

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ
Н. Э. БАУМАНА»
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
(МГТУ им. Н.Э.Баумана)



ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»
КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ
ТЕХНОЛОГИИ»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:**

**АВТОМАТИЗАЦИЯ ПОДБОРА ИНДЕКСОВ ДЛЯ SQL
ЗАПРОСОВ В СУБД MYSQL**

Студент ИУ7-81	_____	И. С. Петухов
Руководитель ВКР	_____	Е. А. Просуков
Консультант	_____	_____
Консультант	_____	_____
Нормоконтролер	_____	_____

Москва, 2017

Содержание

Реферат	4
Введение	6
1 Аналитический раздел	8
1.1 Современные СУБД	8
1.1.1 Oracle	9
1.1.2 MS SQL Server	10
1.1.3 Azure SQL Database	12
1.1.4 PostgreSQL	13
1.1.5 MySQL	14
1.2 Результаты анализа возможностей СУБД и их инструментов для индексирования	16
1.3 Выводы	16
2 Конструкторский раздел	19
2.1 Индексы	19
2.2 Подсистемы хранения MySQL	19
2.2.1 Подсистема MyISAM	20
2.2.2 Подсистема InnoDB	20
2.3 Типы индексов	20
2.3.1 B-tree-индексы	20
2.3.2 Хеш-индексы	22
2.3.3 Полнотекстовые индексы	22
2.4 B-tree индексы в MySQL InnoDB	23
2.4.1 Кластерный индекс	23
2.4.2 Вторичный индекс	23
2.4.3 Составной индекс	24
2.4.4 Частичный индекс	24
2.4.5 Селективность индексов	24
2.4.6 Покрывающий индекс	25
2.5 Индексы для сложных запросов	25
2.6 Определение индексов для сложных запросов	26
2.6.1 Определение таблицы для полного сканирования	26
2.6.2 Отбрасывание лишних полей в блоке сортировки	28
2.6.3 Разбиение сложного запроса	29
2.6.4 Построение индекса для простого запроса	29
2.6.5 Объединение индексов подзапросов	32
2.7 Объектно-ориентированный анализ	32
2.7.1 Поле	33

2.7.2	Аргумент	33
2.7.3	Условное выражение	33
2.7.4	Выражение блока ON	33
2.7.5	Выражение блока ORDER BY	34
2.7.6	Индекс	34
2.7.7	Простой запрос	34
2.7.8	Сложный запрос	35
2.7.9	Диаграмма классов	35
2.8	Тестирование разрабатываемого приложения	36
3	Технологический раздел	40
3.1	Выбор средств программной реализации	40
3.2	Сборка инструмента	40
3.3	Руководство пользователя	40
4	Экспериментальный раздел	43
4.1	Результаты экспериментов	44
4.2	Выводы	47
	Заключение	49
	Список использованных источников	50

Реферат

Обозначения

АБД	Администратор базы данных
БД	База данных
ПО	Програмное обеспечение
СУБД	Система управления базой данных

Введение

Рост количества хранимых данных и уникальных пользователей вынуждает разработчиков искать пути обеспечения высокого качества функционирования сервисов и низкого времени отклика пользовательского интерфейса. Одной из главных причин низкой скорости обработки запросов является работа с БД, которая не оптимизирована. [1]

Для АБД и разработчиков приложений настройка приложений является критически важной задачей, и они тратят значительное время на ее выполнение. Плохо настроенное бизнес-приложение может потенциально повлиять не только на нескольких пользователей, но и на всю операционную деятельность организации, поэтому компании вкладывают значительные ресурсы, чтобы обеспечить хорошую работу приложений, критически важных для их бизнеса.[2]

Ручная настройка SQL предоставляет собой очень сложный и проблемный процесс. Он требует глубокого знания в различных областях, отнимает много времени, требует детального знания структуры данных и модели использования данных приложения. Все эти факторы делают процесс ручной настройки SQL сложной и ресурсоемкой задачей, которая в конечном счете обходится очень дорого для бизнеса. [2]

Оптимизация производительности сводится к следующим задачам:

- а) корректировка параметров СУБД;
- б) денормализация данных;
- в) выявление медленных запросов и их анализ:
 - 1) корректировка запросов;
 - 2) создание индексов.

Каждый из этих пунктов может стать темой отдельного исследования на определенных наборах данных, однако среднестатистическому разработчику необходимы рецепты, позволяющие быстро обойти проблемы производительности без досконального изучения документации по СУБД и сосредоточить свое внимание на бизнес логике приложения. [1]

Основным приемом увеличения производительности выполнения запросов к базе данных является индексирование. Индексы представляют собой структуры данных, которые помогают СУБД эффективно извлекать данные. Они критичны для достижения хорошей производительности, но многие часто забывают о них или плохо понимают их смысл, поэтому индексирование является главной причиной проблем с производительностью в реальных условиях. [3]

В данной работе проводится обзор самых популярных СУБД и существующих для них инструментов для администрирования, а в частности индексирования.

Целью работы является разработка инструмента для облегчение работы администраторов СУБД MySQL по созданию индексов для SQL запросов.

Для достижения поставленной цели для выбранной СУБД необходимо изучить:

- а) доступные инструменты для анализа выполнения запросов
- б) структуру хранения данных
- в) типы используемых индексов
- г) правила использования индексов
- д) работу оптимизатора запросов

На основе этой информации разработать инструмент, пригодный для использования АБД. Для этого нужно продумать и реализовать взаимодействие с пользователем.

Также необходимо предусмотреть возможность дальнейшей автоматизации процесса индексирования БД.

1 Аналитический раздел

В данном разделе проводится обзор современных СУБД и анализ их инструментов для управления индексами.

1.1 Современные СУБД

Рассмотрим самые популярные СУБД. [1.1](#)

Таблица 1.1 — Рейтинг СУБД

название	год	sql	разработчик
MySQL	1995	да	Oracle
PostgreSQL	1995	да	сообщество
MS SQL Server	1988	да	Microsoft
MongoDB	2009	нет	MongoDB
SQLite	2000	да	Hwaci, сообщество
Oracle Database	1979	да	Oracle
Firebird	2000	да	сообщество
CouchDB	2005	нет	Apache
DB2	1995	да	IBM
MariaDB	2009	да	MariaDB Corporation Ab, MariaDB Foundation, сообщество
RavenDB	2009	нет	Hibernating Rhinos
Redis	2009	нет	Redis Labs, сообщество
SAP ASE (ex: Sybase)	1988	да	SAP AG
Percona Server	2006	да	Percona

Рейтинг СУБД выпускается Тэглайном впервые и сформирован на основе анкетирования (проводилось с августа 2014 по апрель 2016 года) 390 digital-агентств с продакшном и/или клиентским офисом в России: респондентам предлагалось выбрать один или несколько вариантов ответа на вопрос «Укажите СУБД, которые вы используете при разработке проектов». [4]

Большинство реляционных СУБД, за исключением MS Access, состоят из двух отдельных компонентов: «back-end», где хранятся данные и «front-end» — пользовательский интерфейс для взаимодействия с данными. Этот тип конструкции достаточно умный, так как он распараллеливает двухуровневую модель программирования, которая отделяет слой данных от пользовательского интерфейса и позволяет сконцентрировать рынок ПО непосредственно на улучшении своих продуктов. Эта модель открывает двери для третьих сторон, которые создают свои приложения для взаимодействия с различными базами данных. [5]

Данный подход позволяет сторонним разработчикам создавать инструменты для администрирования СУБД. Рассмотрим некоторые СУБД и предлагаемые инструменты для их администрирования, а в частности, индексирования более подробно.

1.1.1 Oracle

Oracle Database – СУБД, ориентированная на применение в корпоративных сетях распределенной обработки данных (Enterprise Grid), в облачных системах (Cloud Computing), а также для построения корпоративных информационных систем. Она позволяет сократить расходы на информационные технологии благодаря автоматизации управления, использованию недорогих модульных компонентов и кластеризации серверов в целях эффективного использования ресурсов.

Архитектура СУБД Oracle рассчитана на работу с огромными объемами данных и большим (десятки и сотни тысяч) числом пользователей; она демонстрирует широкие возможности обеспечения высокой готовности, производительности, масштабируемости, информационной безопасности и самоуправляемости. СУБД Oracle может быть развернута на любой платформе, начиная от небольших серверов-лезвий и заканчивая симметричными многопроцессорными компьютерами и мейнфреймами. Уникальная способность СУБД Oracle работать со всеми типами данных, от традиционных таблиц до XML-документов и картографических данных, позволяет рассматривать ее в качестве оптимального выбора для работы с приложениями оперативной обработки транзакций, поддержки принятия решений и управления коллективной работой с информацией.

Oracle Tuning Pack – дополнительная опция для управления Oracle Database, наиболее эффективное и легкое в использовании решение, которое полностью автоматизирует процесс настройки приложений. Улучшение производительности SQL достигается с помощью мониторинга выполнения SQL в реальном времени и SQL-советников, интегрированных с Oracle Enterprise Manager Cloud Control 12c, и все это вместе предоставляет всестороннее решение для сложной и требующей много времени задачи по настройке приложений.

SQL Tuning Advisor является ответом Oracle на все недостатки и проблемы ручной настройки SQL. Он автоматизирует процесс настройки SQL путем всестороннего исследования всех возможных вариантов настройки SQL-предложения. Анализ и настройка осуществляются с помощью существенно улучшенного оптимизатора запросов, встроенного в ядро базы данных.

SQL Tuning Advisor проводит шесть типов анализа:

а) Анализ статистики: выявление объектов с отсутствующей или устаревшей статистикой, выдача соответствующих рекомендаций по устранению проблемы.

б) SQL-профилирование: Эта возможность, появившаяся в Oracle Database 10g, революционизировала подход к настройке SQL. SQL-профилирование позволяет настраивать SQL-предложения без каких-либо изменений кода приложения.

в) Анализ путей доступа: Во время этого анализа определяются новые индексы, которые могут значительно улучшить производительность запросов.

г) Анализ структуры SQL: Здесь проверяется неявное преобразование типов и даются рекомендации по изменению кода SQL.

д) Степень параллелизма: SQL Tuning Advisor определяет, можно ли улучшить время выполнения с помощью параллельных потоков на определенных этапах выполнения SQL.

е) Альтернативные планы: Во время этого анализа SQL Tuning Advisor находит другие планы выполнения запроса, используя текущие и исторические данные производительности.

Он всесторонне анализирует всю нагрузку и дает рекомендации по созданию новых секций таблицы или индексов, удалению неиспользуемых индексов, созданию новых материализованных представлений и журналов. Определение оптимальной стратегии секционирования или индексирования для конкретной нагрузки является сложным процессом, требующим опыта и времени. SQL Access Advisor учитывает стоимость операций ввода/обновления/удаления в дополнение к запросам и дает соответствующие рекомендации, сопровождаемые количественной мерой ожидаемого выигрыша в производительности, а также скрипты, необходимые для реализации этих рекомендаций.

Настройка SQL-предложений больше не является прерогативой только специалистов. Oracle встроила эксперта по настройке SQL в ядро базы данных, позволив администраторам баз данных выполнять эту очень важную задачу за доли времени и затрат, необходимых для выполнения той же задачи вручную.

[2]

1.1.2 MS SQL Server

Данный программный продукт представляет собой СУБД реляционного типа, разработанную корпорацией Microsoft. Для манипуляции данными используется специально разработанный язык Transact-SQL. Команды языка для выборки и модификации базы данных построены на основе структурированных запросов.

