

ClickHouse

Архитектурный документ

Авторы документа:

Абрамов Александр Сергеевич (БПИ213)

Коледаев Алексей Дмитриевич (БПИ213)

Преподаватель группы:

Мицюк Алексей Александрович

0. Раздел регистрации изменений

Версия документа	Дата изменения	Описание изменения	Автор изменения
1.0.0	22.04.2024	Создание документа	Абрамов Александр Коледаев Алексей
2.0.0	01.06.2024	<p>Внесение доработок архитектурного решения системы, предложенных в результате его оценки методом анализа архитектурных компромиссов:</p> <ol style="list-style-type: none">1. Модифицировано логическое представление (соответствующие п. 4.2 диаграммы п. 6.2)2. Описаны технические решения 4 и 5 (пп. 5.4, 5.5)	Абрамов Александр Коледаев Алексей

1. Введение

1.1. Название проекта

ClickHouse

1.2. Задействованные архитектурные представления

Настоящий документ содержит следующие архитектурные представления:

1. Представление прецедентов, показывающее основные услуги системы и ключевых акторов, взаимодействующих с системой при оказании этих услуг.
2. Логическое представление, описывающее проектное устройство системы, её основные классы и их разделение на пакеты и подсистемы.
3. Представление архитектуры процессов, показывающее ключевые аспекты взаимодействия классов и акторов системы во время её работы.
4. Представление развёртывания, описывающее механизмы запуска системы на физических или логических узлах, а также методы горизонтального и вертикального масштабирования системы.
5. Представление архитектуры данных, описывающее принципы получения, обработки и хранения данных системой, позволяющие обеспечить необходимый уровень надёжности системы с точки зрения работы с данными.
6. Представление реализации и разработки, устанавливающее требования и дающее рекомендации по организации процессов разработки и развёртывания системы на тестовом стенде.
7. Представление производительности, описывающее основные алгоритмы и структуры данных, обеспечивающие достаточную производительность системы.
8. Атрибуты качества системы, представляющие набор качественных и количественных характеристик, позволяющих оценить, насколько разработанная система удовлетворяет установленным нефункциональным требованиям.

Представление архитектуры безопасности не задействовано в настоящем документе, так как специальных требований к безопасности системы не предъявляется.

Структура документа представлена в содержании:

1. Введение.....	3
1.1. Название проекта.....	3
1.2. Задействованные архитектурные представления.....	3
1.3. Контекст задачи и среда функционирования системы.....	6
1.4. Рамки и цели проекта.....	7

2. Архитектурные факторы.....	8
2.1. Ключевые заинтересованные лица.....	8
2.2. Ключевые требования к системе.....	8
2.3. Ключевые ограничения.....	9
3. Общее архитектурное решение.....	10
3.1. Принципы проектирования.....	10
4. Архитектурные представления.....	11
4.1. Представление прецедентов.....	11
4.2. Логическое представление.....	11
4.3. Представление архитектуры процессов.....	11
4.4. Представление развертывания.....	11
4.5. Представление архитектуры данных.....	11
4.6. Представление реализации и разработки.....	12
4.7. Представление производительности.....	13
4.7.1. Организация хранения данных.....	13
4.7.2. Синтаксический разбор текста запроса.....	13
4.7.3. Основные принципы обработки данных.....	14
4.7.4. Организация распределённых вычислений.....	14
4.8. Атрибуты качества системы.....	14
4.8.1. Объем данных и производительность системы.....	15
4.8.2. Гарантии качества работы системы.....	16
5. Технические описания отдельных ключевых архитектурных решений.....	17
5.1. Техническое решение №1: монолитная архитектура.....	17
5.1.1. Проблема.....	17
5.1.2. Идея решения.....	17
5.1.3. Факторы.....	17
5.1.4. Решение.....	17
5.1.5. Мотивировка.....	17
5.1.6. Неразрешенные вопросы.....	18
5.1.7. Альтернативы.....	18
5.2. Техническое решение №2: возможность загрузки данных из сторонних СУБД.....	18
5.2.1. Проблема.....	18

5.2.2. Идея решения.....	18
5.2.3. Факторы.....	18
5.2.4. Решение.....	18
5.2.5. Мотивировка.....	19
5.2.6. Неразрешенные вопросы.....	19
5.2.7. Альтернативы.....	19
5.3. Техническое решение №3: SQL.....	20
5.3.1. Проблема.....	20
5.3.2. Идея решения.....	20
5.3.3. Факторы.....	20
5.3.4. Решение.....	20
5.3.5. Мотивировка.....	20
5.3.6. Неразрешенные вопросы.....	21
5.3.7. Альтернативы.....	21
5.4. Техническое решение №4 (Доработка №1): применение традиционных шаблонов проектирования.....	21
5.4.1. Проблема.....	21
5.4.2. Идея решения.....	21
5.4.3. Факторы.....	21
5.4.4. Решение.....	22
5.4.5. Мотивировка.....	22
5.4.6. Неразрешенные вопросы.....	22
5.4.7. Альтернативы.....	23
5.4.8. Описание доработки.....	23
5.4.9. Исходный код доработки.....	23
5.4.10. Оценка результативности доработки.....	23
5.5. Техническое решение №5 (Доработка №4): широкое использование принципов ООП.....	24
5.5.1. Проблема.....	24
5.5.2. Идея решения.....	24
5.5.3. Факторы.....	24
5.5.4. Решение.....	24
5.5.5. Мотивировка.....	25
5.5.6. Неразрешенные вопросы.....	25
5.5.7. Альтернативы.....	25

5.5.8. Описание доработки.....	25
5.5.9. Исходный код доработки.....	26
5.5.10. Оценка результативности доработки.....	26
6. Приложения.....	27
6.1. Словарь терминов.....	27
6.2. Задействованные диаграммы.....	27

