。基础管理主要是负责数据的增删改查以及数据表结构和数据表索引的管理。

数据库性能评估模块：主要分为性能评估以及压力测试两个子功能模块。性能评估主要针对参数以及数据库并发性能的评估。

数据库优化模块：主要实现 soar 的一些常用功能，这里选取了 sql 语句重写、美化、评估、EXPLAIN 结果分析、ALTER 语句合并。

系统的总体功能结构如图 12 所示：

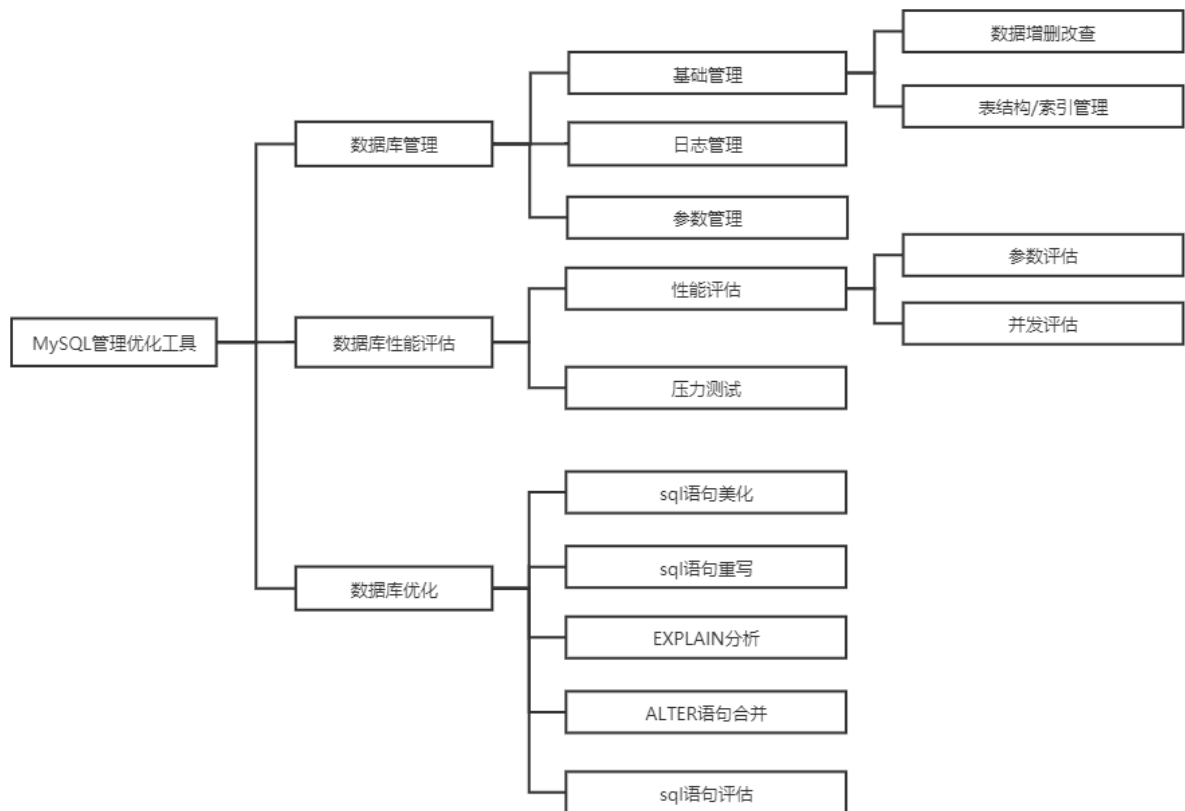


图 12 系统功能结构图

4.2.2 主要模块说明书

(1) 数据库管理：主要功能为实现数据的增删改查，数据表结构调整，数据表的索引管理，MySQL 日志管理。数据库管理模块的详细信息见表 14：

表 14 数据库管理模块表

所属子系统		MySQL 管理系统
模块名称	数据库管理	
调用模块	数据库连接、数据库操作、日志管理	
输入	数据库地址，用户名，密码，sql 命令/操作	
输出	连接成功，连接失败的异常分析，查询结果，固定操作的异常分析，自定义操作的 MySQL 反馈，时间戳，执行时长，日志文件。	
相关外部数据	连接信息 connInfo.txt	
主要内部变量	数据库连接，sql 命令/操作，查询结果	
算法	利用 JDBC 连接数据库，如果出现异常，对其进行分析反馈给用户是因为何种原因错误。将连接相关信息写入本地文件，对密码做简单加密处理。连接数据之后，构建连接-模式-表的 3 层树状图。对于用户操作生产相应的 sql 语句执行，反馈结果。用户自定义操作则返回结果至控制台。	

(2) 数据库性能评估：主要功能为根据用例执行的时长，分别对数据库参数进行评估，给 MySQL 的参数调优提供参考指标。同时弥补 mysqlslap 在测试历史比较，特定参数测试等方面的缺陷。数据库性能评估模块的详细信息见表 15：

表 15 数据库性能评估模块表

所属子系统	MySQL 优化系统
模块名称	数据库性能评估
调用模块	连接管理, 评估日志管理, 数据库管理
输入	用例 (sql 语句/脚本), 执行次数
输出	执行时间, 前次执行时间, 主要参数数值, 评估日志文件。
相关外部数据	评估日志 Evaluation.csv, 默认用例 defaultcase.txt
主要内部变量	连接信息, sql 语句, 评估结果字典
算法	针对非并发测试, 采用函数方法, 串行执行用例, 利用单线程实现进度条功能, 计算平均用时, 格式化输出并记录。并发测试建立多线程执行, 主线程带动多个子线程, 阻塞主线程, 等待所有子线程完成, 最终同步格式化输出并记录。