СУБД является частью длинной цепочки специализированного программного обеспечения, которое корпорация Microsoft создала для разработчиков. А это значит, что все звенья этой цепи (приложения) глубоко интегрированы между собой. То есть их инструментарий легко взаимодействует между собой, что во многом упрощает процесс разработки и написания программного кода. Примером такой вза-

вместе является среда программирования MS Visual Studio. В ее инсталляционный пакет уже входит SQL Server Express Edition. [6]

Database Engine Tuning Advisor

Помощник Database Engine Tuning Advisor анализирует рабочую нагрузку и выдает рекомендации по физической структуре одной или нескольких баз данных. Анализ содержит рекомендации по добавлению, удалению или модификации физических структур баз данных, таких как индексы, индексированные представления или секции. Помощник Database Engine Tuning Advisor рекомендует набор физических структур базы данных, которые оптимизируют задачи, входящие в рабочую нагрузку.

Рекомендации помощника Database Engine Tuning Advisor, связанные с физическими структурами, можно также просматривать, используя ряд отчетов, которые предоставляют информацию о некоторых представляющих значительный интерес опциях. Эти отчеты позволяют увидеть, как помощник Database Engine Tuning Advisor выполнял оценку рабочей нагрузки. Для просмотра доступны следующие отчеты:

а) Index Usage Report (recommended configuration) (отчет по использованию индексов (рекомендуемая конфигурация)) — содержит информацию об ожидаемом использовании рекомендуемых индексов и их предполагаемых размерах;

б) Index Usage Report (current configuration) (отчет по использованию индексов (текущая конфигурация)) — предоставляет ту же самую информацию, что и предыдущий отчет, но для текущей конфигурации;

в) Index Detail Report (recommended configuration) (подробный отчет по индексам (рекомендуемая конфигурация)) — содержит информацию об именах всех рекомендуемых индексов и их типах;

г) Index Detail Report (current configuration) (подробный отчет по индексам (текущая конфигурация)) — предоставляет ту же самую информацию, что и предыдущий отчет, но для фактической конфигурации до начала процесса настройки;

д) Table Access Report (отчет о доступе к таблицам) — предоставляет информацию о затратах всех запросов в рабочей нагрузке (используя таблицы базы данных);

е) Workload Analysis Report (отчет анализа рабочей нагрузки) — предоставляет информацию об относительных частотах всех инструкций по модификации данных (затраты подсчитываются относительно наиболее затратной инструкции при текущей конфигурации индексов).

Эти рекомендации можно использовать тремя способами: немедленно, по расписанию или после сохранения в файл. [7]

1.1.3 Azure SQL Database

SQL Azure – проекция традиционного SQL Server на облако, предоставляющая возможности для работы с базой данных посредством интернет-сервисов. Эта технология позволяет хранить структурированную и неструктурированную информацию, исполнять реляционные запросы, а также предоставляет функционал для осуществления поиска, создания аналитических отчетов, интеграции и синхронизации данных. На данный момент SQL Azure поддерживает сервис реляционных баз данных, имеющий название SQL Azure Database.

SQL Azure Database – облачная платформа реляционной базы данных, построенная на технологиях SQL Server. При использовании этой платформы можно легко построить в облаке проект реляционной базы данных со всеми преимуществами, предоставляемыми любой облачной технологией. Кроме того, SQL Azure предоставляет высокий уровень безопасности со встроенной защитой данных, самовосстановлением и системой резервного копирования. Хотя SQL Azure и базируется на технологиях SQL Server, он представляет такие новые возможности, как высокий уровень масштабируемости, постоянная доступность и самоуправление, предоставляя клиентам легкие и удобные способы работы посредством сети Интернет, не требуя при этом особых навыков или знаний, отличных от применимых с технологиями традиционного SQL Server. [8]

Index Advisor

Используя помощник по базам данных SQL Azure на портале Azure можно просматривать и реализовывать рекомендации для существующих баз данных SQL, которые могут повысить текущую производительность запросов. На странице рекомендаций приводится список основных предлагаемых рекомендаций с учетом их потенциального влияния на повышение производительности.

Помощник по работе с базами данных SQL можно настроить на автоматическое выполнение рекомендаций. В этом случае появляющиеся рекомендации будут применяться автоматически. Как и во всех остальных операциях с индексами, управляемых службой, если выполнение рекомендации ведет к ухудшению производительности, она отменяется.

Помощник по работе с базами данных SQL предоставляет рекомендации по повышению производительности базы данных SQL. Предоставляя сценарии T-SQL, а также параметры индивидуального или полностью автоматизированного управления (в настоящее время только для индексов), помощник помогает оптимизировать базу данных и, таким образом, повысить производительность запросов. [9]

Index Advisor поможет найти недостающие индексы (а так же предложит удалить ненужные) и улучшить производительность базы данных.

1.1.4 PostgreSQL

PostgreSQL это мощная объектно-реляционная система управления базами данных с открытыми исходными текстами. Она разрабатывается на протяжении более 15 лет и улучшает архитектуру, чем завоевала репутацию надежной, ингерированной и масштабируемой СУБД. Она запускается на всех основных платформах, включая Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), и Windows. Она полностью соответствует ACID, имеет полную поддержку ключей, объединений, представлений, триггеров, и хранимых процедур (на разных языках). Она включает большинство типов данных SQL92 и SQL99, включая INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, и TIMESTAMP. Она также поддерживает хранение больших двоичных объектов (BLOB's), включая картинки, звук, или видео. Она имеет API для C/C++, Java, Perl, Python, Ruby, Tcl, ODBC.

Являясь СУБД класса предприятия, PostgreSQL предоставляет такие особенности как Multi-Version Concurrency Control (MVCC), восстановление по точке во времени, табличное пространство, асинхронная репликация, вложенные транзакции (точки сохранения), горячее резервирование, планировщик/оптимизатор запросов, и упреждающее журналирование на случай поломки. Он поддерживает международные кодировки, в том числе и многобайтовые, при использовании различных кодировок можно использовать сортировку и полнотекстовый поиск, различать регистр. Большое количество подконтрольных данных и большое число одновременно работающих пользователей, тем не менее, не сильно влияет на масштабируемость системы. Есть действующие PostgreSQL системы, которые управляют более чем 4 терабайтами данных. [10]

Index Advisor

Утилита *Index Advisor* помогает определить, какие столбцы следует индексировать, чтобы повысить производительность в заданной рабочей нагрузке. *Index Advisor* рассматривает типы индексов B-tree (одностолбцовые или составные) и не идентифицирует другие типы индексов (GIN, GiST, Hash), которые могут повысить производительность. *Index Advisor* устанавливается вместе с *Postgres Plus Advanced Server*.

Index Advisor работает с планировщиком запросов *Advanced Server*, создавая гипотетические индексы, которые планировщик запросов использует для расчета затрат на выполнение, как если бы такие индексы были доступны. *Index Advisor* определяет индексы, анализируя SQL-запросы, поставляемые в рабочей нагрузке.

Один из способов использования *Index Advisor* для анализа SQL-запросов, это вызов служебной программы, предоставив текстовый файл, содержащий SQL-запросы, которые необходимо проанализировать; *Index Advisor* сгенерирует текстовый файл с инструкциями *CREATE INDEX* для рекомендуемых индексов.

В ходе анализа *Index Advisor* сравнивает затраты на выполнение запроса с гипотетическими индексами и без них. Если стоимость исполнения с использованием гипотетического индекса меньше стоимости исполнения без него, что сообщается в выводе инструкции EXPLAIN, где вычисляются показатели, которые оценивают улучшение, то *Index Advisor* генерирует инструкцию *CREATE INDEX*, необходимую для создания индекса.

Index Advisor фактически не создает индексы в таблицах. необходимо использовать инструкции *CREATE INDEX*, предоставляемые *Index Advisor*, чтобы добавить в таблицы все рекомендуемые индексы.

Pg_advice_index - это служебная программа, которая считывает предоставленный пользователем входной файл, содержащий SQL-запросы, и создает текстовый файл, содержащий инструкции *CREATE INDEX*, которые можно использовать для создания индексов, рекомендованных *Index Advisor*.

1.1.5 MySQL

Одной из самых распространенных реляционных СУБД является MySQL. Она кроссплатформенна, свободно распространяется по лицензии GNU и такими компаниями как Google, Adobe, Facebook и др. [1]

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

MySQL портирована на большое количество платформ: AIX, BSDi, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, SGI IRIX, Solaris, SunOS, SCO OpenServer, UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Server 2003, WinCE, Windows Vista, Windows 7 и Windows 10. Существует также порт MySQL к OpenVMS. Важно отметить, что на официальном сайте СУБД для свободной загрузки предоставляются не только исходные коды, но и откомпилированные и оптимизированные под конкретные операционные системы готовые исполняемые модули СУБД MySQL.[11]

EXPLAIN

Если оператор SELECT предваряется ключевым словом EXPLAIN, MySQL сообщит о том, как будет производиться обработка SELECT, и предоставит информацию о порядке и методе связывания таблиц.

При помощи EXPLAIN можно выяснить, когда стоит снабдить таблицы индексами, чтобы получить более быструю выборку, использующую индексы для поис-

ка записей. Кроме того, можно проверить, насколько удачный порядок связывания таблиц был выбран оптимизатором.

Для непростых соединений EXPLAIN возвращает строку информации о каждой из использованных в работе оператора SELECT таблиц. Таблицы перечисляются в том порядке, в котором они будут считываться. [12]

Профайлер

Директива *SHOW PROFILES* показывает список запросов, выполненных в рамках текущей сессии и время выполнения каждого запроса. Директива *SHOW PROFILE* показывает подробную информацию об этапах выполнения отдельного запроса. По умолчанию, выводится информация о последнем запросе.

Percona Toolkit

Штатные инструменты поставляемые с MySQL предоставляют лишь базовые возможности по администрированию, в результате многие операции приходится выполнять вручную. Это может быть проблемой, ведь уследить за всем очень сложно и часто потребуется определенный опыт, да и легко допустить ошибку. Пакет Percona Toolkit for MySQL собрал наработки двух проектов Maatkit и Aspensa и предоставляет скрипты позволяющие производить многие рутинные операции администрирования: [13]

а) pt-archiver - архивирует записи из одной таблицы в другую, можно задать условие

б) pt-deadlock-logger - логирует информацию о мертвых блокировках

в) pt-diskstats - мониторинг загрузки дисков

г) pt-duplicate-key-checker - находит дубликаты индексов в базе

д) pt-fk-error-logger - логирует информацию об ошибках внешних ключей

е) pt-heartbeat - мониторинг задержек репликации

ж) pt-index-usage - читает запросы из логов и анализирует как используются индексы

з) pt-kill - убивает запросы, подходящие под те или иные условия.

и) pt-mysql-summary - суммарная информация о сервере MySQL.

к) pt-online-schema-change - изменение таблицы без блокировки - создает пустую таблицу - делает все изменения, затем переносит данные в новую таблицу и переименовывает ее.

л) pt-query-digest - анализирует запросы из slow.log или из processlist

м) pt-show-grants - показывает доступы всех пользователей.

н) pt-table-usage - анализирует запросы из лога и как они используют таблицы.

