ВЫСОКОПРОИЗВОДИТЕЛЬНЫЙ SQL-ДВИЖОК НА ОСНОВЕ СЕМАНТИЧЕСКОГО АНАЛИЗА

Ходиев Шухрат Ильхамович[1], Буриев Искандар И.[2], Джуманиёзов Донёрбек Б.[2],

[1] Доцент Национального Университета Узбекистана имени М. Улугбека, Ташкент, к.т.н.
[2] Национальный Университет Узбекистана имени М. Улугбека, Ташкент, студент
[1,2,3] E-mail: ilch17333@gmail.com
DOI: 10.61587/mmit.tiue.uz.v1i1.249

Аннотация. Рост сложности современных архитектур СУБД и объемов обрабатываемых данных превращает классические методы выполнения запросов в «узкое место» информационных систем. В статье рассматриваются проектирование и программная реализация высокопроизводительного реляционного SQL-движка, использующего методы семантического анализа и оптимизации на основе правил (RBO). Описана строгая конвейерная архитектура системы, включающая AST-парсер, связыватель (Binder) и оптимизатор дерева запросов с применением соответственных алгоритмов. В работе обоснован отказ от классической итераторной модели исполнения (Volcano) в пользу пакетно-векторизованного колоночного подхода, реализованного с помощью инструментов библиотеки NumPy. Внедрение модулей телеметрии и визуализации через Graphviz позволяет использовать разработанный движок как прозрачную платформу для дальнейших научных исследований в области системного программного обеспечения. Разработка собственного SQL-движка позволяет изнутри изучить механику превращения высокоуровневых запросов в быстрые низкоуровневые операции

Ключевые слова: *SQL-движок, семантический анализ, оптимизация запросов, архитектура СУБД, высокопроизводительные вычисления, разбор запросов (parsing) и синтаксическое дерево (AST).*

HIGH-PERFORMANCE SQL ENGINE BASED ON SEMANTIC ANALYSIS

Khodiev Shukhrat Ilkhamovich[1], Buriev Iskandar I.[2], Jumaniyozov Donyorbek B.[3],

[1] Associate Professor, Mirzo Ulugbek National University of Uzbekistan, Tashkent, PhD in Technical Sciences
[2] Student, Mirzo Ulugbek National University of Uzbekistan, Tashkent
[1,2,3] E-mail: ilch17333@gmail.com

Abstract. The increasing complexity of modern DBMS architectures and growing volumes of processed data frequently turn classical query execution methods into a critical bottleneck for information systems. This paper discusses the design and software implementation of a high-performance relational SQL engine that utilizes semantic analysis and rule-based optimization (RBO) techniques. A strict pipelined system architecture is described, which includes an AST parser, a binder, and a query tree optimizer employing appropriate optimization algorithms. The study justifies the rejection of the classical iterator execution model (Volcano) in favor of a batch-vectorized columnar approach implemented using the NumPy library tools. The integration of telemetry and visualization modules via Graphviz allows the developed engine to be used as a transparent platform for further scientific research in the field of system software. Furthermore, developing a custom SQL engine provides an inside look into the mechanics of transforming high-level queries into fast, low-level operations.

Keywords: *SQL engine, semantic analysis, query optimization, DBMS architecture, high-performance computing, query parsing, Abstract Syntax Tree (AST).*

Введение

Мощные ресурсы, а также современные технологии создания ПО на основе компонентов (распределенное программирование, компонентный дизайн и другие) – как набор стандартов, инструментов и библиотек, предназначенных для разработки приложений разных типов и связанных друг с другом использованием языка программирования, не способны стать панацеей от последствий некачественного программного кода. В условиях, когда коммерческие и научные организации ежедневно аккумулируют терабайты транзакционных и производственных данных, традиционные архитектурные решения СУБД перестают отвечать требованиям производительности реального времени. Экспоненциальный рост объемов информации и усложнение аналитических сценариев предопределяют необходимость разработки принципиально новых подходов к анализу и оптимизации программных конструкций. В данной ситуации именно глубокий семантический анализ и интеллектуальная оптимизация инструкций становятся ключевыми инструментами повышения реальной пропускной способности систем обработки данных (Дейт, К. Дж. 2019).