(3) 数据库压力测试: 主要功能为利用 `mysqlslap` 工具对数据库进行压力测试, 反馈结果。使用户能够快速使用 `mysqlslap` 的基础功能。数据库压力测试模块的详细信息见表 16:

表 16 数据库压力测试模块表

所属子系统	MySQL 优化系统
模块名称	数据库压力测试
调用模块	数据库连接, 压力测试
输入	<code>mysqlslap</code> 参数, 测试库信息
输出	用例的执行最小, 最大, 平均执行时长。slap 日志
相关外部数据	slap 日志 slap.log
主要内部变量	数据库连接, sql 语句/脚本, <code>mysqlslap</code> 命令
算法	根据用户选择的功能与参数生成命令, 交由 <code>mysqlslap</code> 执行, 获取反馈。

(4) sql 优化: 主要功能为 sql 语句的评估、重写、美化。`EXPLAIN` 语句内容分析、`ALTER` 语句合并、为用户提供优化建议。Sql 优化模块的详细信息见表 17:

表 17 sql 优化模块表

所属子系统	MySQL 优化系统
模块名称	sql 优化
调用模块	数据库连接, sql 优化
输入	sql 语句/脚本, 测试库信息
输出	优化结果, 反馈信息, 优化日志
相关外部数据	优化日志 optimizeLog.log
主要内部变量	数据库连接, sql 命令/操作, 查询结果
算法	根据用户选择的功能与参数生成命令, 交由外部程序执行, 获取反馈。

4.3 界面设计

4.3.1 主界面

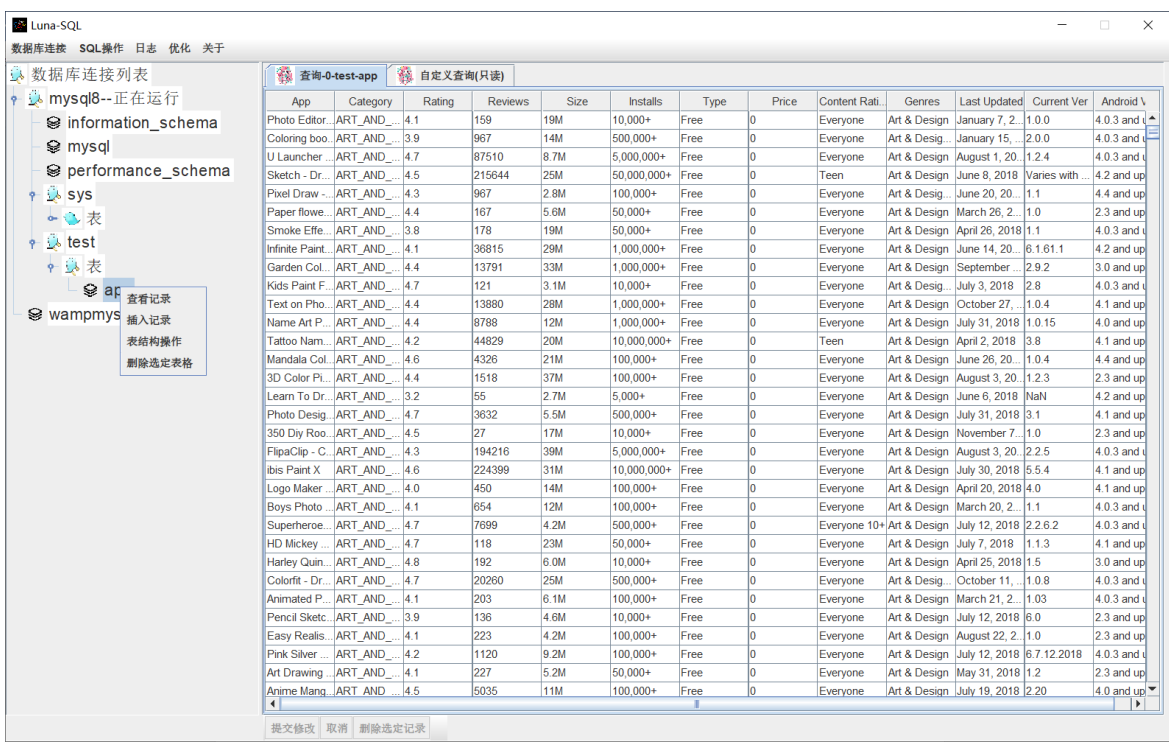


图 13 主界面设计

左侧展示数据库结构树状图，提供数据库操作选项。右侧展示查询结果，使用固定功能查询出的结果可以直接在表格上进行修改,通过下方工具栏进行 UPDATE 和 DELETE 操作。上方为工具栏。主界面设计如图 13 所示。

4.3.2 自定义 sql 界面

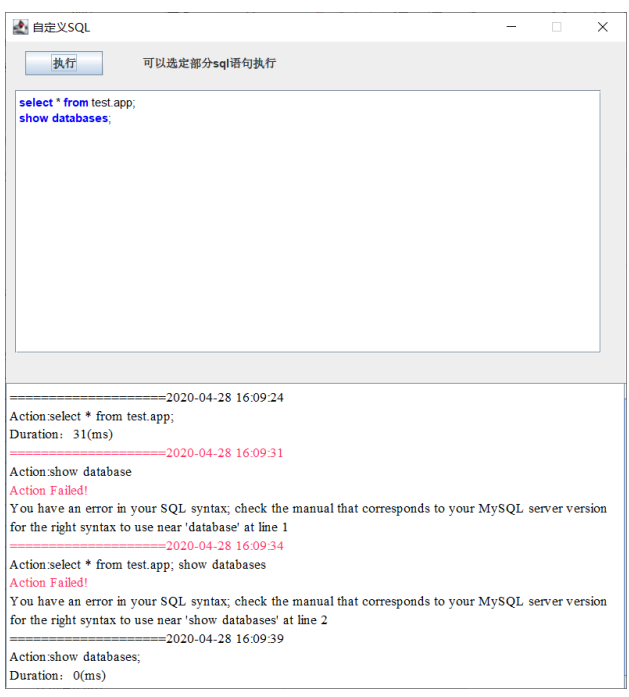


图 14 自定义 sql 界面设计

自定义 SQL 界面，主要分为两个部分，输入区和反馈区。输入区输入内容，执行后的反馈信息将记录在反馈区。输入区，采用 sql 文本编辑器的风格，关键字高亮，同时支持自定义选择部分 sql 语句执行。输出结果主要分为时间戳、执行的语句、报错/执行时长这三个部分。自定义 sql 界面如图 14 所示。