о) pt-variable-advisor - анализирует переменные MySQL и выдает советы по возможным проблемам

п) pt-visual-explain - форматирует вывод EXPLAIN в виде дерева

1.2 Результаты анализа возможностей СУБД и их инструментов для индексирования

Результаты проведенного анализа представлены в таблице 1.2

Таблица 1.2 — Возможности СУБД их инструментов для индексирования

Возможность	Oracle	MS SQL Server и Azure SQL Database	PostgreSQL	MySQL
автоматический процесс создания индексов	да	да	нет	нет
автоматический процесс удаления индексов	да	да	нет	нет
отчет о рекомендуемых для создания индексов	да	да	да	нет
отчет о рекомендуемых для удаления индексов	да	да	да	да
отчет о найденных дубликатах индексов	да	да	да	да
анализ использования индексов	да	да	да	да

1.3 Выводы

Из результатов проведенного анализа видно, что в одной из самых популярных СУБД MySQL отсутствует возможность получения отчетов о рекомендациях по созданию индексов, что становится причиной отсутствия в данной СУБД инструмента по автоматизированному управлению индексами (создание, применение, удаление).

На основе выполненного анализа обоснована необходимость разработки нового ПО для рекомендации создания индексов для СУБД MySQL, что подтверждается

многочисленными вопросами о поиске данного инструмента и ответами о его отсутствии. [15, 16, 17, 18, 19]

Для процесса разработки такого инструмента необходимо учитывать, что:

- а) инструмент создается для использования АБД;
- б) инструмент может быть использован для дальнейшей автоматизации процесса управления индексами.

В качестве аналогичного инструмента рассмотрим *Index Advisor* для СУБД *PostgreSQL*. Процесс работы инструмента представлен на рисунке 1.1

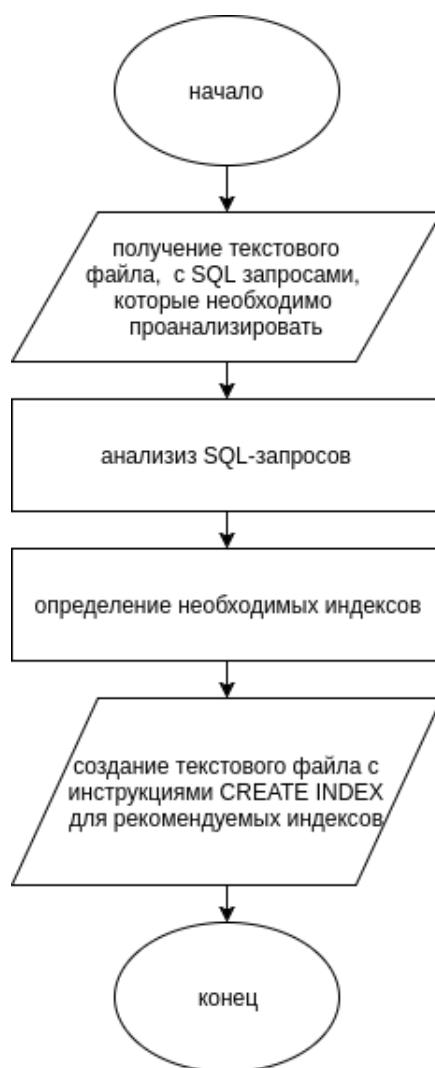


Рисунок 1.1 — Процесс работы Index Advisor для PostgreSQL

Требования к разрабатываемому инструменту

- а) Считывание текстового файла с набором SQL запросов. Каждый запрос записывается в одной строчку. Несколько запросов разделяется символом ";" и переводом строки;

б) В зависимости от считанного файла в случае успеха разбора всех запросов программа должна возвращать 0, а в случае ошибки хотя бы в одном запросе - ненулевое значение, которое, как правило, интерпретируется как код ошибки. Если встречается ошибка, то прекратить дальнейший разбор файла;

в) Для успешно считанных индексов создать файл-скрипт с инструкциями CREATE INDEX для создания индексов для этих запросов.

Тестирование разрабатываемого инструмента

Правильность работы инструмента необходимо проверить на экспериментальной БД с помощью:

а) команды *EXPLAIN*, доказывающим, что СУБД использует построенный индекс при выполнении данного запроса;

б) команды *SHOW PROFILER*, доказывающим уменьшение времени выполнения запросов после добавления индекса.

2 Конструкторский раздел

В данном разделе изучаются индексы, типы индексов, индексы в СУБД MySQL. На основе этой информации описывается алгоритм построения индексов для сложных запросов. Данная задача рассматривается по объектно-ориентированной методологии.

2.1 Индексы

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск.

Производительность

Для оптимальной производительности запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах. Для одной таблицы может быть создано несколько индексов. Однако увеличение числа индексов замедляет операции добавления, обновления, удаления строк таблицы, поскольку при этом приходится обновлять сами индексы. Кроме того, индексы занимают дополнительный объем памяти, поэтому перед созданием индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса. [20]

Существует много типов индексов, каждый из которых лучше всего подходит для достижения той или иной цели. Индексы реализуются на уровне подсистем хранения, а не на уровне сервера. Таким образом, они не стандартизованы: в каждой подсистеме индексы работают немного по-разному, и далеко не все подсистемы допускают использование существующего разнообразия индексов. Даже если некоторый тип поддерживается в нескольких подсистемах хранения, внутренняя реализация может различаться. [3]

Поэтому для начала рассмотрим какие подсистемы хранения реализованы в MySQL, а затем некоторые типы индексов.

2.2 Подсистемы хранения MySQL

При разработке приложения для MySQL необходимо решить, какую подсистему хранения использовать. Поскольку допустимо выбирать способ хранения дан-

ных для каждой таблицы в отдельности, нужно ясно понимать, как будет использоваться каждая таблица, и какие данные в ней планируется хранить. Это также поможет получить хорошее представление о приложении в целом и о потенциале его роста. Вооружившись этой информацией, можно осознанно выбирать подсистемы хранения данных.

2.2.1 Подсистема MyISAM

Как одна из самых старых подсистем хранения, включенных в MySQL, MyISAM обладает многими функциями, которые были разработаны за годы использования СУБД для решения различных задач. Она предоставляет полнотекстовое индексирование, сжатие и пространственные функции (для геоинформационных систем – ГИС). MyISAM не поддерживает транзакции и блокировки на уровне строк.

2.2.2 Подсистема InnoDB

Подсистема хранения InnoDB была разработана для транзакционной обработки, в частности для обработки большого количества краткосрочных транзакций, которые значительно чаще благополучно завершаются, чем откатываются. Она остается наиболее популярной транзакционной подсистемой хранения. Высокая производительность и автоматическое восстановление после сбоя делают ее популярной и для нетранзакционных применений.

Согласно [4] эти две рассмотренные подсистемы хранения являются самыми распространенными: на MyISAM — 74,4%, на InnoDB — 68,5%, на другом движке — 31%. В связи с тем, что один и тот же тип индекса в разных подсистемах хранения может быть реализован по-разному, они будут иметь свои преимущества и недостатки. [3, р. 137]

2.3 Типы индексов

2.3.1 B-tree-индексы

Когда говорят об индексе без упоминания типа, обычно имеют в виду B-Tree индексы, в которых для хранения данных используется структура, называемая B-tree. Во многих подсистемах хранения на самом деле используются индексы типа B+Tree, в которых каждый листовый узел содержит указатель на следующий для ускорения обхода дерева по диапазону значений.

Общая идея B-дерева заключается в том, что значения хранятся по порядку, и все листовые страницы находятся на одинаковом расстоянии от корня. На рисунке 2.1 показано абстрактное представление B-Tree индекса. B-Tree-индекс ускоряет доступ к данным, поскольку подсистеме хранения не нужно сканировать всю таб-

лицу для поиска нужной информации. Вместо этого она начинает с корневого узла (не показанного на этом рисунке). В корневом узле имеется массив указателей на дочерние узлы, и подсистема хранения переходит по этим указателям. Чтобы найти подходящий указатель, она просматривает значения в узловых страницах, которые определяют верхнюю и нижнюю границы значений в дочерних узлах. В конечном итоге подсистема хранения либо определяет, что искомое значение не существует, либо благополучно достигает листовой страницы.

Поскольку в В-Tree индексах индексированные столбцы хранятся в упорядоченном виде, то они полезны для поиска по диапазону данных. Например, при спуске вниз по дереву индекса, построенного по текстовому полю, значения перебираются в алфавитном порядке, поэтому поиск «всех лиц, чьи фамилии начинаются с буквы от В до Д» оказывается эффективным.

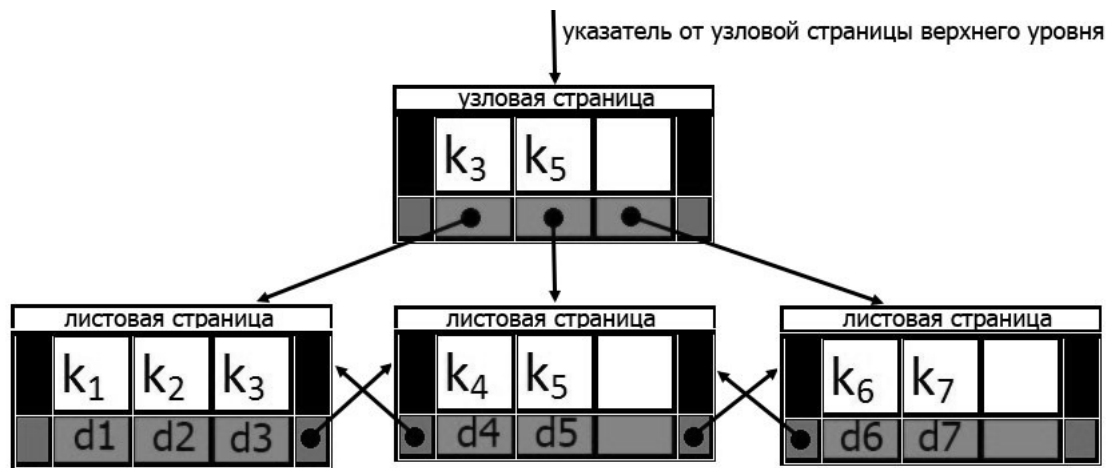


Рисунок 2.1 — Структура данных B+tree.

Где k_i - ключи ($k_i < k_{i+1}$), d_i - данные.

Сложность поиска в линейной структуре данных $O(N)$, а в B-tree $O(\log(k) + N_k)$, где k — количество уровней, а N_k — количество элементов в узле. Поэтому индексы, использующие структуру данных B-tree, эффективны для поиска данных.

Подсистемы хранения представляют B-Tree-индексы на диске по-разному, и это может влиять на производительность. Например, в MyISAM используется техника сжатия префикса, позволяющая уменьшить размер индекса, а InnoDB не сжимает индексы, поскольку это лишило бы ее возможности выполнять некоторые оптимизации. Кроме того, индексы MyISAM ссылаются на индексированные строки по их физическому адресу на диске, а InnoDB — по значениям первичного ключа. Каждый вариант имеет свои достоинства и недостатки.

2.3.2 Хеш-индексы

Хеш-индекс строится на основе хеш-таблицы и полезен только для точного поиска с указанием всех столбцов индекса. Для каждой строки подсистема хранения вычисляет хеш-код индексированных столбцов – сравнительно короткое значение, которое, скорее всего, будет различно для строк с разными значениями ключей. В индексе хранятся хеш-коды и указатели на соответствующие строки. Т.е. хеш-индексы предполагают хранение не самих значений, а их хэшей, благодаря чему уменьшается размер (а, соответственно, и увеличивается скорость их обработки) индексов из больших полей. Таким образом, при запросах с использованием хеш-индексов, сравниваться будут не искомое со значения поля, а хэш от искомого значения с хэшами полей.

Из-за нелинейности хэш-функций данный индекс нельзя сортировать по значению, что приводит к невозможности использования в сравнениях больше/меньше и «is null». Кроме того, так как хэши не уникальны, то для совпадающих хэшей применяются методы разрешения коллизий.