1.3. Контекст задачи и среда функционирования системы

Хранение и обработка больших массивов данных является одной из важных современных задач в области компьютерных наук. В частности, наблюдается большой спрос на выполнение аналитических запросов в реальном времени, что сопряжено с необходимостью эффективной обработки и агрегации малой доли информации о большом количестве записей. Такой сценарий использования характеризуется рядом особенностей, накладывающих специальные ограничения на возможное решение:

1. Подавляющая доля запросов выполняет чтение и агрегацию данных;
2. При обработке запросов требуется большая пропускная способность: миллионы и даже миллиарды строк в секунду;
3. Результат выполнения значительно меньше количества исходных данных;
4. Обновление данных происходит сравнительно редко, большими пакетами;
5. Обновлению существующих записей предпочитают создание новых записей;
6. Значения в столбцах относительно небольшие, не превышают нескольких десятков байт;
7. Транзакционное выполнение операций не требуется.

Описанные особенности делают невозможным использование традиционных СУБД – PostgreSQL, MongoDB, Redis и др. – для решения поставленной задачи. Более того, даже системы, направленные на использование в аналитических целях, такие как Spark и Hadoop, не всегда отвечают потребностям бизнеса и не могут быть использованы в отдельных случаях, требующих чтения лишь малой доли столбцов из большого массива данных. И даже столбцовые СУБД (Amazon Redshift, Google PowerDrill и др.), которые, казалось бы, должны решать эту проблему, не всегда достигают достаточного уровня производительности по многим причинам: хранение данных в оперативной памяти, отсутствие векторизованного выполнения команд, чрезмерное использование кодогенерации при выполнении запросов или отсутствие поддержки индекса по основному ключу.

Таким образом, требуется разработать продукт, способный агрегировать отдельные значения о большом количестве записей без необходимости их полного чтения. Система не должна хранить и обрабатывать какие-либо лишние данные, включая длины значений, если это не является необходимым. Более того, система должна предоставлять удобный интерфейс

взаимодействия, не требующий от аналитика специальной квалификации. Это может быть выполнено путём использования одного из существующих общеизвестных языков запросов: SQL, MDX, J, K, APL и др.

1.4. РАМКИ И ЦЕЛИ ПРОЕКТА

Целью проекта является разработка уникального решения, позволяющего эффективно агрегировать большие массивы данных путём организации колоночного хранения информации и её обработки в виде набора столбцов, а не строк, как реализовано в большинстве существующих СУБД. Это многократно уменьшит количество дорогостоящих операций ввода-вывода, необходимых для выполнения аналитического запроса, а также позволит использовать современные алгоритмы сжатия данных, что дополнительно уменьшит затраты на получение информации с энергонезависимых запоминающих устройств.

При этом основной задачей системы является эффективная обработка запросов на чтение, что позволяет не уделять особенного внимания оптимизации операций редактирования и удаления записей, хотя они и должны быть поддержаны для удовлетворения законодательству. Более того, поддержка транзакционного выполнения операций не требуется, хотя и бывает полезна в отдельных ситуациях, из-за чего может быть реализована позже.

Также важно заметить, что в аналитическом хранилище не подразумевается содержание секретной информации, включая персональные данные пользователей. Это позволяет "переложить" ответственность за сохранность данных на внешнюю сетевую конфигурацию и не предпринимать специальных мер по обеспечению безопасности, что позволяет дополнительно повысить производительность.

В рамках проекта, описанного настоящим документом, была произведена разработка общей архитектуры системы со следующими ограничениями:

1. Получение запросов от пользователей производится только по протоколу GRPC;
2. Хранение данных производится в сторонней системе на базе PostgreSQL с их последующей выгрузкой в ClickHouse;
3. Программа обрабатывает только простейшие запросы на чтение;
4. Обработка запросов производится однопоточно на одном устройстве; специальная проработка механизмов распределения вычислений оставлена за рамками настоящего проекта.

При этом система должна быть легко расширяемой и должна обеспечивать возможность последующего добавления других протоколов взаимодействия (TCP, HTTP), дополнительных источников данных (Redis, S3 и др.), включая организацию собственного хранилища, а также обработки запросов других типов (вставки, редактирования, удаления, репликации, резервного копирования и др.).

2. Архитектурные факторы

2.1. Ключевые заинтересованные лица

Действующее лицо	Заинтересованность в системе
Бизнес / Владелец	Извлечение прибыли, простота модификации и расширения, полнота предоставляемых функций, отказоустойчивость
Разработчик	Простота поддержки, возможность введения новой функциональности без значительных изменений архитектуры, прозрачность внутренних API
Тестировщик	Удобство верификации и валидации продукта, простота проведения регрессионного тестирования и тестирования новых функций
Поддержка пользователей	Простота диагностирования ошибок, возможность получения состояния и лог-файлов системы
Администратор системы	Простота развертывания и конфигурирования системы, предсказуемость использования ресурсов, возможность мониторинга состояния системы
Сторонний разработчик	Удобство интеграции с системой, простота написания запросов, прозрачность внешних API и ясность документации к ним
Сторонняя система и конечный пользователь	Надёжность и отказоустойчивость системы, невозможность нарушения её работоспособности при любых входных данных, удобство формирования запросов и высокая скорость их выполнения

2.2. Ключевые требования к системе

Ключевые требования к системе определяются контекстом и целями проекта:

1. Продукт должен реализовывать основные функции системы управления базами данных: возможность создания, чтения, редактирования и удаления записей. Аналитические запросы, требующие агрегации информации о большом количестве данных, должны обрабатываться особенно эффективно (с пропускной способностью, составляющей миллионы строк в секунду), обеспечивая возможность выполнения в реальном времени.
2. Система должна предоставлять возможность взаимодействия по протоколам TCP, HTTP, GRPC. Добавление дополнительных протоколов взаимодействия должно быть возможно в будущем без значительных изменений архитектуры.
3. Продукт должен иметь возможность работы с данными, хранящимися во внешнем хранилище на базе PostgreSQL, S3, Redis или др. Добавление дополнительных источников данных, включая внутреннее хранилище, должно быть возможно в будущем без значительных изменений архитектуры.