Несмотря на развитую теоретическую базу, в области системного программирования наблюдается дефицит открытых учебных и исследовательских реализаций, интегрирующих полный конвейер обработки SQL-запросов в рамках прозрачной модульной архитектуры. Существующие прототипы зачастую охватывают лишь изолированные этапы (например, исключительно синтаксический разбор или только оптимизацию планов) либо реализованы на языках, ограничивающих возможности интроспекции и визуализации внутренних процессов. Промышленные СУБД, такие как PostgreSQL, MySQL или Oracle, состоят из миллионов строк кода, что создает непреодолимый барьер для начинающих исследователей и маскирует фундаментальную связь между теорией компиляторов и ее практическим воплощением. Настоящая работа призвана ликвидировать данный разрыв, предлагая законченную сквозную реализацию — от текстового представления декларативного запроса до измеримых результатов бенчмарка (Коулинг, Э. 2022).

Основная часть

Семантический анализ в контексте реляционных движков представляет собой не просто валидацию синтаксиса, а критически важный слой формирования логики системы. Он обеспечивает разрешение идентификаторов, валидацию типов данных и выявление логических противоречий на ранних стадиях обработки. Подобный подход позволяет отклонять некорректные запросы (например, ошибочную агрегацию несоответствующих типов столбцов) на этапе компиляции, полностью исключая трудновоспроизводимые ошибки времени выполнения (runtime errors) в эксплуатируемых системах.

Созданный движок представляет собой верифицированный программный инструмент, пригодный для интеграции в качестве встраиваемой (embedded) СУБД в исследовательские и учебные проекты. При отсутствии жестких требований к промышленной отказоустойчивости система обеспечивает полноценный SQL-интерфейс, поддерживающий операции фильтрации, предикатной оптимизации, агрегации, сортировки и группировки данных. В рамках данного исследования SQL рассматривается как специализированная вычислительная среда, а ключевой целью является разработка ядра, способного эффективно трансформировать абстрактные декларативные конструкции в оптимальный физический план выполнения низкоуровневых команд (Ахо, А. В.,2021, Гарсиа-Молина, 2019).

Процесс связывания (Binding) и разрешение идентификаторов

Роль Binder в конвейере обработки запроса. Если парсер формирует абстрактное синтаксическое дерево (AST), опираясь исключительно на текст запроса, то этап семантического связывания (Semantic Binding) отвечает за наполнение этого дерева контекстом. Парсер – принципиально «слепой» инструмент: он проверяет лишь грамматическую правильность конструкций, но не знает ничего о реальных данных. Узел ColumnRef(name='salary') для парсера – просто строка. Он понятия не имеет, существует ли такая колонка, в какой таблице она находится, какого она типа и можно ли применять к ней агрегатную функцию SUM.

Именно здесь в работу вступает Binder. Его задача – превратить «синтаксический» AST в «семантический» логический план, в котором каждый идентификатор однозначно привязан к конкретному объекту схемы. После прохода Binder система знает не просто «пользователь написал salary», а «это колонка salary таблицы employees, тип FLOAT, не NULL, принадлежащая текущей области видимости». Такая точность является обязательным условием для корректной работы всех последующих этапов – оптимизатора, исполнителя и визуализатора.

Место Binder в конвейере строго зафиксировано: он работает после парсера (AST уже построен) и до оптимизатора (план ещё не оптимизирован). Это разделение не случайно: оптимизатор работает с логическими операторами и применяет алгебраические преобразования – ему важна семантика, а не синтаксис. Если Binder не выполнит свою работу, оптимизатор окажется в ситуации, когда он не может принять ни одного обоснованного решения: он не знает, к каким таблицам относятся столбцы, нельзя применить ни Column Pruning, ни Predicate Pushdown (Дейт, К. Дж, 2019).

Конвейерная архитектура и семантический анализ запросов

Архитектура разработанного высокопроизводительного SQL-движка базируется на принципе строгой конвейеризации и декомпозиции процесса обработки декларативного запроса на изолированные функциональные модули. Традиционный подход, объединяющий синтаксический разбор и семантическую валидацию в рамках единого прохода, неизбежно приводит к созданию монолитного и трудноподдерживаемого программного кода. Для решения этой проблемы в проектируемой системе процесс трансформации запроса жестко разделен на три последовательных этапа, включающих абстрактный синтаксический разбор, логическое связывание и оптимизацию плана. На первом этапе текстовый SQL-запрос подвергается лексическому и синтаксическому анализу, результатом которого является построение абстрактного синтаксического дерева. Однако само по себе синтаксическое дерево является лишь грамматическим каркасом и не содержит информации о реальной структуре данных (Кузнецов, С.Д., 2022).