Подсистема хранения InnoDB поддерживает так называемые адаптивные хеш-индексы. Когда InnoDB замечает, что доступ к некоторым значениям индекса происходит очень часто, она строит для них хеш-индекс в памяти, помимо уже имеющихся B-Tree-индексов. Тем самым к B-Tree-индексам добавляются некоторые свойства хеш-индексов, например очень быстрый поиск. Этот процесс полностью автоматический, и вы не можете ни контролировать, ни настраивать его.

2.3.3 Полнотекстовые индексы

В большинстве типичных запросов присутствует фраза WHERE, в которой значения сравниваются на равенство, выделяются диапазоны строк и т. д. Но иногда нужно искать по ключевому слову, и в этом случае поиск должен быть основан на релевантности, а не простом сравнении строк. Для этой цели и предназначены системы полнотекстового поиска. Для полнотекстового поиска требуется специальный синтаксис запроса. Индекс необязателен, но при его наличии поиск выполняется быстрее. Полнотекстовые индексы имеют специальную структуру, ускоряющую поиск документов, содержащих заданные ключевые слова.

Полнотекстовый (FULLTEXT) индекс позволяет искать в тексте ключевые слова, а не сравнивать искомое значение со значениями в столбце. Полнотекстовый поиск не имеет ничего общего с другими типами поиска. С ним связано много тонкостей, например стоп-слова, стемминг, учет множественного числа, а также булевский поиск. Он гораздо больше напоминает поисковые системы, нежели обычное сравнение с критерием во фразе WHERE.

В MySQL поддержка полнотекстового поиска реализована только в подсистеме MyISAM.

2.4 B-tree индексы в MySQL InnoDB

В данной работе рассматриваются только B-tree-индексы для подсистемы хранения InnoDB.

Рассмотрим более подробно B-tree индексы в MySQL InnoDB. В качестве примера возьмем таблицу [2.1](#), заголовок которой *poet(poet_id, last_name, first_name, dob, country)*

Таблица 2.1 — Таблица поэтов

poet_id	last_name	first_name	dob	country
1	"Блок"	"Александр"	1880	"ru"
2	"Фет"	"Афанасий"	1820	"ru"
3	"Лермонтов"	"Михаил"	1814	"ru"
4	"Ильф"	"Илья"	1897	"ru"
5	"Пушкин"	"Александр"	1799	"ru"
6	"Булгаков"	"Михаил"	1891	"ru"
7	"Есенин"	"Сергей"	1895	"ru"

2.4.1 Кластерный индекс

В InnoDB данные хранятся в структуре данных B+tree, где в узловых страницах хранятся первичные ключи, а в листовых страницах хранятся данные. Такое дерево называется кластерным индексом. Над таблицей можно построить только один кластерный индекс, поскольку невозможно хранить одну и ту же запись одновременно в двух местах. Однако часть или всю запись можно хранить в нескольких местах, что будет использоваться в покрывающих индексах [2.4.6](#).

Для рисунка [2.1](#) и таблицы [2.1](#): $k_1 \dots k_7$ - первичные ключи (*poet_id*), $k_i = i$, $d_1 \dots d_7$ - данные, т.е. d_1 = Блок, Александр, 1880, ru; d_2 = Фет, Афанасий, 1820, ru; ...

2.4.2 Вторичный индекс

Для оптимизации конкретных запросов используются *вторичные индексы* (далее просто индексы). В узловых страницах индексов хранятся поля, по которым создан этот индекс, а в листовых страницах хранится значение первичного ключа. Для каждой таблицы в одном запросе используется только один индекс. При использовании в запросах индекса, сначала будет найдено значение первичного ключа,

затем по этому значению будут найдены данные в кластерном индексе. Поэтому при создании вторичного индекса, в конец неявно добавляется первичный ключ.

На рисунке 2.2 показан индекс по полю (*dob*): $k_1 \dots k_7$ - ключи, по которым построен индекс; $k_1 = 1799$, $k_2 = 1814$, $k_3 = 1820$, ...; $p_1 \dots p_7$ - первичные ключи (*poet_id*); $p_i = i$.

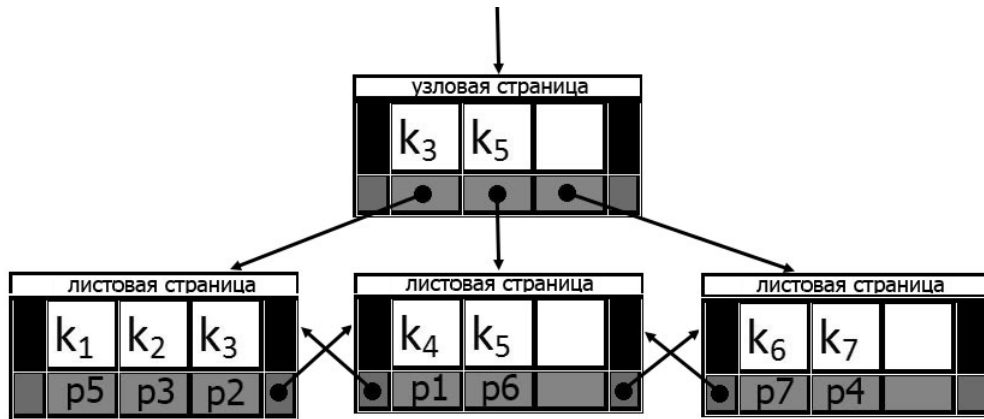


Рисунок 2.2 — Структура вторичного индекса

2.4.3 Составной индекс

Составной индекс - это индекс, построенный по нескольким полям. Чтобы правильно использовать составные индексы, необходимо понять структуру их хранения. Все работает точно так же, как и для обычного индекса, но для значений используются значения всех входящих колонок сразу, по которым строится индекс.

Если построить индекс по полям (*dob*, *last_name*), то для рисунка 2.2 $k_1 = 1799$ Пушкин, $k_2 = 1814$ Лермонтов, $k_3 = 1820$ Фет, ...

2.4.4 Частичный индекс

Составной индекс может использоваться не полностью, но только как левый префикс. Такое использование индекса называется *частичным индексом*.

Оператор *LIKE* с использованием знака % в конце указываемого шаблона рассматривается как левый префикс.

2.4.5 Селективность индексов

Чем меньше строк войдет в выборку, тем быстрее будет работать поиск по ней. Индекс, дающий наименьшую выборку, называется *более селективным*. Если СУБД может применить несколько индексов к данному SQL запросу, то использоваться будет более селективный.

2.4.6 Покрывающий индекс

Индекс называется *покрывающим*, если в нем есть все поля, используемые в запросе. Для того чтобы вернуть результат запроса при использовании покрывающего индекса СУБД не нужно обращаться к кластерному индексу. Покрывающие индексы позволяют имитировать кластерные индексы.

Для запроса на листинге 2.1 можно построить индекс $(last_name, dob)$, который будет считаться покрывающим.

Листинг 2.1 — запрос для covering-index

```
1 SELECT last_name , dob
2 FROM poet
3 WHERE last_name = "Пушкин"
```

2.5 Индексы для сложных запросов

Общие правила

- а) в одном запросе для каждой таблицы используется максимум один индекс;
- б) в выражениях *GROUP BY* и *ORDER BY* поля только из одной таблицы.

Под *сложным запросом* будем понимать запросы, вида листинг 2.2.

Листинг 2.2 — Вид сложного запроса

```
1 SELECT * FROM
2 t1 {INNER | LEFT | RIGHT} JOIN t2 ON conditional_definition
3   [WHERE where_definition]
4   [ORDER BY col_name [ASC | DESC]]
5
6 where_definition:
7   where_expression or
8   where_expression [AND] where_expression
9
10 where_expression:
11   column_name [> | >= | = | < > | <= | < ] constant or
12   column_name LIKE constant or
13   where_definition
14
15 conditional_definition:
16   conditional_expression or
17   conditional_expression [AND] conditional_expression
18
19 conditional_expression:
20   column_name = column_name or
21   conditional_expression
```

2.6 Определение индексов для сложных запросов

Последовательность действий, для определения индексов, применимых для *сложного запроса* представлена на рисунке 2.3.

Опишем некоторые действия более подробно.

2.6.1 Определение таблицы для полного сканирования

При соединении двух таблиц (А и В), для каждой строки одной таблицы А будет происходить сканирование другой таблицы В. Такое сканирование таблицы А назовем *полным сканированием*. Формализуем действия MySQL оптимизатора [3, р. 217] по соединению таблиц в виде *fullscan алгоритма*.

Fullscan алгоритм - алгоритм для определения таблицы, по которой будет осуществлено полное сканирование. Алгоритм на вход принимает запрос и возвращает название таблицы, по которой будет полное сканирование или *NULL*, если таблица на данном этапе не может быть определена. Псевдокод представлена в алгоритме 1. Схема алгоритма представлена на рисунке 2.4.

Рассмотрим работу *fullscan алгоритма* по обработке заданного SQL запроса на конкретных примерах.

Пример №1

query = FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b > 1 AND t2.c = 5

Условие соединения - *LEFT JOIN*. В условии *WHERE* исключается возможность равенства поля *t2.c* значению *NULL*, значит изменим запрос на

query = FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b > 1 AND t2.c = 5

Условие соединения - *INNER JOIN*. Сортировок нет, значит вернуть *NULL*.

Ответ: по какой таблице будет осуществлено полное сканирование на данном этапе не определено.

Пример №2

query = FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5000 AND t1.c > 3 ORDER BY t2.c, t2.d

Условие соединения - *LEFT JOIN*. В условии *WHERE* не исключается возможность равенства любого поля таблицы *t2* значению *NULL*, значит вернуть *t1*.

Ответ: по таблице *t1* будет осуществлено полное сканирование.

Пример №3

query = FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5000 AND t2.c > 3 ORDER BY t2.c, t2.d

Условие соединения - *LEFT JOIN*. В условии *WHERE* исключается возможность равенства поля *t2.b* и *t2.c* значению *NULL*, значит изменим запрос на

query = FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t2.b = 5000 AND t2.c > 3