4. Система должна легко масштабироваться как горизонтально (распределённая обработка запросов на многих серверах, шардирование), так и вертикально (обработка запросов с использованием всех ресурсов системы).

2.3. Ключевые ограничения

Цель проекта – разработка системы выполнения аналитических запросов к большим массивам данных – позволяет ограничить реализацию системы лишь частью возможностей, доступных в традиционных СУБД. В частности, не требуется поддержка следующих функций:

1. Эффективная обработка подзапросов;
2. Транзакционное выполнение запросов;
3. Внешние ключи и другие алгоритмы валидации данных и поддержания их целостности;
4. “Специфические” алгоритмы объединения таблиц, такие как RIGHT JOIN и FULL OUTER JOIN;
5. Обеспечение привилегированного доступа к отдельным записям таблиц;
6. Хранимые процедуры, функции, триггеры;
7. Генерируемые столбцы и столбцы с автоматическим увеличением значения.

Помимо прочего, особенное внимание должно быть уделено обеспечению высокой производительности выполнения аналитических запросов чтения данных и их агрегации по значениям отдельных столбцов. Более того, важно предусмотреть возможность дальнейшего расширения системы, а также реализовать поддержку простого, удобного для большинства пользователей, включая сотрудников аналитических отделов компаний, языка запросов.

3. Общее архитектурное решение

Общее архитектурное решение основано на монолитной архитектуре с применением объектно-ориентированного подхода и большого количества традиционных шаблонов проектирования, таких как "единая точка входа", "цепочка ответственности", "одиночка", "фабрика", "внедрение зависимости", "интерпретатор", "строитель", "адаптер" и "команда".

В рамках архитектуры широко используются и основные принципы объектно-ориентированного подхода, включая абстракцию и полиморфизм, что позволяет обеспечить модифицируемость и расширяемость системы путём сокрытия деталей реализации большинства сущностей за общими интерфейсами взаимодействия с ними. Таким образом реализованы алгоритмы синтаксического разбора запроса, построения дерева этапов его выполнения, а также методы непосредственных преобразований данных и их ввода-вывода из различных источников, включая сторонние базы и хранилища данных.

Разработанная архитектура позволяет обеспечить высокую производительность при обработке запросов, не требуя излишних накладных расходов на организацию сетевого взаимодействия, валидацию и интерпретацию получаемых данных. А организация понятной цепочки выполнения запроса (разбор, интерпретация, чтение входных данных, выполнение, запись выходных данных) и инкапсуляция всех алгоритмов непосредственной манипуляции с данными в одном пакете с общим интерфейсом позволяет обеспечить надёжность системы и ясность её исходного кода, хотя ряд решений, необходимых для успешного связывания частей такой архитектуры, могут быть и не столь прозрачны для разработчика, незнакомого с системой.

Более того, описанная архитектура способствует как горизонтальному, так и вертикальному масштабированию, легко интегрируясь с различными провайдерами облачных и распределённых вычислительных ресурсов. Такая гибкость позволяет системе динамически подстраиваться под тренды нагрузки, сохраняя при этом высокую производительность и стабильность работы, но не расходуя излишние ресурсы.

Особое внимание уделяется также удобству мониторинга и логирования, что делает процесс отладки и диагностики ошибок более прозрачным и контролируемым. Такие меры предоставляют возможность более глубокого понимания внутренних процессов системы и способствуют эффективному устранению возникающих неисправностей.

3.1. Принципы проектирования

В рамках настоящего архитектурного документа не было выявлено использование каких-либо специальных подходов к проектированию архитектуры системы разработчиками ClickHouse. По информации из открытых источников, основные архитектурные и технические решения принимались или утверждались главным разработчиком системы по мере разработки системы и написания исходного кода.

4. Архитектурные представления

4.1. Представление прецедентов

Основные прецеденты использования системы и задействованные в них акторы описываются диаграммой прецедентов, представленной на рисунке 1 приложения 2.

4.2. Логическое представление

Устройство системы представлено диаграммами классов и пакетов, изображенными на рисунках 2 – 3 приложения 2.

4.3. Представление архитектуры процессов

Основной процесс, выполняемый системой, – получение, обработка и ответ на запрос пользователя, полученный по протоколу GRPC, – представлен на диаграммах последовательности (рис. 4 – 10 прил. 2). Специальная проработка организации процессов в рамках рассматриваемой части системы не требуется, асинхронный код в проекте не используется. Методы параллелизации вычислений и задач системы, реализованные в продукте, оставлены за рамками проекта.

4.4. Представление развертывания

Система разворачивается на одном устройстве под управлением Linux, FreeBSD, MacOS или Windows WSL путём запуска исполняемого файла, который можно получить командой `curl https://clickhouse.com/ | sh`. Отдельно устанавливаемые компоненты и модули отсутствуют.

Обновление системы производится путём ручного сохранения пользовательских данных и параметров конфигурации, удаления старой версии и установки новой версии системы. Автоматическое обновление системы не предусмотрено.

При необходимости масштабирования копии системы могут быть развёрнуты на необходимом количестве физических или логических узлов, связанных посредством утилиты ClickHouse Keeper или сторонней системы ZooKeeper.