4.3.3 日志控制台界面



图 15 展示了日志管理界面的设计。该界面主要分为两个部分：查询日志状态和慢日志状态。在查询日志状态部分，用户可以选择日志的状态（当前为“开启”）、存储方式（当前为“FILE”），并指定日志路径（当前为“D:/mysql/log/general_log/mysql_log.txt”）。在慢日志状态部分，用户可以设置慢日志的时间限制（当前为“2”秒）和慢日志的路径（当前为“D:/mysql/log/slow_log/slow-log.txt”）。此外，界面底部还提供了错误日志路径的设置选项（当前为“D:/mysql/log/error_log/error-log.txt”）。每个配置项旁边都有一个“保存修改”按钮，用于保存所做的更改。

图 15 日志管理界面设计

日志控制台，针对 MySQL 的 general_log、slow_log、error_log 进行管理，显示日志的存储位置，设置日志开关，以及相应日志参数设置。日志管理界面见图 15。

4.3.4 数据表管理界面

主要氛围数据表创建界面，数据表结构管理界面两个界面，分别处理数据表的创建、结构管理与索引管理。数据表创建界面，除了基础的表名、字段名、字段类型等信息，还增加了最常用的三种字段设定，分别是主键、非空性和唯一索引。数据表创建界面见图 16：

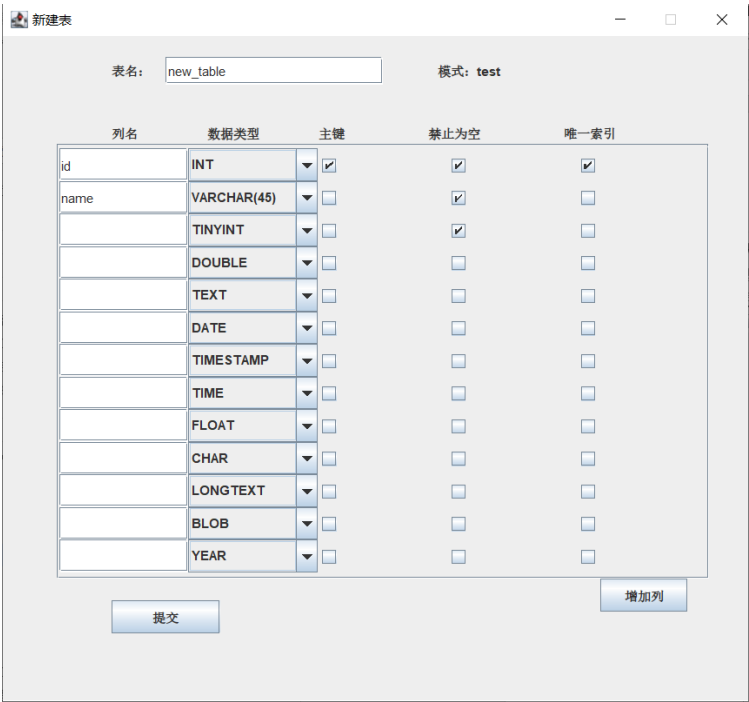


图 16 展示了数据表创建界面的设计。该界面用于创建新表，包含表名输入框、模式选择、列名、数据类型、主键、禁止为空和唯一索引等选项。表名为“new_table”，模式为“test”。列名输入框下方是一个表格，列出了常用的数据类型及其对应的非空性和唯一索引选项。表格下方有一个“提交”按钮，用于保存创建表的配置。

列名	数据类型	主键	禁止为空	唯一索引
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	TINYINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	TIMESTAMP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	TIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	CHAR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	LONGTEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	BLOB	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	YEAR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 16 数据表创建界面

数据表的结构管理界面，展示了数据表中所有的字段的基础信息，包括字段名，类型，长度，是否为主键，非空性。同时也可以对字段及其索引进行增加与删除处理。在用户对表结构的操作展现出非异常单不合理性的情况下，如对于表不进行主键设置时，系统会发出相应的警告。数据表结构管理界面设计如图 17 所示：

图 17 数据表结构管理界面

4.3.5 参数管理界面

常见参数的展示与修改，数据缓存区是 MySQL 的重要参数之一，缓存区大小是否合理直接影响数据库的性能，要根据当前操作系统综合评估^[18]。所以参数管理界面提供了常用的缓存区参数管理，也对于这些参数的作用进行了简明扼要的说明，方便用户快速了解这些参数，达到优化的目的。同时，针对其他不常用的参数，提供了快速打开 MySQL 配置文件的按钮。

参数界面提供了一些常用的数据库参数，缓冲区大小（table_open_cache）这是关于数据库性能优化非常重要的参数之一，同时提供了 Open_tables 和 Opened_tables 两个参数作为缓冲区大小的调整参考、innodb_flush_log_at_trx_commit，这个参数涉及数据库的安全性和效率的平衡性，默认设置为 1，这是考虑到对于用户来说回滚数据等操作的复杂性，同时还需要避免脏读等问题，故将安全性放在第一位，同时提供的 innodb_lock_wait_timeout 参数也是为了安全性考虑。其余的参数多为缓存区大小，一般而论这些参数的提高，在不考虑存储空间的情况下，往往都能显著地提高数据库的处理速度，更改这些参数的优化效率相交于其他参数来说要更高。参数管理界面如图 18 所示：