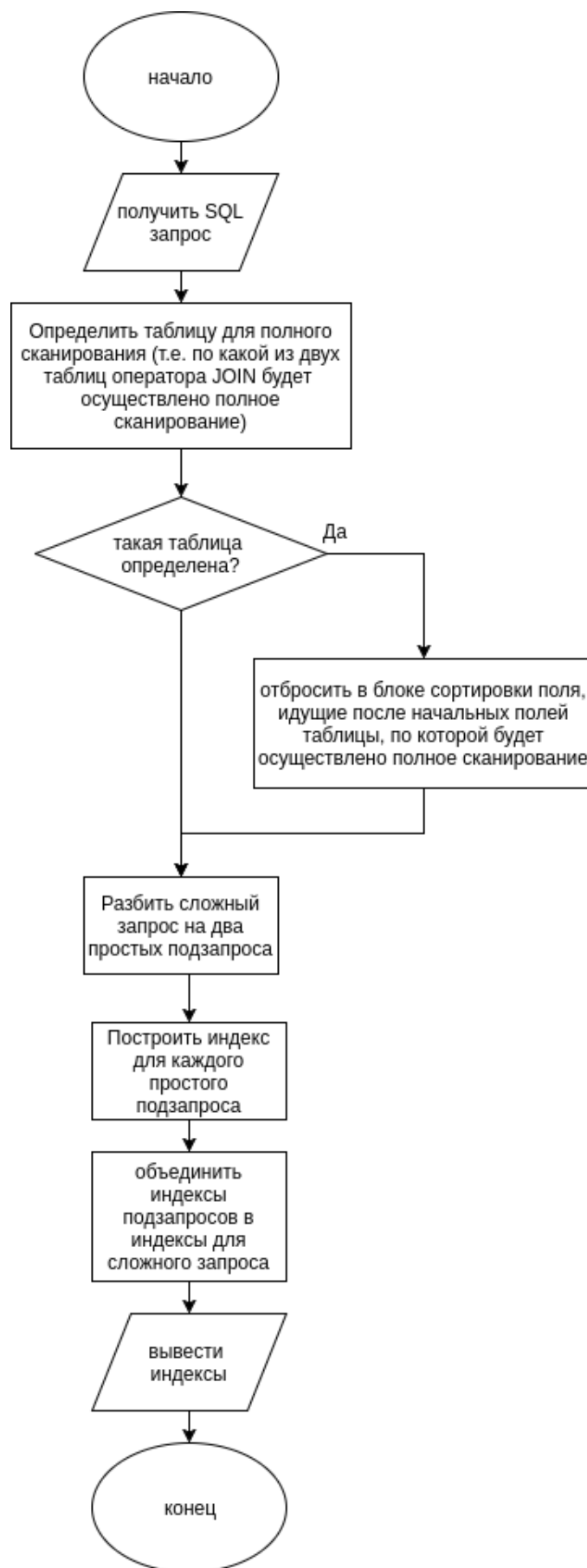


Рисунок 2.3 — Схема определения индексов

Algorithm 1 Fullscan алгоритм

```
1: function FULLSCANALG(query)
2:   if условие соединения  $A$  LEFT JOIN  $B$  then
3:     if в условии WHERE исключается возможность равенства любого из полей
        таблицы  $B$  значению NULL then
4:       заменить LEFT JOIN на INNER JOIN
5:     else
6:       return таблица  $A$ 
7:     end if
8:   end if

9:   if условие соединения  $A$  RIGHT JOIN  $B$  then
10:    if в условии WHERE исключается возможность равенства любого из полей
        таблицы  $A$  значению NULL then
11:      заменить RIGHT JOIN на INNER JOIN
12:    else
13:      return таблица  $B$ 
14:    end if
15:  end if

16:  if условие соединения  $A$  INNER JOIN  $B$  then
17:    if есть сортировка ORDER BY  $A.a, \dots$  then
18:      return таблица  $A$ 
19:    else if есть сортировка ORDER BY  $B.a, \dots$  then
20:      return таблица  $B$ 
21:    end if
22:  end if

23:  return NULL
24: end function
```

ORDER BY $t2.c, t2.d$

Условие соединения - *INNER JOIN*. Есть сортировка *ORDER BY* $t2.c, \dots$. Вернуть $t2$.

Ответ: по таблице $t2$ будет осуществлено полное сканирование.

2.6.2 Отбрасывание лишних полей в блоке сортировки

Необходимо отбросить в блоке сортировки поля, идущие после начальных полей таблицы, по которой будет осуществлено полное сканирование.

Примеры

Пусть

$T = t2$,

$query = \dots ORDER BY t2.z, t2.y, t1.z, t1.y, t2.x \dots$,

тогда

$query1 = \dots ORDER BY t2.z, t2.y$

Пусть

$T = t2$,

$query = \dots ORDER BY t1.x, t2.x \dots$,

тогда отбрасываются все поля (т.е. в $query1$ не будет сортировки)

2.6.3 Разбиение сложного запроса

Сложный запрос необходимо разбить на два простых подзапроса, при этом в условии соединения таблиц отбрасываемую таблицу заменить на *CONST* и переместить в блок *WHERE*.

Примеры

Пусть

$query = \dots ON t1.a = t2.a WHERE t1.b = Z, t2.b = Y \dots$,

тогда

$query1 = \dots WHERE t1.a = CONST AND t1.b = Z \dots$,

$query2 = \dots WHERE t2.a = CONST AND t2.b = Y \dots$

2.6.4 Построение индекса для простого запроса

Для того, чтобы автоматизировать процесс построения индексов, рассмотренных в разделе 2.4, необходимо исследовать, в каких случаях они могут применяться. Рассмотрим примеры использования индексов для *простых запросов*.

Под *простым запросом* будем понимать запросы, вида листинг 2.3.

Листинг 2.3 — Вид простого запроса

```
1 SELECT * FROM
2 t2 [WHERE where_definition]
3 [ORDER BY col_name [ASC | DESC]]
4
5 where_definition:
6     where_expression or
7     where_expression [AND] where_expression
8
9 where_expression:
10     column_name [> | >= | = | <> | <= | < ] constant or
11     column_name LIKE constant or
12     where_definition
```

Индексы для WHERE

Для запроса на листинге 2.4 можно построить индексы $(dob, last_name)$ и $(last_name, dob)$.

Листинг 2.4 — запрос для index-on-where

```
1 SELECT *
2 FROM poet
3 WHERE last_name = "Пушкин"
4     AND dob = 1799
```

Рассмотрим работу индексов на примере индекса $INDEX \equiv (a, b, c)$, где a, b - числа, c - строка.

а) $a = 5 \text{ AND } b = 10 \text{ AND } c = \text{"Hello world"}$

$INDEX$ применяется полностью

б) $b = 5$

$INDEX$ не применяется

в) $a = 5$

$INDEX$ применяется как левый префикс по первому полю

г) $a = 5 \text{ AND } b = 10$

$INDEX$ применяется как левый префикс по первым двум полям

д) $a = 5 \text{ AND } b = 10 \text{ AND } c \text{ LIKE } \text{"Hello w\%"} \text{"}$

$INDEX$ применяется как левый префикс по первым трем полям

е) $a > 5$

$INDEX$ применяется как левый префикс по первому полю

ж) $a = 5 \text{ AND } b \text{ IN } (2, 3)$

условие IN - рассматривается как поиск по диапазону, $INDEX$ применяется как левый префикс по первым двум полям

з) $a=5 \text{ AND } b=10 \text{ AND } c \text{ LIKE } \text{"\% world"}$

$INDEX$ применяется как левый префикс по первым двум полям

и) $a > 5 \text{ AND } b = 2$

$INDEX$ применяется как левый префикс по первому полю

к) $a=5 \text{ AND } c=10$

$INDEX$ применяется как левый префикс по первым двум полям

Индексы при сортировке

В индексе данные хранятся в отсортированном виде, поэтому дополнительно сортировать данные выборки не требуется.

Для запроса на листинге 2.5 строим индекс (dob) .

Листинг 2.5 — запрос для index-order

```
1 SELECT *  
2 FROM poet  
3 ORDER BY dob
```

Для запроса на листинге 2.6 строим индекс (*country*, *dob*). В индексе находим строки, удовлетворяющие условию *country="ru"*, а в этой выборке строки уже отсортированы по *dob*.

Листинг 2.6 — запрос для index-order

```
1 SELECT *  
2 FROM poet  
3 WHERE country="ru"  
4 ORDER BY dob
```

Для запроса на листинге 2.7 строим индекс (*dob*). *GROUP BY* возьмет уже отсортированные строки из индекса и уберет повторы, а т.к. строки уже отсортированные - *ORDER BY* всего лишь задаст, в каком порядке выводить данные.

Листинг 2.7 — запрос для index-order

```
1 SELECT *  
2 FROM poet  
3 GROUP BY dob  
4 ORDER BY dob
```

Примеры запросов, к которым применяется индекс (*a*, *b*):

- а) сортировка по первой колонке
- б) первая колонка в условии *WHERE*, и сортировка по второй колонке
- в) первая колонка в условии *WHERE* и сортировка по первой колонке
- г) сортировка по двум колонкам и обе в одну сторону

Примеры запросов, к которым не применяется индекс (*a*, *b*):

а) сортировка по второй колонке, при этом в условии *WHERE* первая колонка не проверяется на строгое равенство. Например, для запроса *WHERE a>5 ORDER BY b* в индексе будут данные (*a*, *b*): *a=6, b=2; a=6, b=3; a=7, b=0; a=8, b=1*. MySQL сделает выборку по условию *a>5*, но в этой выборке строки не отсортированы по *b*.

- б) *WHERE a IN (1,2) ORDER BY b* (аналогично предыдущему запросу)
- в) сортировка разных столбцов в разных направлениях

Чтобы обойти это, можно сделать виртуальную колонку, например, перед числом поставить минус.

Агрегирующие функции MIN, MAX

Так как данные в индексе отсортированы, то для нахождения минимального или максимального значения достаточно взять крайнее значение.

Для запроса на листинге 2.8 строим индекс (*dob*).

Листинг 2.8 — запрос для index-aggr

```
1 SELECT MAX(dob)
2 FROM poet ;
```

Для запроса на листинге 2.9 строим индекс (*first_name, dob*).

Листинг 2.9 — запрос для index-aggr

```
1 SELECT MAX(dob)
2 FROM poet
3 GROUP BY first_name
```

2.6.5 Объединение индексов подзапросов

При объединении индексов для двух простых подзапросов необходимо из индекса для таблицы, по которому будет полное сканирование, удалить поле, по которому происходит соединение таблиц. Если таблица, по которой происходит полное сканирование не определена, то вернуть две пары индексов, каждая из которых может быть использована для оптимизации выполнения запроса.

Пусть

$$T = t1,$$

$$index(t1) = t1(a, b),$$

$$index(t2) = t2(c, a),$$

$$query0 = \dots ON t1.a = t2.a \dots,$$

тогда индексы для сложного запроса:

$$index(t1) = t1(b), index(t2) = t2(c, a).$$

Пусть

$$T = NULL,$$

$$index(t1) = t1(a, b, c),$$

$$index(t2) = t2(a, b, c),$$

$$query1 = \dots ON t1.a = t2.a \dots,$$

тогда индексы для сложного запроса:

$$index1(t1) = t1(a, b, c), index2(t2) = t2(b, c),$$

$$index2(t1) = t1(b, c), index1(t2) = t2(a, b, c).$$

2.7 Объектно-ориентированный анализ

В результате анализа можно выделить следующие сущности:

а) поле

- б) аргумент
- в) условное выражение
- г) выражение блока ON
- д) выражение блока ORDER BY
- е) простой запрос
- ж) сложный запрос
- з) индекс

Рассмотрим каждую сущность более подробно.

2.7.1 Поле

Класс *Поле* представляет одну колонку таблицы.

Атрибуты:

- а) *table_number* - идентификатор таблицы
- б) *column_number* - идентификатор колонки

2.7.2 Аргумент

Класс *Аргумент* представляет аргумент, который используется в выражениях сравнения.

Атрибуты:

- а) *type* - тип аргумента {value | const | null}
- б) *value* - значение аргумента

2.7.3 Условное выражение

Класс *Условное выражение* представляет условное выражение, с указанием оператора сравнения и аргументов, с которыми сравнивается указанное поле.

Атрибуты:

- а) *field* - ссылка на объект класса *поле*,
- б) *operator* - оператор {g | ge | e | ne | le | l | like},
- в) *args* - массив объектов класса *аргумент*,

2.7.4 Выражение блока ON

Класс *Выражение блока ON* представляет выражение блока ON, где указываются два поля с оператором сравнения.

Атрибуты:

- а) *fields* - массив из двух ссылок на объекты класса *поле*,
- б) *operator* - оператор {> | >= | = | <> | <= | < | like},

2.7.5 Выражение блока ORDER BY

Класс *Выражение блока ORDER BY* представляет выражение блока ORDER BY, где указываются поле сортировки и направление сортировки.