Оптимизация на основе правил (Rule-Based Optimization)

После этапа связывания сформированный логический план передается в модуль оптимизации. В рамках данного проекта ставка была сделана на оптимизацию на основе правил, использующую фундаментальные законы реляционной алгебры для эквивалентных преобразований дерева запросов. Это позволило избежать высоких вычислительных накладных расходов, свойственных стоимостным оптимизаторам, которые требуют постоянного сбора и актуализации сложной системной статистики. Основной фокус при реализации оптимизатора был направлен на два критически важных правила трансформации, среди которых ключевое место занимают проталкивание предикатов и усечение столбцов (Кузнецов, С.Д., 2022).

Логика работы проталкивания предикатов заключается в максимально раннем выполнении операций фильтрации на нижних уровнях дерева запроса, как можно ближе к физическим источникам данных. Перенос фильтра под операции соединения позволяет отсечь избыточные строки до начала дорогостоящей фазы сопоставления. В свою очередь, усечение столбцов удаляет из плана выполнения все поля, которые не участвуют в финальной выборке, условиях фильтрации или группировки. Практическая реализация данных правил доказала, что наивный декларативный SQL-запрос может быть переформатирован алгоритмически с высокой эффективностью. Устранение избыточных кортежей и неиспользуемых полей на ранних стадиях обработки позволяет на порядки сократить объем сканируемой оперативной памяти и снизить общую нагрузку на шину данных процессора (Кузнецов, С.Д., 2022).

Векторизованная колоночная модель исполнения

Центральной задачей при проектировании исполнительного ядра стал выбор модели обработки данных. Традиционная итераторная модель выполнения, восходящая к классической архитектуре Volcano, предполагает обработку данных по принципу одной записи за один раз. Однако аналитическое исследование выявило критическое слабое место данной модели при реализации на интерпретируемых языках, в частности на Python. Колоссальные накладные расходы на динамическую диспетчеризацию и вызовы магических методов интерфейса итераторов при обработке миллионов строк фактически нивелируют эффективность любых алгоритмов (Иванов, И.И.,2022).

Для преодоления этого барьера в разработанном движке был осуществлен переход к пакетно-векторизованному колоночному подходу. Вместо обработки отдельных строк таблицы данные группируются в фиксированные пакеты и организуются по столбцам. Физическим базисом для реализации векторизованной модели послужила интеграция интерфейсов библиотеки NumPy. Векторизованные операции данной библиотеки выполняются на уровне скомпилированного C-кода, что позволяет задействовать инструкции процессора SIMD и эффективно утилизировать кэш-память. В результате интерпретатор Python освобождается от тяжелых циклов обработки строк, выполняя лишь высокоуровневое управление пакетами колоночных данных.

Алгоритмическая оптимизация низкоуровневых операторов и телеметрия

Высокая производительность движка на физическом уровне достигается за счет оптимизации ключевых реляционных операторов, таких как соединение таблиц и векторизованная агрегация. Для обеспечения стабильной масштабируемости системы была поставлена задача сохранения линейно-логарифмической вычислительной сложности алгоритмов. При реализации оператора HashJoin вместо классического вложенного цикла применяется хэш-таблица, строящаяся по меньшей из соединяемых таблиц, после чего выполняется векторизованное сопоставление со второй таблицей. В операторе VectorizedAggregate критически важным оказалось использование специализированных низкоуровневых функций NumPy. В частности, для быстрого распределения данных по группам применяется бинарный поиск на основе метода np.searchsorted, а для выполнения агрегатных функций в рамках сформированных групп задействовано атомарное сложение по хэш-индексам с помощью метода np.add.at. (Кузнецов, С.Д., 2022).

Благодаря отказу от итераторов и глубокой векторизованной обработке массивов, математическая сложность выполнения тяжелых аналитических операций на больших объемах данных удерживается в строгих пределах, описываемых формулой: O(B log N), где B обозначает размер обрабатываемого пакета данных, а N отражает общий объем

записей в реляционном отношении. Это гарантирует линейный характер роста временных затрат при масштабировании массивов данных.