部分优化参数

table_open_cache(MB)

2000

缓冲区大小，当opened_tables不断增加且open_tables接近table_cache时，可以考虑适当调高该值,范围[1,524288]

innodb_flush_log_at_trx_commit

1

建议值为安全性最高的1,值为0时效率最高安全性最低，值为2时平均

innodb_lock_wait_timeout

50

行锁的死锁判断时间限制(s)

innodb_log_buffer_size

1048576

若更新操作峰值或负载较大，可以提高该值(Byte)

sync_binlog

1

索引缓冲区大小，所有数据库共享(Byte)

read_buffer_size

524288

全连接缓冲区大小，影响多表连接速度

binlog_cache_size

32768

全表扫描缓冲区大小

key_buffer_size

8388608

排序缓冲区大小，影响ORDER BY和GROUP BY速度

join_buffer_size

524288

日志缓存区大小

sort_buffer_size

524288

日志同步频率，为0表示日志文件写满再与磁盘同步

提交修改

参数更改后需重启数据库服务，更多参数可在配置文件中修改：

打开配置文件

Open_tables:410 / Opened_tables:9

图 18 参数管理界面设计

4.3.6 性能评估界面

性能评估的主界面提供评估方式的选择。主界面分别设三个子界面，自定义用例界面，并发测试的用例输入界面，结果反馈。自定义的四个参数为常用的缓存区大小参数^[1]。

自用用例界面主要分为两个部分，参数测试用例部分和默认用例部分。参数测试用例部分，主要负责进行四个常用参数的设置。数据库评估结果界面和自定义用例界面设计分别如图 19，20 所示：

评估结果

若使用的是默认模式：

所有值相同

参数名称	用例测试时间(ms)	较上次测试变化(ms)	上次测试时间
join_buffer_size	50	(-184)	2020-03-11 17:11:16
read_buffer_size	50	(-184)	2020-03-11 17:11:16
sort_buffer_size	50	(-184)	2020-03-11 17:11:16
key_buffer_size	50	(-184)	2020-03-11 17:11:16

参数值可在历史记录中查询

查看历史记录

图 19 评估结果界面设计



图 20 自定义用例页面设计

当然优化参数并不是越高越好，同时不同的参数对不同的语句有影响，需要对参数合理安排修改顺序，达到监控最优的效果^[19]。针对不同的参数使用不同的测试用例是很有必要的。评估的结果主要包含了执行时长、较上次测试的时间变化、上次测试时间这些能够直观反映结果的内容。较上次的时间变化也使用的不同的颜色，若时间减少则为绿色，无变化为橙色，时间更长则为红色，这样使得用户能更为直观的感受变化。

4.3.7 压力测试界面

压力测试界面下设一个选择测试库的子界面用于配置数据库，显示当前已经保存的连接可以直接选择，同时增加一行用于填写自定义测试库。主界面主要分为 3 部分。用例输入，主要用于输入 sql 语句或引入脚本以及查看日志；mysqlslap 参数填写，主要用于配置 mysqlslap 的相关参数；反馈区，主要用于显示执行结果。压力测试结果界面的界面设计如图 20 所示：



图 21 压力测试界面设计

参数部分主要是对于 `mysqlslap` 的常用功能进行了整合，分别设置了用例类型、并发数量、查询总数、总执行次数、测试引擎选择、查看测试语句这六个功能。反馈的结果分为时间戳、数据库引擎、平均执行时间、最小执行时间、最大执行时间、并发数量、每条并发执行的次数这 7 个部分。

4.3.8 sql 优化界面

优化界面主要分为三个部分，用例输入区、功能选择区、反馈区。用例输入区负责输入测试需要的 `sql` 语句或脚本。功能选择区主要负责不同优化功能的选择。反馈区负责向用户展示结果。功能区分为六个功能：导入 `sql` 脚本、评估 `sql` 语句、`sql` 重写、`sql` 美化、合并 `alter`、查询优化日志。优化日志的内容主要分为时间戳，`soar` 的执行结果以及执行命令三个部分。其中执行命令如果包含数据库密码等敏感信息，会对其进行抹除的处理。`Sql` 优化界面设计如图 22 所示：



图 22 优化界面设计

5. MySQL 管理优化系统实现

本节主要通过具体的代码来展示程序是如何实现的。通过对每一个模块的重要部分进行说明来体现程序实现的具体方法。

5.1 程序模块实现

5.1.1 数据库连接模块

数据库连接后用户才能够进行管理工作，此模块中主要填写，地址、用户名、密码、连接名称等信息。同时为了让用户快速定位问题所在，对于抛出的异常进行了语义分析，返回确切的错误原因。对于用户密码进行简单加密，避免明文存储的风险。

数据库连接模块部分代码如图 23、24 所示：

```
public Connection conn(String url,String user,String password)throws SQLException{
    String forward="jdbc:mysql://";String backward="?useSSL=false&serverTimezone=GMT&allowPublicKeyRetrieval=true";
    //Class.forName("com.mysql.jdbc.Driver");
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");//8以上的版本用这个驱动
    }catch (ClassNotFoundException e){
        e.printStackTrace();
        JOptionPane.showMessageDialog(null,"系统错误！ ",
            "请报告JKL4131@126.com",JOptionPane.ERROR_MESSAGE);
    }
    Connection conn=DriverManager.getConnection(forward+url+backward, user, password);
    return conn;
}

public ResultSet showDB(Connection conn)throws SQLException{
    String sql = "show databases;";
    ResultSet rs=null;
    PreparedStatement pst=(PreparedStatement)conn.prepareStatement(sql);
    rs = pst.executeQuery();
    return rs;
}

public ResultSet openDB(Connection conn,String dbName)throws SQLException{
    String sql="select table_name from information_schema.tables where table_schema='"+dbName+"'"; // and table_type='base table';
    ResultSet rs;
    PreparedStatement pst=(PreparedStatement)conn.prepareStatement(sql);
    rs=pst.executeQuery();
    return rs;
}
```