Атрибуты:

- а) *field* - ссылка на объект класса *поле*,
- б) *asc* - направление сортировки, если *TRUE* - то по возрастанию, иначе - по убыванию.

2.7.6 Индекс

Класс *Индекс* представляет набор полей, по которым следует построить индекс.

Атрибуты:

- а) *table_name* - имя таблицы,
- б) *columns_name* - имена колонок таблицы,
- в) *where_eq_fields* - массив ссылок на поля, по которым происходит строгое равенство,
- г) *where_not_eq_fields* - массив ссылок на поля, по которым происходит нестрогое равенство,
- д) *order_by_fields* - массив ссылок на поля, по которым происходит сортировка

Методы:

- а) *delete_fields([fields])* - удалить из всех атрибутов объекта *индекс* поля, указанные в передаваемом массиве,
- б) *sql()*: строка - получить SQL строку создания индекса, при этом поля будут из *where_eq_fields*, *order_by_fields*, *where_not_eq_fields* в порядке перечисления и с учетом правил из раздела [2.6.4](#)

2.7.7 Простой запрос

Класс *Простой запрос* представляет собой распарсенный запрос в структуру программы.

Атрибуты:

- а) *sql_query* - строка с SQL запросом,
- б) *table_name* - имя таблицы из запроса,
- в) *columns_name* - имена колонок таблицы из запроса,
- г) *where* - массив объектов класса *условное выражение*,
- д) *order_by* - массив объектов класса *выражение блока ORDER BY*.

Методы:

а) *init_from_sql(query)* - инициировать объект из SQL строки, т.е. по заданной SQL строке заполнить атрибуты:

- 1) *table_name*
- 2) *columns_name*
- 3) *where*
- 4) *order_by*

б) *get_index(): index* - создать и вернуть объект класса *индекс* с помощью правил из раздела [2.6.4](#)

2.7.8 Сложный запрос

Класс *Сложный запрос* представляет собой распарсенный запрос в структуру программы.

Атрибуты:

- а) *sql_query* - строка с SQL запросом,
- б) *tables_name* - массив из двух элементов-имён таблиц из запроса,
- в) *columns_name* - массив из двух элементов-массивов имён колонок таблиц из запроса,
- г) *join_type* - тип соединения таблиц {inner | left | right}
- д) *on* - массив объектов класса *выражение блока ON*,
- е) *where* - массив объектов класса *условное выражение*,
- ж) *order_by* - массив объектов класса *выражение блока ORDER BY*.

Методы:

- а) *init_from_sql(query)* - инициировать объект из SQL строки,
- б) *get_optimization_join(): join_type* - получить тип join, который будет использован оптимизатором MySQL (см. раздел [2.6.1](#)),
- в) *get_fullscan_table(): table_number* - определить по какой таблице будет осуществлено полное сканирование,
- г) *get_simple_queries(): simple_query_1, simple_query_2* - разбить запрос на два простых подзапроса,
- д) *get_indexes(): [indexes]* - вернуть индексы этого запроса.

2.7.9 Диаграмма классов

Выделенные сущности можно представить на диаграмме классов на рисунке [2.5](#)

2.8 Тестирование разрабатываемого приложения

Основываясь на виде запроса (листинг 2.2), который будет обрабатывать разрабатываемый инструмент, выделим параметры, которые могут изменяться и значения, которые они могут принимать:

- а) **join**: {left, right, inner},
- б) **where**: {-, $t1.a = const$, $t2.a = const$, $t1.a <> const$, $t2.a <> const$ }
- в) **order by**: {-, t1.a, t2.a}

Учитывая количество параметров и их значения, для тестирования разрабатываемого ПО необходимо провести $3 * 5 * 3 = 45$ экспериментов.

По *методу всех пар*, т.к. большинство ошибок проявляются либо при конкретных значениях одного параметра, либо взаимным влиянием значений двух параметров [21], то достаточно провести 15 экспериментов. План экспериментов представлен в таблице 2.2.

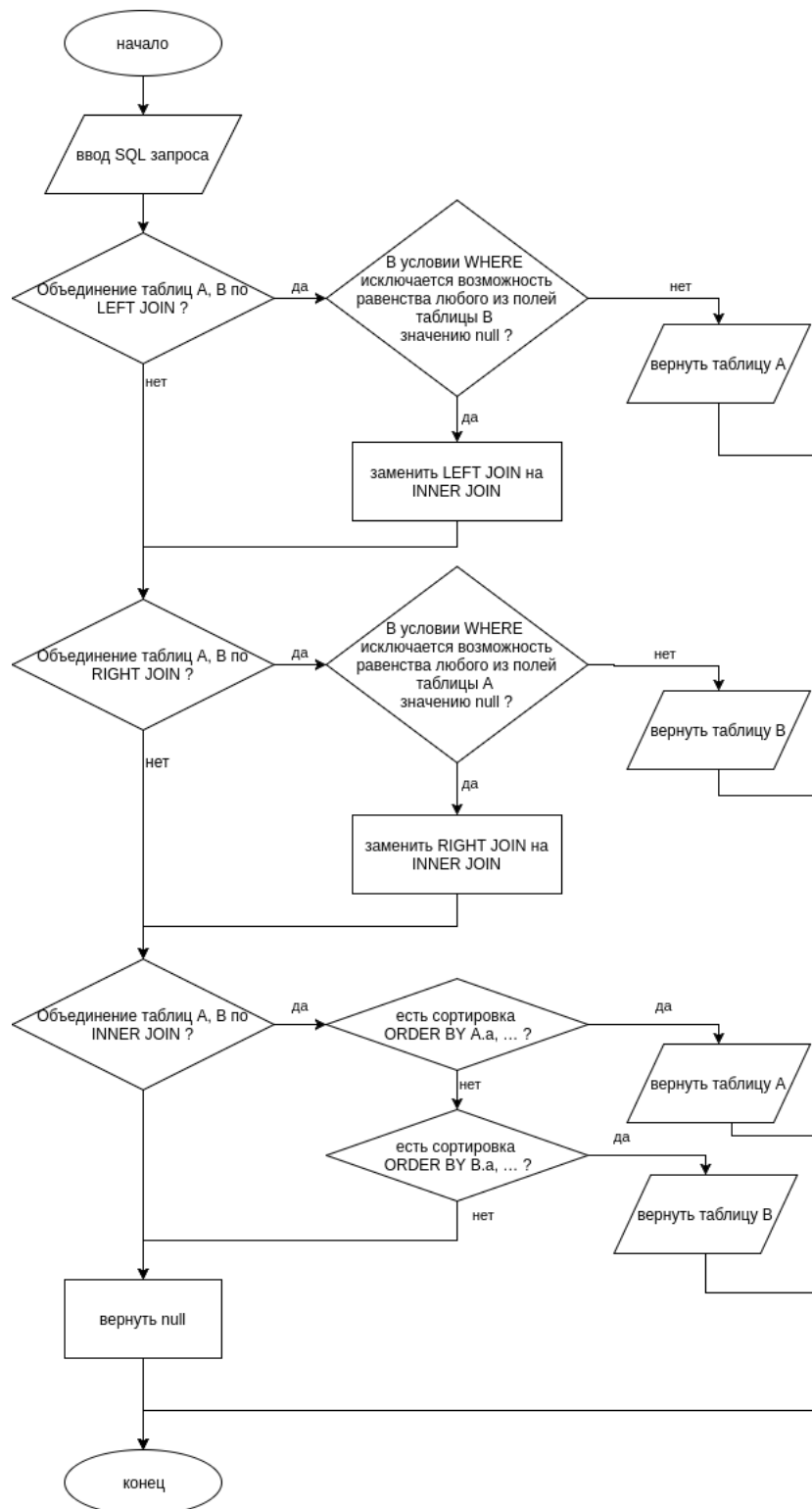


Рисунок 2.4 — Схема fullscan алгоритма

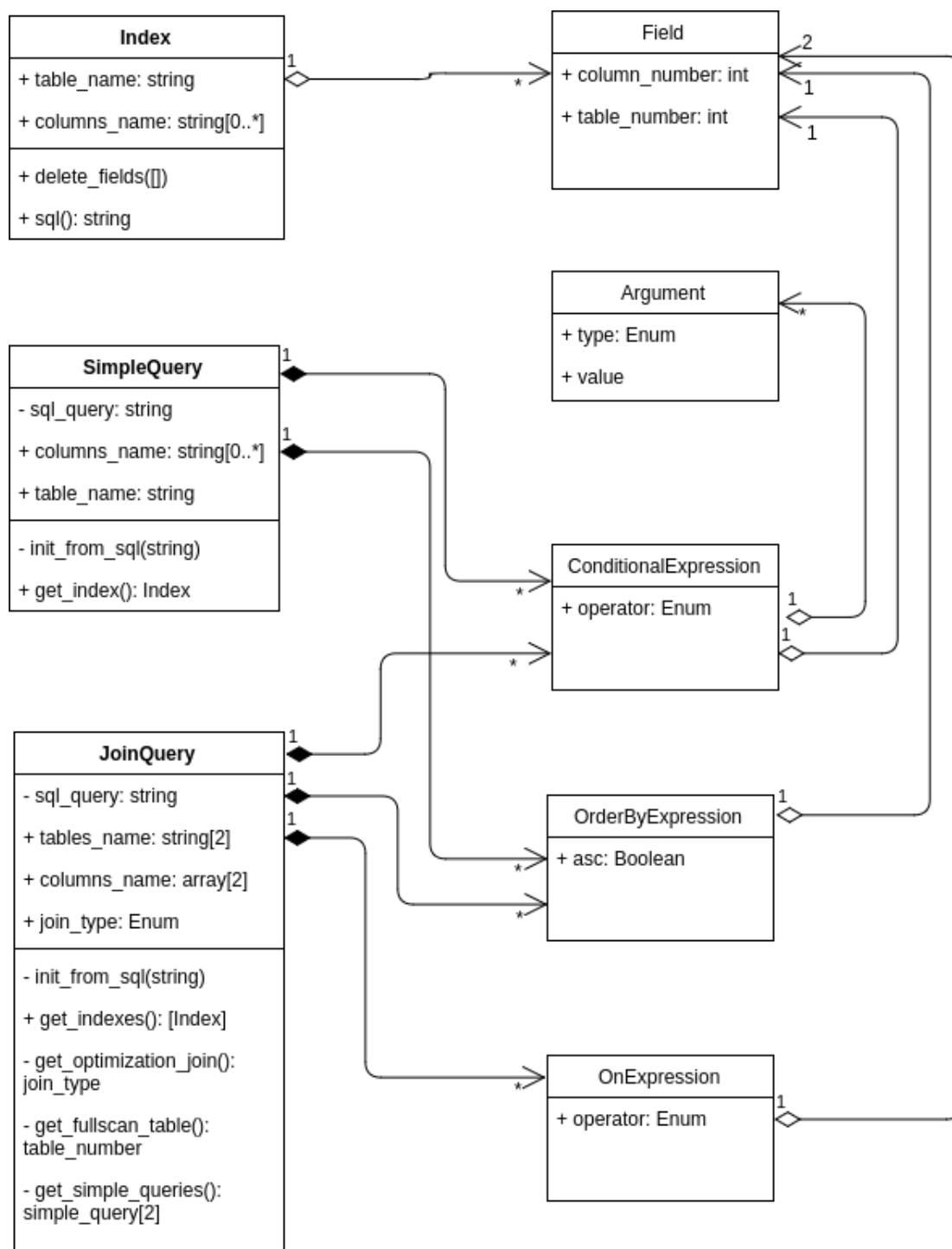


Рисунок 2.5 — Диаграмма классов

Таблица 2.2 — План экспериментов

№	join	order by	where
1	inner	-	—
2	inner	t1.c	$t1.b = const$
3	inner	t2.c	$t2.b = const$
4	inner	-	$t1.b > const$
5	inner	t1.c	$t2.b > const$
6	left	t2.c	—
7	left	-	$t1.b = const$
8	left	t1.c	$t2.b = const$
9	left	t2.c	$t1.b > const$
10	left	-	$t2.b > const$
11	right	t1.c	—
12	right	t2.c	$t1.b = const$
13	right	-	$t2.b = const$
14	right	t1.c	$t1.b > const$
15	right	t2.c	$t2.b > const$

3 Технологический раздел

3.1 Выбор средств программной реализации

В качестве языка программирования был выбран язык *Python3*. Это скриптовый язык с динамической типизацией, что позволяет ускорить процесс разработки, а программу можно запускать на любом компьютере, где есть интерпретатор *Python3*.