Важной при создании движка стало обеспечение его полной архитектурной прозрачности, что устраняет проблему черного ящика, характерную для большинства коммерческих СУБД. В систему был интегрирован модуль динамической интроспекции, позволяющий осуществлять экспорт текущего состояния планов выполнения. С помощью инструментов визуализации Graphviz и специализированного языка разметки DOT движок генерирует интерактивные схемы графов на каждом этапе трансформации, начиная от исходного синтаксического дерева и заканчивая оптимизированным физическим планом. Параллельно с этим сквозные счетчики телеметрии фиксируют метрики времени выполнения каждого оператора, объемы прокачиваемой памяти и накладные расходы на векторизацию. Наличие развитого слоя интроспекции превращает разработанный SQL-движок в открытую и предсказуемую инструментальную платформу, полностью пригодную для проведения глубокого научного поиска.

Результаты проектирования высокопроизводительного SQL-движка подтвердили эффективность выбранных архитектурных и алгоритмических решений. Строгая конвейеризация системы с разделением на AST-парсер, связыватель (Binder) и оптимизатор обеспечила глубокую модульность, что позволило реализовать сложные трансформации дерева запросов без усложнения грамматического разбора, а применение DataClasses минимизировало риски ошибок типизации. В области оптимизации внедрение подходов на основе правил (RBO) — в частности, алгоритмов Predicate Pushdown и Column Pruning — доказало возможность эффективного переформатирования «наивных» SQL-запросов, что привело к многократному сокращению объема сканируемой памяти без привлечения ресурсоемких стоимостных моделей (CBO).

Заключение

Детально рассмотрены архитектурные аспекты проектирования и реализации высокопроизводительного SQL-движка. Переход от классических жестких методов синтаксического разбора к оптимизации на основе глубокого семантического анализа доказал свою высокую эффективность, особенно в условиях пиковых нагрузок и одновременного обращения тысяч клиентов. Разработанный подход позволяет системе гибко адаптироваться к логической структуре данных и находить наиболее рациональные пути выполнения аналитических задач, что нивелирует проблему «узких мест» и обеспечивает стабильно высокую скорость агрегации миллионов строк при минимальных вычислительных затратах.

Практическая реализация собственного движка позволила изнутри изучить механику трансформации высокоуровневых декларативных запросов в быстрые низкоуровневые операции, создав надежный фундамент для развития отечественного системного программного обеспечения. Расширение созданной архитектуры за счет интеграции механизмов динамической компиляции (JIT) и адаптивного перестроения планов выполнения непосредственно в процессе обработки потоковых данных имеет большое значение.

Список литературы

Журнальные статьи

1. Дейк, Д. Современные методы оптимизации SQL-запросов на основе семантических моделей данных / Д. Дейк, С. А. Петров. – Текст: непосредственный // Вестник компьютерных и информационных технологий. – 2023. – Т. 20, № 4. – С. 15–24.

2. Иванов, И. И. Архитектура распределенных высокопроизводительных SQL-движков для аналитической обработки данных / И. И. Иванов, П. П. Сидоров. – Текст : непосредственный // Программирование. – 2024. – № 2. – С. 45–58.
3. Кузнецов, С.Д. Оптимизация запросов в объектно-реляционных СУБД: семантический аспект / С. Д. Кузнецов. – Текст: электронный // Труды Института системного программирования РАН. – 2022. – Т. 34, вып. 1. – URL: https://www.ispras.ru/proceedings/.

Книги

1. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт ; перевод с английского под редакцией под ред. К. А. Птицына. – 8-е издание. – Москва : Вильямс, 2019. – 1328 с. – ISBN 978-5-8459-2045-4.
2. Коулинг, Э. Высокопроизводительный SQL. Оптимизация планов выполнения и архитектура движков СУБД / Э. Коулинг; перевод с английского И. В. Рузманова. – Москва: ДМК Пресс, 2022. – 412 с. – ISBN 978-5-97060-951-4.
3. Гарсиа-Молина, Г. Системы баз данных. Полный курс : учебное пособие / Г. Гарсиа-Молина, Д. Ульман, Д. Уидом ; перевод с английского под редакцией В. В. Бакарева. – Москва : Вильямс, 2019. – 1088 с. – ISBN 978-5-8459-2041-6. (Здесь подробно описаны алгоритмы оптимизации запросов, логические и физические планы выполнения).
4. Ахо, А. В. Компиляторы: принципы, технологии и инструментарий / А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман ; перевод с английского под редакцией А. А. Матросова. – 2-е издание. – Санкт-Петербург : Диалектика, 2021. – 1184 с.