图 23 数据库连接数据访问代码

```
test.addActionListener(event->{
    String title="连接测试";
    try {
        StringBuilder psw=new StringBuilder();
        for(int i=0;i<password.getPassword().length;i++) {
            psw.append(password.getPassword()[i]);
        }
        MysqlConn mysqlConn = new MysqlConn();
        Connection tempConn;
        tempConn=mysqlConn.conn(url.getText(), name.getText(),psw.toString());
        JOptionPane.showMessageDialog(null, "连接成功", title, JOptionPane.PLAIN_MESSAGE);
        tempConn.close();
    }
    catch (SQLException e){
        if(e.getMessage().contains("Access denied for user")){
            JOptionPane.showMessageDialog(null, "用户名或密码错误",title,JOptionPane.ERROR_MESSAGE);
        }
        else{
            JOptionPane.showMessageDialog(null, "地址错误", title, JOptionPane.ERROR_MESSAGE);
        }
    }
});
```

图 24 数据库连接模块部分业务代码

使用 JDBC 进行数据库连接，反馈数据库结构，在主界面利用 swing 控件构建数据库结构树状图。异常处理方面对于 excuteQuery（）方法抛出的 SQL 异常进行分析，并反馈相应的异常信息。

5.1.2 数据管理模块

主要负责构建数据库结构树状结构，执行用户的数据库操作，让用户对于数据库进行简便快捷的管理。根据 MySQL 的参考手册中第八章第一节给出的优化总则^[17]，数据库级的优化需要首先考虑表结构的正确性，各属性的类型问题，所以在数据库管理模块加入了，表结构的管理功能。sql 执行的数据访问层代码如图 25 所示：

```
public CusRes excuteSql(Connection conn, String sql){
    CusRes cr=new CusRes();
    ResultSet rs=null;
    try {
        PreparedStatement pst=conn.prepareStatement(sql);
        rs=pst.executeQuery();
        cr.setCusRes(0,rs,null);
    }
    catch (SQLException ex) {
        StringWriter sw=new StringWriter();
        ex.printStackTrace(new PrintWriter(sw));
        if(sw.toString().contains("with executeQuery")){
            try {
                PreparedStatement pst=conn.prepareStatement(sql);
                pst.executeUpdate();
                cr.setCusRes(0,null,null);
            }catch (SQLException e){
                JOptionPane.showMessageDialog(null,e.getMessage(),
                    "Error",JOptionPane.ERROR_MESSAGE);
                cr.setCusRes(2,null,e);
            }
        }
        else{
            JOptionPane.showMessageDialog(null,ex.getMessage(),
                "Error",JOptionPane.ERROR_MESSAGE);
            cr.setCusRes(2,null,ex);
        }
    }
    return cr;
}
```

图 25 自定义 sql 执行数据访问代码

查询语句返回的结果，将会反馈至主界面上，对于数据表增加监听，使用户能够对数据进行热修改。同时对于会使数据表变成无主键状态的操作，做出警告，但是考虑到用户的操作需求不做阻止操作。表格体监听部分代码如图 26 所示：

```
table[tabIndex].getModel().addTableModelListener(new TableModelListener() {
    @Override
    public void tableChanged(TableModelEvent e) {
        if(e.getType()==TableModelEvent.UPDATE){
            confirmTool[tabIndex].setEnabled(true);
            cancelTool[tabIndex].setEnabled(true);
            sqlCount[tabIndex]++;
            Query query1=new Query();
            try {
                int pkCount=query1.queryPK(connecting[connectingCount], dbName, temp).length;
                String[] pkName=query1.queryPK(connecting[connectingCount], dbName, temp);
                if(pkCount!=0) {
                    updateSQL[sqlCount[tabIndex]] = "UPDATE " + dbName + "." + temp + " SET " + table[tabIndex].getColumnName(e.getColumn()) +
                        "=" + table[tabIndex].getValueAt(e.getLastRow(), e.getColumn()).toString() + " WHERE ";
                    updateSQL[sqlCount[tabIndex]] += pkName[0] + "=" + tempTable[tabIndex].getValueAt(e.getLastRow(), 0) + " ";
                    for (int i = 1; i < pkCount; ++i) {
                        updateSQL[sqlCount[tabIndex]] += "AND " + pkName[i] + "=" + tempTable[tabIndex].getValueAt(e.getLastRow(), i) + " ";
                    }
                    tempTable[tabIndex].setValueAt(table[tabIndex].getValueAt(e.getLastRow(), e.getColumn()), e.getLastRow(), e.getColumn());
                }
            }
            else{
                int result=JOptionPane.showConfirmDialog(null,"此表未设置主键，建议您设置主键，继续可能会引发更新错误，是否继续？",
                    "未设置主键!",JOptionPane.YES_NO_OPTION);
                if(result!=1){
                    updateSQL[sqlCount[tabIndex]] = "UPDATE "+dbName+"."+temp+" SET "+table[tabIndex].getColumnName(e.getColumn()) +
                        "=" + table[tabIndex].getValueAt(e.getLastRow(), e.getColumn()).toString() + " WHERE ";
                    updateSQL[sqlCount[tabIndex]] += table[tabIndex].getColumnName(0) + "=" + tempTable[tabIndex].getValueAt(e.getLastRow(), 0) + " ";
                    for(int i=1; i<table[tabIndex].getColumnCount(); ++i){
                        updateSQL[sqlCount[tabIndex]] += "AND "+table[tabIndex].getColumnName(i) +
                            "=" + tempTable[tabIndex].getValueAt(e.getLastRow(), i) + " ";
                    }
                    tempTable[tabIndex].setValueAt(table[tabIndex].getValueAt(e.getLastRow(), e.getColumn()), e.getLastRow(), e.getColumn());
                }
            }
        }
    }
})
```