Подробное описание основных способов развёртывания, масштабирования и конфигурирования системы доступно на официальном сайте системы по адресам <https://clickhouse.com/docs/ru/getting-started/install>, <https://clickhouse.com/docs/ru/operations/clickhouse-keeper> и <https://clickhouse.com/docs/ru/operations/configuration-files>.

4.5. Представление архитектуры данных

Большую часть данных, обрабатываемых системой, представляет произвольная пользовательская информация, получаемая из сторонних источников данных (PostgreSQL, Redis,

S3 и др.) или сохраняемая ClickHouse самостоятельно при выполнении команды INSERT. Репликация и резервное копирование данных доступны посредством параметров конфигурации и команд во время выполнения наравне с командами чтения, записи, редактирования и удаления.

Более того, определённые данные (информация о пользователях СУБД, настроенных уровнях доступа и др.) могут размещаться в системных таблицах наряду с пользовательскими – в отдельной базе данных.

Гонок данных, требующих особого внимания, в рамках анализируемого проекта не ожидается. При дальнейшей интеграции многопоточной обработки запросов может потребоваться придание особенного внимания разрешению гонок данных, которые могут привести к нарушению целостности информации.

4.6. Представление реализации и разработки

При работе над проектом допускается использование любых инструментов и сред разработки на усмотрение исполнителя. Рекомендованные среды разработки – CLion, KDevelop, QT Creator, Sublime Text, Visual Studio Code, Kate.

Исходный код системы доступен на Github по адресу <https://github.com/ClickHouse/ClickHouse>. Для хранения исходного кода и удобной работы с ним следует использовать систему контроля версий Git. Разработка системы ведётся на языке C++ с использованием компилятора Clang версии 11 или выше с применением стиля кода, описанного на официальном сайте, доступном по адресу <https://clickhouse.com/docs/ru/development/style>. Краткое описание устройства проекта с исходным кодом системы доступно по адресу <https://clickhouse.com/docs/ru/development/architecture>.

Для организации процесса разработки используется Github Issues, доступный по адресу <https://github.com/ClickHouse/ClickHouse/issues>. При внесении изменений в исходный код разработчик должен склонировать репозиторий на локальную машину и вести разработку в полученной копии, регулярно синхронизируя изменения с основным репозиторием. По окончании реализации новая версия исходного кода должна быть загружена в основной репозиторий посредством "Pull Request", после чего изменения будут рассмотрены сотрудниками ClickHouse, а Pull Request будет помечен ярлыком "can be tested", после чего он должен будет пройти все существующие процессы тестирования. Только в этом случае изменения могут быть интегрированы в основную версию репозитория. Для упрощения процесса тестирования изменений могут быть использованы анонимизированные данные Яндекс.Метрики, доступные для скачивания по адресам https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz, https://datasets.clickhouse.com/visits/tsv/visits_v1.tsv.xz.

Подробная инструкция для разработчиков доступна на официальном сайте ClickHouse по адресу <https://clickhouse.com/docs/ru/development/developer-instruction>.

4.7. Представление производительности

4.7.1. Организация хранения данных

Для обеспечения высокой производительности хранения и обработки информации в ClickHouse используется структура данных MergeTree, подробное описание принципов работы которой доступно в официальной документации по адресу <https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree>.

В основе этой структуры данных лежит набор деревьев, содержащих необходимые данные с информацией о первичном ключе, что позволяет эффективно выполнять основные операции, необходимые для аналитической СУБД: фильтрация, сортировка и агрегация.

Добавление данных в структуру производится по частям с дальнейшим объединением записанных блоков в фоновом режиме, оптимизируя количество занимаемого места на диске и улучшая производительность типичных запросов.

4.7.2. Синтаксический разбор текста запроса

Одним из ключевых этапов цепочки обработки запроса к ClickHouse является его синтаксический разбор и представление в виде некоторой внутренней структуры, не зависящей от пользователя. В качестве такой структуры было использовано абстрактное синтаксическое дерево (Abstract Syntax Tree, AST), представленное интерфейсом IAST.

При получении запроса в текстовом виде система инициирует экземпляр класса ParserQuery, который производит синтаксический разбор полученной строки с использованием других реализаций интерфейса IParser, в результате чего производится построение AST, являющегося временным представлением запроса, не зависящем от пользовательского ввода. Каждая вершина полученного дерева соответствует некоторой функции или оператору, который был обнаружен в исходном запросе.

Для AST существуют большие возможности анализа и оптимизации, реализованные в ClickHouse пакетом Analyzer, производящем проверку корректности (отсутствия синтаксических и семантических ошибок, наличия используемых таблиц и столбцов, типов данных и функций), а также оптимизацию (удаление операций чтения избыточной информации, не являющейся необходимой для вычисления результата, и перестановка операций для более эффективного выполнения). Конкретные реализации описанных алгоритмов выходят за рамки настоящего документа и описаны в официальной документации по адресу <https://clickhouse.com/docs/en/guides/developer/understanding-query-execution-with-the-analyzer>.

В заключение, оптимизированное синтаксическое дерево подаётся на вход экземплярам реализаций интерфейса IInterpreter для построения конечного плана выполнения запроса, определения последовательности операций над данными и возможности их параллельного выполнения в зависимости от структуры запроса и параметров конфигурации системы.

4.7.3. Основные принципы обработки данных

Для повышения производительности и уменьшения затрат на диспетчеризацию операций обработки информации ClickHouse эффективно комбинирует два различных подхода:

1. Векторизация операций над столбцами для наилучшей эксплуатации всех возможностей и ресурсов современных процессоров, что позволяет обрабатывать большие блоки данных одновременно, значительно ускоряя агрегацию и анализ данных. При этом код операции содержит хорошо оптимизированный внутренний цикл.
2. Кодогенерация, позволяющая заранее создавать высокопроизводительный код выполнения запроса, в котором подставлены все косвенные вызовы, и использовать его без необходимости пересборки.