3.2 Сборка инструмента

Для того, чтобы запустить разработанный инструмент, необходимо:

- а) установить интерпретатор *Python3*,
- б) установить пакетный менеджер *pip* для *Python3*,
- в) установить модуль *sqlparse*, для парсинга SQL запросов:

```
pip install --upgrade sqlparse
```

- г) скачать скрипты по адресу:

```
https://github.com/iproha94/mysql-indexes/tree/master/src
```

3.3 Руководство пользователя

Процесс использования ПО АБД

а) подготовить входной файл с SQL запросами, для которых необходимо построить индексы. Каждый SQL запрос должен быть на одной строке, и заканчиваться переводом строки. Каждый знак сравнения должен быть обрамлен пробелами.

б) запустить скрипт *index_advisor* с указанием пути входного файла (например 3.1)

```
python3 index_advisor.py example.sql
```

в) результаты работы программы будут в сгенерированном выходном файле (например 3.2). В нем для каждого успешно считанного SQL запроса будут сгенерированы SQL строки с созданием рекомендуемого индекса.

Листинг 3.1 — Пример входного файла

```
1 select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b > 1 AND t2.c = 5;
2 select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5000 AND t1.c >
  3 ORDER BY t2.c , t2.d;
3 select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5000 AND t2.c >
  3 ORDER BY t2.c, t2.d;
4 select * FROM t1 LEFT JOIN t2 ON t2.a = t1.a ORDER BY t2.b"
5 select * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t1.c;
6 select * from t where a = 5 and b > 1;
7 select * from t where a = 5 and c = 5;
8 select * from t where a = 5 and c > 3 and b = 5000;
```



```

9  select * from t where a = 5;
10 select * from t where b = 5 and a = 5 and c > 3 order by c, d;

```

Листинг 3.2 — Пример выходного файла

```

1  select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b > 1 AND t2.c = 5;
2  CREATE INDEX index_t1_ab t1(a, b);
3  CREATE INDEX index_t2_c t2(c);
4
5  CREATE INDEX index_t1_b t1(b);
6  CREATE INDEX index_t2_ac t2(a, c);
7
8  select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5000 AND t1.c >
   3 ORDER BY t2.c , t2.d;
9  CREATE INDEX index_t1_bc t1(b, c);
10 CREATE INDEX index_t2_a t2(a);
11
12 select * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5000 AND t2.c >
   3 ORDER BY t2.c, t2.d;
13 CREATE INDEX index_t1_a t1(a);
14 CREATE INDEX index_t2_bcd t2(b, c, d);
15
16 select * FROM t1 LEFT JOIN t2 ON t2.a = t1.a ORDER BY t2.b;
17 CREATE INDEX index_t2_a t2(a);
18
19 select * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t1.c;
20 CREATE INDEX index_t1_bc t1(b, c);
21 CREATE INDEX index_t2_a t2(a);
22
23 select * from t where a = 5 and b > 1;
24 CREATE INDEX index_t_ab t(a, b);
25
26 select * from t where a = 5 and c = 5;
27 CREATE INDEX index_t_ac t(a, c);
28
29 select * from t where a = 5 and c > 3 and b = 5000;
30 CREATE INDEX index_t_abc t(a, b, c)
31
32 select * from t where a = 5;
33 CREATE INDEX index_t_a t(a);
34
35 select * from t where b = 5 and a = 5 and c > 3 order by c, d;
36 CREATE INDEX index_t_bacd t(b, a, c, d);

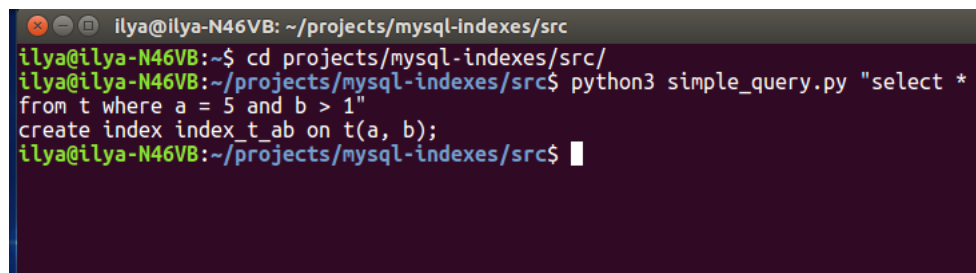
```

Процесс использования ПО другими приложениями

а) запустить скрипт *simple_query* или *join_query*, первым аргументом передав строку с простым или сложным SQL запросом соответственно. Например 3.1, 3.2 :

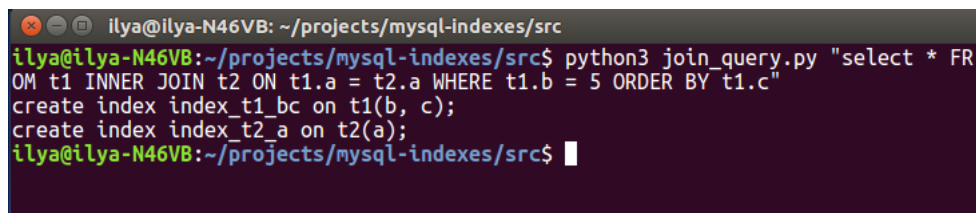
```
python3 simple_query.py "select * from t where a = 5 and b > 1"
python3 join_query.py "select * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t1.c"
```

б) в результате работы в выводе будет строка с SQL запросом для создания индекса. При ошибке в работе программы код возврата будет отличным от нуля.



```
ilya@ilya-N46VB: ~/projects/mysql-indexes/src
ilya@ilya-N46VB:~$ cd projects/mysql-indexes/src/
ilya@ilya-N46VB:~/projects/mysql-indexes/src$ python3 simple_query.py "select *
from t where a = 5 and b > 1"
create index index_t_ab on t(a, b);
ilya@ilya-N46VB:~/projects/mysql-indexes/src$
```

Рисунок 3.1 — Пример запуска скрипта *simple_query.py*



```
ilya@ilya-N46VB: ~/projects/mysql-indexes/src
ilya@ilya-N46VB:~/projects/mysql-indexes/src$ python3 join_query.py "select * FR
OM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t1.c"
create index index_t1_bc on t1(b, c);
create index index_t2_a on t2(a);
ilya@ilya-N46VB:~/projects/mysql-indexes/src$
```

Рисунок 3.2 — Пример запуска скрипта *join_query.py*

4 Экспериментальный раздел

В данном разделе будет осуществлено системное тестирование разработанного ПО по плану экспериментов из таблицы 2.2. Целью экспериментов является исследование использования СУБД MySQL предложенных индексов разработанным ПО. В больше части экспериментов индексы должны быть использованы СУБД, и при этом должен быть выигрыш во времени. Однако в небольшом количестве экспериментов выигрыша во времени может не оказаться, по причинам, указанных в параграфе 2.1.

Эксперименты проводятся на машине со следующими программными и аппаратными характеристиками:

- а) **ОС:** UBUNTU 12
- б) **версия СУБД:** MySQL 5.5
- в) **ОЗУ:** 1Гб.

Скрипты по созданию таблиц БД представлены на листингах 4.1 и 4.2. Создается 2 таблицы с 4 полями, где 3 поля - это случайные числа, четвертое поле - случайная строка. Каждая таблица заполняется на 10000 записей.

Листинг 4.1 — Создание таблиц БД

```
1 create table t1 (a INT(5), b INT(5), c INT(5), d CHAR(50)) engine innodb;  
2 create table t2 (a INT(5), b INT(5), c INT(5), d CHAR(50)) engine innodb;
```

Листинг 4.2 — Наполнение таблиц БД

```
1 import MySQLdb  
2 import random  
3 import string  
4  
5 db = MySQLdb.connect(host="localhost", user="root", passwd="1", db="test",  
6     charset='utf8')  
7 cursor = db.cursor()  
8 for i in range(10000):  
9     a = random.randint(0, 10)  
10    b = random.randint(0, 10)  
11    c = random.randint(0, 10)  
12    d = ''.join(random.SystemRandom().choice(string.ascii_uppercase +  
13        string.digits) for _ in range(30))  
14    sql = """INSERT INTO t1(a, b, c, d) VALUES (%d, %d, %d, '%s')""" % (a,  
15        b, c, d)  
16    cursor.execute(sql)
```

```

17 |      sql = """INSERT INTO t2(a, b, c, d) VALUES (%d, %d, %d, '%s')""" % (a,
18 |      cursor.execute(sql)
19 |
20 |      db.commit()
21 |
22 |      db.close()

```

Для проведения экспериментов будут использованы встроенные средства MySQL:

- а) **EXPLAIN** - для проверки использования индекса,
- б) **SHOW PROFILE** - для сравнения времени выполнения запросов.

4.1 Результаты экспериментов

Эксперимент №1

Запрос: *SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.a*

Результат: *create index index_t1_a on t1(a);*

Используется индекс: да

Время выполнения с индексом: 0.00033800

Время выполнения без индекса: 0.00304400

Индексы ускорили время в 9 раз

Эксперимент №2

Запрос: *SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t1.c*

Результат: *create index index_t1_bc on t1(b, c);, create index index_t2_a on t2(a);*

Используется индекс: да

Время выполнения с индексом: 0.00623250

Время выполнения без индекса: 0.93866375

Индексы ускорили время в 150 раз

Эксперимент №3

Запрос: *SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t2.b = 5 ORDER BY t2.c*

Результат: *create index index_t1_a on t1(a);, create index index_t2_bc on t2(b, c);*

Используется индекс: да

Время выполнения с индексом: 0.00045800

Время выполнения без индекса: 0.94153075

Индексы ускорили время в 2055 раз

Эксперимент №4

Запрос: *SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.a WHERE t1.b > 5*

Результат: *create index index_t1_ab on t1(a, b);*

Используется индекс: да

Время выполнения с индексом: 0.00972600

Время выполнения без индекса: 0.00577575

Индексы ускорили время в 0,5 раза

Эксперимент №5

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b > 5
ORDER BY t1.c*

Результат: *create index index_t1_c on t1(c);, create index index_t2_ab on t2(a, b);*

Используется индекс: -

Время выполнения с индексом: 0.00148550

Время выполнения без индекса: 6.01183250

Индексы ускорили время в 4047 раз

Эксперимент №6

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a ORDER BY t2.c*

Результат: *create index index_t2_a on t2(a);*

Используется индекс: да

Время выполнения с индексом: 12.71380375

Время выполнения без индекса: 33.84384225

Индексы ускорили время в 2,6 раз

Эксперимент №7

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5*

Результат: *create index index_t1_b on t1(b);, create index index_t2_a on t2(a);*