图 26 数据表修改功能显示层代码

同时增加 sql 语句编辑的 MySQL 关键词识别功能，方便用户构建 sql 语句。关键词参照参考 MySQL 参考操作手册第 9 章第 3 节 Keywords and Reserved Words [17]。采用先识别再染色的逻辑，实现文本域接口中的插入方法，实现顺序为：使用双指针遍历 sql 文本，每次遇到关键分隔符（空格、星号、换行符、左括号）将两个指针之间的词与枚举类中的关键词进行比对，比对成功则进行染色工作。自定义 sql 的业务层代码如图 27 所示：

```
sql.getDocument().addDocumentListener(new DocumentListener() {
    @Override
    public void insertUpdate(DocumentEvent e) {
        StyledDocument doc=(StyledDocument) sql.getDocument();
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                doc.setCharacterAttributes(0, doc.getLength(),
                    sql.getStyle("DEFAULT"), true);
            }
        });
        int flag=0;
        for (int i = 0; i < sql.getDocument().getLength(); i++) {
            try {
                String nowChar=sql.getDocument().getText(i,1);
                if(nowChar.equals(" ")||nowChar.equals("\n")
                    ||nowChar.equals("(")||nowChar.equals(";")) {
                    String s = sql.getDocument().getText(flag, i - flag);
                    for (SqlKeywords skw : SqlKeywords.values()) {
                        if (skw.toString().equals(s) || skw.getLowWord(skw).equals(s)) {
                            int finalFlag = flag;
                            int finalI = i;
                            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                                @Override
                                public void run() {
                                    doc.setCharacterAttributes(finalFlag, finalI - finalFlag, sql.getStyle("BLUE"), true);
                                }
                            });
                        }
                    }
                    flag = i + 1;
                }
            } catch (BadLocationException ex) {
                ex.printStackTrace();
            }
        }
    }
});
```

图 27 MySQL 关键词识别业务层代码

5.1.3 性能评估模块

串行测试，将用户输入的用例进行处理之后，调用数据库管理模块的自定义 sql 执行功能，直接执行相应的命令，获得反馈信息。主线程直接调用函数方法。函数方法中使用 swing 的进度条组件来完成向用户反馈的工作。性能评估的主线程部分代码如图 28 所示：

```

@Override
public void run(){

    if(testType!=2) {
        OptFunction optFunction = new OptFunction();
        HashMap<Integer, String[]> res = optFunction.initTest(sql, conn, testType);
        if(!res.isEmpty()) {
            ParaEvaluation paraEvaluation = new ParaEvaluation();
            paraEvaluation.showRes(res, conn);
        }
    }else {
        multiSql();
        if(!stopFlag) {
            ParaEvaluation paraEvaluation = new ParaEvaluation();
            paraEvaluation.showRes(map, conn);
        }
    }
}
}

```

图 28 性能评估主线程部分代码

并行测试，使用多线程处理，根据输入的用例，分配多个子线程处理，阻塞主线程，等待全部完成后获得反馈。最后将反馈结果写入 HashMap 并进行返回。分配的子线程的触发与反馈代码如图 29 所示：

```

public void multiSql(){
    map=new HashMap<>();
    sumOfMulti=0;
    stopFlag=false;

    ArrayList<Thread> queue=new ArrayList<>();

    for(int j=0;j<sql.length;++j) {
        OptimizeThread opt=new OptimizeThread(conn,sql[j],j);
        queue.add(opt);
        opt.start();
    }
    try{
        for(Thread t:queue){
            t.join();
        }
    }catch (InterruptedException e){
        e.printStackTrace();
    }

    int avg=sumOfMulti/sql.length;

    EvaluationIO eio=new EvaluationIO();
    String[] t=eio.readLast();

    for(int i=0;i<4;++i) {
        String[] temp={t[i],avg+"",t[4]};
        map.put(i,temp);
    }
}
}

```

图 29 子线程触发与反馈

5.1.4 压力测试模块

用户填写参数，利用 JSON 传输数据，功能层整合成 cmd 命令执行。将结果以及命令存储至日志中，隐藏密码部分。考虑到 MyISAM 引擎不能在表损坏后恢复数据，MyISAM 引擎只适用于纯查询的业务场景^{[4][20]}，所以选用 InnoDB 作为 mysqlslap 的默认测试引擎。

压力测试在进行日志存储的过程中会注意到用户敏感数据的问题，对于数据库密码这种用户敏感数据进行抹去操作，不同于用户连接模块的数据库密码，在压力测试中的命令不需要进行复用，所以不采用任何加密方式，而是直接抹去。slap 日志记录的部分代码见图 30：


```

public void writelog(String text,String cmd) throws IOException {
    StringBuilder path= Location.Path.getPath();
    path.append("classes/LunaLOG");
    File file = new File(path.toString());
    File file1 = new File(path.toString() +"/slap.log");
    if(!file.exists()){
        file.mkdirs();
    }
    if(!file1.exists()){
        file1.createNewFile();
    }

    StringBuilder nonPwdCmd=new StringBuilder();
    char[] chars=cmd.toCharArray();
    boolean flag=false;boolean first=true;
    for(int i=0;i<chars.length-1;i++){
        if(flag){
            if(chars[i]!=' ') nonPwdCmd.append("*");
            else {flag=false;nonPwdCmd.append(chars[i]);}
        }else{
            nonPwdCmd.append(chars[i]);
        }
        StringBuilder temp=new StringBuilder();
        temp.append(chars[i]).append(chars[i+1]);
        if(temp.toString().equals("-p")&&first) {flag=true;first=false;}
    }
    nonPwdCmd.append(chars[chars.length-1]);
    Writer out = new FileWriter(file1, true);
    out.write(text + "\r\n" + "command:"+ nonPwdCmd.toString());
    out.close();
}