4.7.4. Организация распределённых вычислений

ClickHouse поддерживает организацию распределённой обработки информации, что позволяет маршрутизировать данные и запросы между несколькими физическими или логическими копиями системы для повышения производительности путём горизонтального масштабирования. Для синхронизации копий систем, распределённых на различных устройствах, используется ZooKeeper или уникальная утилита над ним – ClickHouse Keeper, основанная на алгоритме RAFT, позволяющем обеспечить высокую производительность и надёжность работы. Подробная документация утилиты и алгоритма доступны по адресу <https://clickhouse.com/docs/gu/operations/clickhouse-keeper>.

4.8. Атрибуты качества системы

Наиболее важными для системы считаются следующие атрибуты качества:

1. Производительность – система должна обеспечивать достаточную скорость выполнения запросов для использования в задачах интерактивной аналитической обработки данных.
2. Надёжность – программа не должна аварийно завершаться при любом наборе входных данных, а также должна производить обработку возникающих ошибок, при этом корректно продолжая работу.
3. Масштабируемость – система должна предоставлять возможность бесшовного горизонтального и вертикального масштабирования с поддержкой синхронизации задач и данных между копиями.
4. Простота использования – система должна предоставлять удобный интерфейс взаимодействия, не требующий от пользователя специальной квалификации.
5. Простота развёртывания – система должна легко запускаться специалистом, обладающим базовыми навыками развёртывания информационных систем, на любом устройстве под управлением операционных систем Linux, FreeBSD, MacOS и Windows WSL.

6. Модифицируемость и распределяемость процесса разработки – система должна поддерживать добавление новых источников данных, алгоритмов их обработки и протоколов взаимодействия без значительных изменений архитектуры в целом. Работу над независимыми пакетами должно быть возможно производить параллельно.
7. Интероперабельность – система должна легко интегрироваться в большинство продуктов и иметь возможность работы с данными, хранящимися в различных сторонних СУБД.
8. Наблюдаемость - система должна предоставлять достаточную для диагностирования ошибок и устранения неполадок информацию о своём состоянии во время выполнения.

4.8.1. Объем данных и производительность системы

Детальные характеристики производительности системы описаны в официальной документации, доступной по адресу <https://clickhouse.com/docs/ru/introduction/performance>. Ключевые количественные характеристики приведены далее в настоящем документе.

При условии, что данные помещаются в страничный кэш, не слишком сложный запрос обрабатывается со скоростью около 2 – 10 ГБ/с несжатых данных на одном сервере, а в отдельных случаях скорость может достигать 30 ГБ/с. Если данные не помещаются в страничный кэш, скорость работы зависит от скорости подсистемы ввода-вывода и коэффициента сжатия данных. При запросе столбцов размера 10 байт ожидаемая скорость обработки будет составлять 100 – 200 млн строк в секунду. При распределённой обработке запроса скорость увеличивается почти линейно при условии, что в результате агрегации или при сортировке получается не слишком большое множество строк.

Если запрос использует первичный ключ и выбирает для обработки не более сотен тысяч строк, оперируя не слишком большим количеством столбцов, ожидается задержка обработки запроса не более 50 миллисекунд при условии, что данные помещаются в страничный кэш. В ином случае задержка пропорциональна количеству операций поиска информации на дисковом накопителе.

При обработке большого количества коротких запросов, что не является типичным для ClickHouse прецедентом использования, следует рассчитывать на скорость не более 100 запросов в секунду.

Данные рекомендуется вставлять пакетами размером не менее 1000 строк, выполняя при этом не более одного запроса в секунду. При вставке в таблицу типа MergeTree из текстового файла с табуляцией в качестве разделителя скорость вставки будет в районе 50 – 200 МБ/с, что соответствует 50 – 200 тысячам строк в секунду при их размере около 1 КБ. При обработке высокооптимизированных для аналитических задачах данных, например, собранных утилитой Graphite, скорость вставки может достигать 1 миллиона строк в секунду. При выполнении нескольких запросов вставка параллельно производительность растёт линейно.

4.8.2. Гарантии качества работы системы

Для обеспечения бесперебойной работы системы реализованы механизмы мониторинга её состояния во время выполнения. Основные метрики сохраняются в системных таблицах `system.metrics`, `system.events` и `system.asynchronous.metrics`. Более того, файл конфигурации позволяет настроить экспорт собираемых метрик в Graphite и Prometheus для их дальнейшей визуализации посредством Kibana, Grafana или другой системы.

Во избежание потери данных в результате сбоя или человеческой ошибки ClickHouse предоставляет возможности репликации и резервного копирования данных посредством команд `BACKUP` и `RESTORE`, настраиваемых в файле конфигурации. Подробная документация соответствующих процессов доступна на официальном сайте системы по адресам <https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/replication> и <https://clickhouse.com/docs/en/operations/backup>.

Для удобства диагностирования и исправления ошибок ClickHouse реализует механизмы логирования с поддержкой OpenTelemetry, настраиваемые в файле конфигурации как описано на странице, доступной по адресу https://clickhouse.com/docs/ru/operations/server-configuration-parameters/settings#server_configuration_parameters-logger.

5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое решение №1: монолитная архитектура

5.1.1. Проблема

Система должна обеспечивать высокую скорость выполнения запросов, минимизируя накладные расходы на передачу данных, а развёртывание системы должно быть упрощено насколько это возможно для обеспечения возможности запуска без специальных навыков.

5.1.2. Идея решения

Во избежание ненужного копирования и манипулирования данными, используется монолитная архитектура с явным разделением ответственности за различные этапы обработки запроса, включая хранение и обработку информации.