Используется индекс: да

Время выполнения с индексом: 0.00062225

Время выполнения без индекса: 0.00045175

Индексы ускорили время в 0,7 раз

Эксперимент №8

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5*

ORDER BY t1.c

Результат: *create index index_t1_c on t1(c);, create index index_t2_ab on t2(a, b);*

Используется индекс: да

Время выполнения с индексом: 0.00029825

Время выполнения без индекса: 1.06560425

Индексы ускорили время в 3572 раз

Эксперимент №9

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b > 5 ORDER BY t2.c*

Результат: *create index index_t1_b on t1(b);, create index index_t2_a on t2(a)*

Используется индекс: да

Время выполнения с индексом: 6.11766875

Время выполнения без индекса: 15.75222000

Индексы ускорили время в 2,5 раз

Эксперимент №10

Запрос: *SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b > 5*

Результат: *create index index_t1_a on t1(a);, create index index_t2_b on t2(b);*

Используется индекс: да

Время выполнения с индексом: 0.00028800

Время выполнения без индекса: 0.00140550

Индексы ускорили время в 4,8 раз

Эксперимент №11

Запрос: *SELECT * FROM t1 RIGHT JOIN t2 ON t1.a = t2.a ORDER BY t1.c*

Результат: *create index index_t1_a on t1(a);*

Используется индекс: да

Время выполнения с индексом: 12.80716150

Время выполнения без индекса: 33.75347400

Индексы ускорили время в 2,6 раз

Эксперимент №12

Запрос: *SELECT * FROM t1 RIGHT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5 ORDER BY t2.c*

Результат: *create index index_t1_ab on t1(a, b);, create index index_t2_c on t2(c);*

Используется индекс: да

Время выполнения с индексом: 0.00044925

Время выполнения без индекса: 1.00510150

Индексы ускорили время в 2237 раз

Эксперимент №13

Запрос: *SELECT * FROM t1 RIGHT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5*

Результат: *create index index_t1_a on t1(a);, create index index_t2_b on t2(b);*

Используется индекс: -

Время выполнения с индексом: 0.00028650

Время выполнения без индекса: 0.00040375

Индексы ускорили время в 1,4 раз

Эксперимент №14

Запрос: *SELECT * FROM t1 RIGHT JOIN t2 ON t1.a = t2.a WHERE t1.b > 5 ORDER BY t1.c*

Результат: *create index index_t1_cb on t1(c, b);, create index index_t2_a on t2(a);*

Используется индекс: да

Время выполнения с индексом: 0.00106175

Время выполнения без индекса: 5.79531800

Индексы ускорили время в 5458 раз

Эксперимент №15

Запрос: *SELECT * FROM t1 RIGHT JOIN t2 ON t1.a = t2.a WHERE t2.b > 5 ORDER BY t2.c*

Результат: *create index index_t1_a on t1(a);, create index index_t2_cb on t2(c, b);*

Используется индекс: да

Время выполнения с индексом: 0.00102675

Время выполнения без индекса: 16.35171300

Индексы ускорили время в 15925 раз

4.2 Выводы

В каждом из проведенных экспериментов СУБД MySQL использовала индексы, предложенные разработанным ПО.

Если собрать в таблицу 4.1 результаты анализа времени выполнения запросов, то можно увидеть, что построенные индексы ускорили время выполнения запросов в 87% экспериментов, при этом в 47% экспериментов, ускорение времени выполнения запроса было больше, чем на порядок (в среднем в 4777 раз). Однако, в

некоторых экспериментах (13% экспериментов) выигрыша во времени не произошло, что было ожидаемо.

Таблица 4.1 — Результаты экспериментов

Результат	номера экспериментов	кол-во	%
Индексы уменьшили время больше, чем на порядок	2, 3, 5, 8, 12, 14, 15	7	47
Индексы уменьшили время меньше, чем на порядок	1, 6, 9, 10, 11, 13	6	40
Индексы увеличили время меньше, чем на порядок	4, 7	2	13
Индексы увеличили время больше, чем на порядок	-	0	0

Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кожушко В.В., Цыганов Д.Л. Особенности оптимизации запросов в СУБД MYSQL / Кожушко В.В., Цыганов Д.Л. // *Международный научно-исследовательский журнал успехи современной науки и образования*. — 2016.
2. ФОРС Дистрибуция. Семейство продуктов Oracle Database 12c. — http://www.partner.fors.ru/info_products_oracle/basic_technologies/Catalog_Oracle_Database_12C_1.pdf. — 2013. — [Online; accessed 25-05-2017].
3. Шварц Б., Зайцев П., Ткаченко В. MySQL. Оптимизация производительности. / Шварц Б., Зайцев П., Ткаченко В. — Символ-Плюс, 2010.
4. Рейтинги сервисов и технологий. Рейтинг систем управления базами данных (СУБД) 2016. — <http://tagline.ru/database-management-systems-rating/>. — [Online; accessed 25-05-2017].
5. Смитенко А. 10 лучших инструментов для разработки и администрирования MySQL. — <https://habrahabr.ru/post/142385/>. — [Online; accessed 25-05-2017].
6. Сайтостроение от А до Я – как создать сайт бесплатно на конструкторах сайтов! Как настроить и запустить Microsoft SQL Server. — http://www.internet-technologies.ru/articles/article_2270.html. — 2015. — [Online; accessed 25-05-2017].
7. Душан Петкович. Microsoft SQL Server 2012. Руководство для начинающих / Душан Петкович. — БХВ-Петербург, 2013.
8. Паринков О. SQL Azure. — <https://habrahabr.ru/post/66815/>. — 2009. — [Online; accessed 25-05-2017].
9. Microsoft Azure: платформа облачных вычислений и службы. Использование помощника по базам данных SQL на портале Azure. — <https://docs.microsoft.com/ru-ru/azure/sql-database/sql-database-advisor-portal>. — 2016. — [Online; accessed 25-05-2017].
10. *openSUSE wiki* - главный источник информации о проекте *openSUSE*. PostgreSQL. — <https://ru.opensuse.org/Postgresql>. — [Online; accessed 25-05-2017].
11. Википедия. MySQL. — <https://ru.wikipedia.org/wiki/MySQL>. — [Online; accessed 25-05-2017].
12. Документация по MySQL. EXPLAIN. — <http://www.mysql.ru/docs/man/EXPLAIN.html>. — [Online; accessed 25-05-2017].
13. Яремчук С. Инструментарий админа MySql. — <https://xakep.ru/2014/10/08/mysql-admin-toolkit/>. — 2014. — [Online; accessed 25-05-2017].

14. Небольшая шпаргалка по Percona Toolkit. — <http://blog.dh.md/2017/04/percona-toolkit.html>. — 2017. — [Online; accessed 25-05-2017].
15. MSSQL Sql tuning advisor - in mysql. — <https://stackoverflow.com/questions/6208208/mssql-sql-tuning-advisor-in-mysql>. — 2011. — [Online; accessed 25-05-2017].
16. Do any databases support automatic Index Creation? — <https://stackoverflow.com/questions/231528/do-any-databases-support-automatic-index-creation/>. — 2008. — [Online; accessed 25-05-2017].
17. Is there a good tool for MySQL that will help me optimise my queries and index settings? — <https://stackoverflow.com/questions/127630/is-there-a-good-tool-for-mysql-that-will-help-me-optimise-my-queries-and-index-> — 2008. — [Online; accessed 25-05-2017].
18. Is there a product to auto-build indexes based on slow queries? — <https://dba.stackexchange.com/questions/66430/is-there-a-product-to-auto-build-indexes-based-on-slow-queries>. — 2014. — [Online; accessed 25-05-2017].
19. Database Engine Tuning Advisor for mysql. — <https://www.experts-exchange.com/questions/26451435/Database-Engine-Tuning-Advisor-for-mysql.html>. — 2010. — [Online; accessed 25-05-2017].
20. Википедия. Индекс (базы данных). — [https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_\(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85\)](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)). — [Online; accessed 25-05-2017].
21. *David M. Cohen, Siddhartha R. Dala, Gardner C. Patton*. The Combinatorial Design Approach to Automatic Test Generation / David M. Cohen, Siddhartha R. Dala, Gardner C. Patton // *Telcordia*. — 1997.

Приложение А. Примеры получения индексов для сложных запросов

Рассмотрим получение индексов для заданного SQL запроса на конкретных примерах.

Пример №1

```
1 query := t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b > 1 AND t2.c = 5
2
3 T := NULL
4
5 query1 := query
6
7 query21 := ON t1.a = const WHERE t1.b > 1
8 query22 := ON t2.a = const WHERE t2.c = 5
9
10 index(t1) := t1(a, b!)
11 index(t2) := t2(a, c)
12
13 index1(t1) := t1(a, b!)
14 index2(t2) := t2(c)
15
16 index1(t2) := t2(a, c)
17 index2(t1) := t1(b)
```

$(t1(a, b!), t2(c)) \equiv (t1(a, b), t2(c))$

$(t1(b), t2(a, c)) \equiv (t1(b), t2(a, c)), (t1(b), t2(c, a))$

Ответ: $(t1(a, b), t2(c)), (t1(b), t2(a, c)), (t1(b), t2(c, a))$ - множество пар индексов, наиболее подходящих данному запросу.

Пример №2

```
1 query := FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5000 AND t1.c >
   3 ORDER BY t2.c, t2.d
2
3 T := t1
4
5 query1 := FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t1.b = 5000 AND t1.c >
   3
6
7 query21 := ON t1.a = const WHERE t1.b = 5000 AND t1.c > 3
8 query22 := ON t2.a = const
9
10 index(t1) := t1(a, b, c!)
11 index(t2) := t2(a)
12
13 index(t1) := t1(b, c!)
```

$$(t1(b, c!), t2(a)) \equiv (t1(b, c), t2(a))$$

Ответ: $(t1(b, c), t2(a))$ - множество пар индексов, наиболее подходящих данному запросу.

Пример №3

```

1 query := FROM t1 LEFT JOIN t2 ON t1.a = t2.a WHERE t2.b = 5000 AND t2.c >
      3 ORDER BY t2.c, t2.d
2
3 T := t2
4
5 query1 := query
6
7 query21 := ON t1.a = const
8 query22 := ON t2.a = const WHERE t2.b = 5000 AND t2.c > 3 ORDER BY t2.c,
      t2.d
9
10 index(t1) := t1(a)
11 index(t2) := t2(a, b, c!, d!)
12
13 index(t2) := t2(b, c!, d!)

```

$$(t1(a), t2(b, c!, d!)) \equiv (t1(a), t2(b, c, d))$$

Ответ: $(t1(a), t2(b, c, d))$ - множество пар индексов, наиболее подходящих данному запросу.

Пример №4

```

1 query := FROM t1 LEFT JOIN t2 ON t2.a = t1.a ORDER BY t2.b
2
3 T := t1
4
5 query1 := t1 LEFT JOIN t2 ON t2.a = t1.a
6
7 query21 := ON t1.a = const
8 query22 := ON t2.a = const
9
10 index(t1) := t1(a)
11 index(t2) := t2(a)
12
13 index(t1) := t1()

```

$$(t1(), t2(a)) \equiv (t1(), t2(a))$$

Ответ: $(t1(), t2(a))$ - множество пар индексов, наиболее подходящих данному запросу.

Пример №5

```

1 query :=

```

```
2      t1 INNER JOIN t2
3          ON t1.a = t2.a
4 WHERE t1.b = const
5 ORDER BY t1.c
```

Ответ: $(t1(b, c), t2(a))$ - множество пар индексов, наиболее подходящих данному запросу.