```

图 30 记录 slap 日志

5.1.5 sql 优化模块

用户输入参数，通过 JSON 将数据传输至功能层整合命令，执行命令，反馈优化结果。将结果以及命令存储至日志中，隐藏密码部分。优化功能的设计是针对最常用的 sql 优化需求，由于非递归 join 子句并不适用于线性递归 sql 查询，也就意味着需要对 join 子句进行优化^[4]。所以需要通过添加 EXPLAIN 语句分析功能来解读 sql 语句结构来优化连接操作。优化命令的执行代码如图 31 所示：

```

public String excuteCmd(String cmd,String type){
    StringBuilder res=new StringBuilder();
    SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    res.append("\r\n").append("=".repeat(20)).append(df.format(new Date())).append("\r\n");
    try{
        Process p = Runtime.getRuntime().exec("cmd.exe /c "+cmd);
        //win7 linux
        //Process p = Runtime.getRuntime().exec(cmd);

        BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream(),StandardCharsets.UTF_8));
        String line="";
        while ((line = br.readLine()) != null) {
            res.append(line).append("\n");
        }
        br.close();
        if(type.equals("soar")) {
            SoarLog sl = new SoarLog();
            sl.writeLog(res.toString(), cmd);
        }else{
            SlapLog sl=new SlapLog();
            sl.writeLog(res.toString(),cmd);
        }
    }catch (IOException e){
        e.printStackTrace();
    }
    return res.toString();
}

```

图 31 优化命令执行

6. MySQL 管理优化系统测试

本节主要阐述了系统的部署测试与实际运行测试过程。为了展示系统运行的稳定性、安全性与可用性。在第一小节中对于本地环境进行部署测试，而在第二小节中则会通过一个用例来展示系统的优化功能是如何工作的。

6.1 运行测试

在 Windows10 环境下，JRE 版本: OpenJDK Runtime Environment (build 14+36-1461)，`java -jar Luna-SQL.jar` 命令直接运行 jar 文件，运行正常。利用 WebSwing 框架将软件部署在 web 环节中，Java 版本为 openJDK 11。程序运行正常。图 32 为 WebSwing 的本地运行代码：


No. (id)	App	User	IP	Start time ▼	Client status	Metrics (min avg max)	Bandwidth (min avg max)	Latency (min avg max)
1	My Application	admin 	127.0.0.1	28 Mar 4:49:54	Connected Record Session	MEM: 54MB(0 55 82) CPU: 0% (0 0 5)	IN: 0k/s(0 0 0) OUT: 56k/s(0 56 56)	E2E: 102ms(0 111 133) PING: 13ms(0 20 36)

图 32 本地测试 WebSwing 后台参数

6.2 优化测试

通过用例来测试优化功能。

用例: `select*from app a join review b on a.App=b.App where a.Rating=4.0`
app 表存储的是谷歌商店的应用信息，review 表则是应用的评论信息。

压力测试测试结果（50 条并发，每条执行 2 次）：

Average number of seconds to run all queries: 0.656 seconds

Minimum number of seconds to run all queries: 0.656 seconds

Maximum number of seconds to run all queries: 0.656 seconds

Number of clients running queries: 50

Average number of queries per client: 2

平均每次 0.013 秒

性能评估结果（100 次，平均每次 0.028 秒），具体结果见图 34：

评估结果			
若使用的是默认模式:		所有值相同	
参数名称	用例测试时间(ms)	较上次测试变化(ms)	上次测试时间
join_buffer_size	28	(-206)	2020-03-11 17:11:16
read_buffer_size	28	(-206)	2020-03-11 17:11:16
sort_buffer_size	28	(-206)	2020-03-11 17:11:16
key_buffer_size	28	(-206)	2020-03-11 17:11:16
参数值可在历史记录中查询		查看历史记录	

图 33 测试用例评估结果

* ****ALL****: 最坏的情况, 从头到尾全表扫描.

* **Using join buffer**: 从已有连接中找被读入缓存的数据，并且通过缓存来完成与表的连接。

为 test 库的 app 表添加索引

* **Severity:** L2

```

* **Case:** ALTER TABLE `test`.`app` add index `idx_App` (`App`
(191) ) ;

```

编辑App结构

字段	字段类型	字段长度	PK	默认是否为空	索引名	是否为唯一索引
Android Ver	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
App	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Category	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Content Rating	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Current Ver	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Genres	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Installs	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Last Updated	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Price	INT	11	<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Rating	DOUBLE		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Reviews	INT	11	<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Size	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>
Type	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	删除字段	<input type="checkbox"/>

提交更改 新建字段

图 34 app 表的表结构管理界面

在增加索引时需要注意索引的数量并不是越多越好,实际上需要不断的通过测试观察效果^[21]。并不能盲目的添加索引。索引过多容易造成插入和更新语句的效率降低。对拥有聚合索引的属性的进行更新操作效率也较低^[22],所以在设计索引时需要谨慎。

综上所述，接着再进行测试观察效果，首先是压力测试结果：

Number of clients running queries: 50

Average number of queries per client: 2

可以看出 50 条用例的完成时间为 0.453 秒,每次只需要 0.009 秒,相较上次提升了 0.004 秒,提升了 40%。再看性能评估的结果,如图 34 所示,由于做了比较处理,可以很明显的看出,较上次速度提高了 10ms,时间提升了 35%。



评估结果

若使用的是默认模式: 所有值相同

参数名称	用例测试时间(ms)	较上次测试变化(ms)	上次测试时间
join_buffer_size	18	(-10)	2020-03-28 21:16:00
read_buffer_size	18	(-10)	2020-03-28 21:16:00
sort_buffer_size	18	(-10)	2020-03-28 21:16:00
key_buffer_size	18	(-10)	2020-03-28 21:16:00