5.1.3. Факторы

1. Логика хранения “сырых” данных должна быть изолирована от остальной системы в отдельных классах и пакетах, контролирующих все операции чтения и записи во избежание излишних расходов на работу с информацией.
2. Развёртывание системы должно производиться путём загрузки и запуска одного или нескольких готовых исполняемых файлов в строгом соответствии с документацией. Вероятность возникновения непредвиденных ошибок должна быть сведена к минимуму.

5.1.4. Решение

Система построена на базе монолитной архитектуры с построением цепочки ответственности обработки запроса, каждый этап которой отвечает строго за определенную задачу, не производя при этом лишних операций. Вся логика непосредственной обработки данных скрыта за общими интерфейсами, описывающими лишь минимальный набор методов, необходимых и достаточных для выполнения одной и только одной подзадачи. При этом такая инкапсуляция бизнес-логики позволила реализовать алгоритмы параллельного и распределённого выполнения вычислений.

5.1.5. Мотивировка

Монолитная архитектура минимизирует затраты на передачу данных между компонентами, что способствует обеспечению высокой производительности системы. Инкапсуляция логики обработки данных за общими интерфейсами позволяет без труда производить аудит ключевых алгоритмов, выявлять и устранять этапы, выполняющие чрезмерно “дорогие” операции, оказывающие значимое влияние на общую производительность системы, а также обеспечивает

возможность реализации методов в изоляции от остальной системы с применением независимых архитектурных и технических подходов, обеспечивающих высокую пропускную способность.

5.1.6. Неразрешенные вопросы

Одним из вопросов, которым следует уделять особое внимание при разработке и расширении системы, является снижение связности классов и пакетов. Несмотря на то, что монолитная архитектура предполагает высокую связность компонентов, чрезмерное их соединение без явной причины может привести к появлению логически необоснованных зависимостей, усложнению кода и снижению его поддерживаемости.

5.1.7. Альтернативы

Основной альтернативой является сервис-ориентированная архитектура в целом и микросервисная архитектура в частности. Тем не менее, применение такого подхода неизбежно потребует передачи больших объемов данных по сети, что негативно скажется на производительности системы, а также усложнит её развёртывание, так как потребует от инженера детальной настройки как отдельных сервисов, так и их взаимодействия.

5.2. Техническое решение №2: возможность загрузки данных из сторонних СУБД

5.2.1. Проблема

Система должна быть интероперабельна – без труда интегрироваться в готовые информационные системы и использовать существующие данные без необходимости их переноса в ClickHouse в ручном режиме.

5.2.2. Идея решения

Система должна иметь возможность подключения к наиболее популярным сторонним СУБД (PostgreSQL, Redis, S3 и др.) и работы с данными, сохранёнными в них.

5.2.3. Факторы

1. Выгрузка данных должна производиться исключительно в автоматическом режиме при выполнении запросов.
2. Реализация поддержки дополнительных источников данных должна быть возможна без внесения изменений в общую архитектуру системы.

5.2.4. Решение

Для хранения, получения и представления информации зафиксирован набор интерфейсов, описывающих протоколы чтения данных из СУБД, таблиц и столбцов соответственно, а алгоритмы непосредственного взаимодействия со сторонними хранилищами полностью инкапсулированы в

классах-реализациях этих интерфейсов. При этом возможна работа как с отдельными таблицами, так и с целыми СУБД.

5.2.5. Мотивировка

Описанное решение позволяет не только скрыть детали реализации взаимодействия со сторонней СУБД за общим интерфейсом, но и обеспечивает возможность расширения системы для поддержки других хранилищ. Для этого требуется создать реализации необходимых интерфейсов, содержащие лишь логику, являющуюся специфичной для добавляемого источника данных. Такой подход позволяет обеспечить модифицируемость системы без ущерба для других атрибутов качества, а также не усложняет структуру исходного кода и его поддерживаемость.

5.2.6. Неразрешенные вопросы

Ключевой проблемой, требующей дополнительного анализа, является обеспечение производительности, масштабируемости и надёжности системы при работе со сторонними данными. Важно уделить особенное внимание эффективной реализации алгоритмов чтения из сторонних СУБД и конвертации данных в формат, пригодный для обработки. Взаимодействие со сторонними хранилищами также создаёт дополнительную точку отказа системы, что может привести к снижению надежности самого ClickHouse. А масштабирование ClickHouse приведёт к значительному росту числа запросов в подключенные сторонние хранилища, что может привести и к снижению их производительности или выходу из строя, таким образом делая непригодным к использованию и сам ClickHouse.

Не менее важно оценить и риски, связанные с безопасностью данных, содержащихся в СУБД, связанных с ClickHouse. Хотя аналитическое хранилище и не обрабатывает чувствительные данные, из-за чего специальных требований к безопасности не предъявляется, сторонние хранилища могут содержать персональные данные пользователей, которые могут по ошибке попасть в ClickHouse и быть скомпрометированы.

5.2.7. Альтернативы

Альтернативным решением является хранение всех данных внутри кластера ClickHouse, фактически используя только одну реализацию интерфейса `IStorage`, основанную на структуре данных MergeTree, описанной в п. 4.7.1 настоящего документа. Хотя это и решает все проблемы, зафиксированные в разделе неразрешённых вопросов, такой подход крайне негативно сказывается на interoperability, простоте развёртывания и использования системы, что может привести к нецелесообразности внедрения ClickHouse в существующие процессы потенциальных клиентов – крупных компаний, нуждающихся в эффективной аналитике собираемой информации.

5.3. Техническое решение №3: SQL

5.3.1. Проблема

Система должна принимать запросы на понятном для целевых пользователей языке, не требующем от них специальной квалификации и прохождения обучения.

5.3.2. Идея решения

Система должна принимать запросы конечных пользователей в виде текста, описывающего выполняемую операцию на языке SQL.