参数值可在历史记录中查询

查看历史记录

图 35 优化后的测试用例评估结果

7. 结论

7.1 创新与不足

本系统通过将现有的数据库优化功能整合,并设计为图形化程序,从而让基础的优化工作摆脱命令行工具,使得用户能够快速的进行数据库优化工作,减少学习周期,降低优化数据库的工作负荷。同时从功能上来说,系统整合了数据库管理工具的功能,让能够使用户通过一个工具同时完成数据库的基础优化与管理工作,通过数据库管理工具不同连接不同管理的特性,使得不同的数据库的优化信息可以在一个工具中实现。

但是本系统主要还是对现有工具的整合,弥补现有工具的一些不足,并没有提出优化的算法,也没有实现数据库参数的自动调优,主要的优化重心仍然放在 sql 语句的优化上。在数据库管理方面则缺失了一些功能,只能够满足基本的操作。综上所述,本系统仍有许多需要完善的地方。

7.2 未来展望

关于数据库优化的研究依然有许多未解决的问题。最为重要的仍然是数据库参数的自动调优,包括了对数据库物理层面上的设置调整。在未来的研究工作中,需要更多的人工智能,机器学习的算法,例如 sklearn 等技术来支持优化工具的开发^[23],能够让具有普适性的优化工具经过机器学习适应各种业务场景,同时也能实现硬件层的参数优化。同时在大规模数据的存储优化方面,例如对于 hdfs 集群等分布式文件系统,或者新一代大规模关系型数据库^[24]^[25]等数据库中的优化方案,仍然有许多的问题需要解决。

参考文献

- [1]张驰曦. 一个游戏系统中的数据库优化技术研究与应用[D]. 复旦大学, 2012.
- [2] Dana Van Aken Andrew Pavlo Geoffrey J. Gordon Bohan Zhang. Automatic Database Management System Tuning Through Large-scale Machine Learning[J]. SIGMOD '17: Proceedings of the 2017 ACM International Conference on Management of Data May 2017:1009–1024
- [3]孙辉. MySQL 查询优化的研究和改进[D]. 华中科技大学, 2007.
- [4] Ordonez, Carlos. Optimization of Linear Recursive Queries in SQL[J]. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2010, 22(2):264-277
- [5]Zhou Huang , Yiran Chen , Lin Wan and Xia Peng. GeoSpark SQL: An Effective Framework Enabling Spatial Queries on Spark[J]. International Journal of Geo-Information 2017(6):285
- [6]梁水德. 医院数据库系统性能优化对策分析[J]. 中国新通信, 2020,22(03):62.
- [7]崔锴. 高校访问境外数据库优化设计[J]. 电子技术与软件工程, 2020(02):161-164.
- [8]王超. 网络数据库安全管理技术的优化策略[J]. 无线互联科技, 2020,17(01):22-23.
- [9]林清树. 高校计算机数据库安全管理现状及优化探讨[J]. 科技经济导刊, 2019,27(36):185.
- [10]黄卓洲. 浅谈 SQL 数据库优化技术在信息管理系统中的应用[J]. 中国新通信,2020,22(02):104.
- [11]张捷. 分布式数据库查询处理和优化算法[J]. 电子测试, 2019(24):66-67+34.
- [12]张家旭. ERP 系统数据库设计优化[J]. 电子技术与软件工程, 2019(15):137-138.
- [13] GNU General Public License v3.0[M] . Free Software Foundation, Inc. 2007.
- [14]彭影. 优化 Java 数据库访问效率的策略研究[J]. 计算机产品与流通, 2020(02):21.
- [15]Webswing Documentation v2.7[M] , Webswing Ltd. 2019.
- [16]刘阳娜. 大数据下的 MySQL 数据库的效率优化[J]. 信息通信, 2017(12):111-112.
- [17]MySQL 8.0 Reference Manual [M], Oracle Corporation and/or its affiliates. 2020.
- [18]谷伟, 陈莲君. 基于 MySQL 的查询优化技术研究[J]. 微型电脑应用, 2013,30(07):48-50.
- [19]刘冬晗, 杨亮. ERP 系统运行中的数据库优化探讨[J]. 化工管理, 2020(07):117-118.
- [20]吴沧舟, 兰逸正, 张辉. 基于 MySQL 数据库的优化[J]. 电子科技,2013,26(09):182-184.
- [21]刘华清, 陈振东, 涂刚. 数据库索引与优化[J]. 现代计算机(专业版), 2012(18)
- [22]张昊钻, 邢小平. SQL 数据查询语句的优化研究[J]. 电脑编程技巧与维护, 2019(09):104-105+120.
- [23]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python[J]. Journal of Machine Learning Research, 2011(12):2825–2830.
- [24]Moniruzzaman, A. Newsql: Towards next-generation scalable rdbms for online transaction processing (oltp) for big data management[J]. Int. J. Database Theory Appl. 2014(7):121–130.
- [25]叶云霜, 林伟华, 刘福江, 董晓莹. 关系数据库中海量要素存储的分区优化研究[J]. 计算机技术与发展, 2020,30(01):167-173.

致谢

首先感谢李敏老师在繁忙的工作中对我的悉心指导。李老师对于论文提出的指导意见使我受益匪浅，为论文的成文提供了极大的帮助。还要感谢本科学习期间的信管系的各位任课老师，他们传授给我的专业知识是我毕业设计的基础。

其次要感谢和我一起交流想法的各位同学，和他们的交流给我提供了许多的设计灵感。

最后还要感谢论文中所用到的开源项目的组织以及开发人员。感谢每一位开发者做出的卓越的贡献，使我作为一位普通的开发者能够更为自由轻松的进行开发工作。