5.3.3. Факторы

1. Для преобразования полученных текстовых запросов в вид, пригодный для исполнения в ClickHouse, потребуется разработка дополнительных алгоритмов синтаксического разбора. При этом вклад затрат на предобработку и разбор запросов в общее время их выполнения должен быть статистически незначим.
2. Для обеспечения возможности оптимизации планов выполнения запросов требуется, чтобы используемый язык был декларативный (SQL, MDX) или хотя бы векторный (J, K, APL), то есть не содержал циклы в явном виде.

5.3.4. Решение

Для обработки запросов реализован набор разборщиков, следующих общему интерфейсу `IParser`. Каждый разборщик представлен отдельным классом в пакете `Parsers` и отвечает за выявление и анализ конкретных типов выражений, а также их запись в абстрактное синтаксическое дерево, структура которого описана в п. 4.7.2 настоящего документа. При этом существуют как “небольшие” разборщики, определяющие отдельные команды или операции, так и разборщики, отвечающие за построение синтаксического дерева по всему тексту запроса. При этом “большие” разборщики могут рекурсивно “пользоваться” реализациями “меньших” разборщиков.

5.3.5. Мотивировка

Выбранное решение позволяет инкапсулировать всю логику синтаксического разбора запроса в одном пакете, а также предполагает разработку удобной для передачи по цепочке ответственности структуры, которая эффективно содержит всю информацию о полученном запросе.

Наличие общего интерфейса всех разборщиков позволяет без труда добавлять поддержку новых синтаксических конструкций и команд, не изменяя общую архитектуру системы. А независимый от пользовательского ввода формат представления запроса – абстрактное синтаксическое дерево – позволяет, если это будет целесообразно в будущем, реализовать

поддержку и других языков запросов. При этом потребуется внести изменения лишь в один пакет системы, не затрагивая другие этапы обработки запроса и архитектуру в целом.

5.3.6. Неразрешенные вопросы

При разработке необходимо уделить особенное внимание обеспечению высокой производительности алгоритмов синтаксического разбора запросов. Хотя спроектированная архитектура и использование абстрактного синтаксического дерева должны обеспечивать достаточную эффективность процесса, следует тщательно следить за отсутствием излишних операций, значительно повышающих время обработки запроса.

Более того, поддержка всех возможностей SQL невозможна и нецелесообразна в разрабатываемой системе, что требует дополнительной проработки механизмов сообщения пользователям об отсутствии поддержки той или иной функции для обеспечения простоты использования и наблюдаемости системы.

5.3.7. Альтернативы

В качестве альтернативных решений могут быть рассмотрены другие языки запросов – MDX, J, K, APL и др. Тем не менее эти языки гораздо менее распространены и их использование отрицательно бы повлияло на простоту работы с ClickHouse. Более того, важно, чтобы используемый язык запросов не содержал циклов в явном виде, открывая широкие возможности для оптимизации планов выполнения.

5.4. Техническое решение №4 (Доработка №1): применение традиционных шаблонов проектирования

5.4.1. Проблема

Расширение системы (добавление источников данных, алгоритмов обработки столбцов, протоколов взаимодействия и др.) не должно требовать изменения существующих файлов с исходным кодом, кроме, возможно, указания реализованного алгоритма в соответствующем управляющем классе.

5.4.2. Идея решения

Использование при проектировании традиционных, проверенных временем шаблонов, обеспечивающих единственность ответственности каждого компонента реализуемой системы.

5.4.3. Факторы

1. Типичные модификации системы (добавление протоколов внешнего взаимодействия, способов получения данных, алгоритмов их обработки и т.д.) должны быть возможны без изменения её общего архитектурного решения.

2. Внедрение новых функций не должно нарушать целостность системы и подвергать риску работоспособность существующих компонентов. Нарушение обратной совместимости при интеграции новых возможностей должно производиться только при крайней необходимости.
3. Должна быть возможность организации распределённой разработки независимых компонентов и настройки эффективной системы контроля версий.

5.4.4. Решение

Архитектурное решение основано на применении большого количества традиционных шаблонов проектирования, таких как “цепочка ответственности”, “интерпретатор” и “строитель” для построения дерева выполнения запроса и организации процесса его обработки; “фабрика” и “объектный пул”, отвечающие за создание и кеширование конкретных реализаций протоколов взаимодействия со сторонними СУБД, а также алгоритмов синтаксического разбора, интерпретации и выполнения запросов; “единая точка входа” и “фасад” для обеспечения унифицированных интерфейсов взаимодействия с отдельными пакетами, включая `Service`, и системой в целом.

5.4.5. Мотивировка

Описанное решение позволяет реализовать эффективную, понятную архитектуру с большим потенциалом для расширения: добавления новых возможностей, внедрения независимых компонентов и интеграции сторонних технологий в систему без значительных изменений существующего исходного кода, что повышает его качество, снижает связность и обеспечивает инкапсуляцию конкретных реализаций за общими интерфейсами. Более того, использование традиционных шаблонов способствует следованию определённым принципам проектирования на всех этапах жизненного цикла проекта, что снижает риск принятия ошибочных решений, негативно влияющих на архитектуру системы, и упрощает разработку набора тестовых сценариев, наиболее полно проверяющего все возможности системы, что особенно важно для проекта с открытым исходным кодом.

5.4.6. Неразрешенные вопросы

При проектировании системы на основе традиционных шаблонов следует уделять особенное внимание обоснованию необходимости и целесообразности их применения в каждом конкретном случае во избежание чрезмерного усложнения и запутывания исходного кода из-за использования сложных шаблонов для выполнения задач, имеющих более простое решение. Более того, некоторые традиционные шаблоны проектирования накладывают специальные ограничения на реализацию и дальнейшую расширяемость системы, что может стать узким местом архитектурного решения и потребовать значительных дорогостоящих её модификаций для внедрения новых функциональных возможностей.

5.4.7. Альтернативы

Альтернативным решением можно считать проектирование системы без особенного внимания к внедрению традиционных шаблонов. Тем не менее такой подход, особенно для открытого проекта, неизбежно приведёт к усложнению и запутыванию исходного кода, бесконтрольному росту связности компонентов и “исчезновению” людей, глубоко разбирающихся в архитектуре системы в целом.

5.4.8. Описание доработки

Реализация нового протокола внешнего взаимодействия с системой требует модификации существующих компонентов: класса Server. Пример такого изменения: <https://github.com/ClickHouse/ClickHouse/pull/15111>. Это не соответствует требованиям к простоте внедрения новых функций и нарушает архитектуру системы, основанную на широком применении традиционных шаблонов проектирования и принципов объектно-ориентированного программирования с инкапсуляцией специфической логики за общими интерфейсами.

В рамках доработки требуется инкапсулировать все функции, реализующие протоколы внешнего взаимодействия с системой и непосредственно запускающие серверы на заданных в файле конфигурации портах, за единым интерфейсом в пакете Server.

Это позволит добавлять новые протоколы взаимодействия без модификации существующих компонентов, что повышает качество исходного кода, снижая его связность и восстанавливая инкапсуляцию конкретных реализаций за общими интерфейсами.

5.4.9. Исходный код доработки

<https://github.com/ClickHouse/ClickHouse/pull/64132>

5.4.10. Оценка результативности доработки

1. Изменения обратно совместимы, все существующие тестовые сценарии успешно пройдены за исключением отдельных “непостоянных” (flaky) тестов. Ограничений функциональных возможностей системы в связи с изменениями не выявлено. Исходный код изменений одобрен сотрудником ClickHouse и слит с основной веткой.
2. Для добавления нового протокола внешнего взаимодействия с системой необходимы изменения исключительно в пакете Server. Модификация файла Server.cpp директории programs не требуется.
3. Программный интерфейс существующих компонентов пакета Server не изменён. Инкапсуляция существующих методов, реализующих протоколы внешнего взаимодействия с системой, в пакете Server не нарушена.
4. Размер файла Server.cpp пакета programs уменьшен чуть менее, чем на 1000 строк – более 35%.

5.5. Техническое решение №5 (Доработка №4): широкое использование принципов ООП

5.5.1. Проблема

Компоненты, представляющие конкретные реализации одного элемента (алгоритма получения запроса, его синтаксического разбора, представления в памяти, интерпретации, выполнения и др.), должны иметь общий программный интерфейс, обеспечивающий возможность реализации единой точки их создания и обработки, а также позволяющий интегрировать новые реализации без изменения общего архитектурного решения.

5.5.2. Идея решения

Применение принципов объектно-ориентированного программирования для снижения связности компонентов системы, что способствует повышению её расширяемости и модифицируемости.

5.5.3. Факторы

1. Внешнее взаимодействие с системой должно быть возможно по протоколам TCP, HTTP и GRPC, а реализация поддержки других протоколов должна быть возможна без изменения общего архитектурного решения.
2. Система должна иметь возможность обработки информации, хранящейся в сторонних хранилищах PostgreSQL, S3, Redis и др., а интеграция с дополнительными источниками, включая внедрение внутреннего хранилища, должна производиться в изоляции от существующих компонентов.
3. Добавление источников данных, алгоритмов обработки столбцов, протоколов взаимодействия с системой и др. не должно требовать изменения существующих файлов с исходным кодом, кроме, возможно, добавления реализованного алгоритма в соответствующий управляющий компонент.

5.5.4. Решение

Абстракция конкретных реализаций ключевых компонентов (алгоритмов синтаксического разбора, элементов синтаксического дерева и плана выполнения запроса, интерпретаторов, хранилищ данных, методов обработки информации и др.) за едиными интерфейсами IParser, IAST и IQueryPlanStep, IInterpreter, IDatabase и IStorage, IParser, а также инкапсуляция их состояния и деталей реализации с помощью непубличных атрибутов и методов для обеспечения изоляции от внешних компонентов; применение наследования для переиспользования функциональных возможностей базовых классов Port, BufferBase и др.; полиморфное хранение конкретных реализаций интерфейсов в экземплярах IServersManager, QueryPipeline, QueryPlan, DatabaseCatalog и др.

5.5.5. Мотивировка

Изложенный подход скрывает состояние и поведение конкретных реализаций компонентов системы за простыми интерфейсами, что упрощает как поддержку существующих функций, позволяя эффективно управлять зависимостями между различными частями исходного кода, так и внедрение новых возможностей, обеспечивая переиспользование готовых алгоритмов и адаптацию системы к новым требованиям. Более того, следование набору строгих правил при проектировании позволяет обеспечить устойчивость общего архитектурного решения, снижая влияние локальных ошибок на систему в целом и упрощая её разработку, что особенно важно для проекта с открытым исходным кодом.

5.5.6. Неразрешенные вопросы

Строгое следование принципам объектно-ориентированного программирования может привести к появлению излишних компонентов (например, интерфейсов и абстрактных классов только с одной реализацией), не упрощающих интеграцию новых возможностей, а лишь усложняющих и запутывающих исходный код системы. При проектировании архитектуры необходимо не допускать появление таких абстракций, не имеющих ценности для расширяемости системы.

Более того, чрезмерный рост уровня абстракции компонентов может привести к значительному увеличению времени сборки исходного кода, усложнению его оптимизации компилятором и, как следствие, снижению общей производительности системы.

5.5.7. Альтернативы

Компоненты, к производительности которых предъявляются особенно строгие требования, могут быть реализованы в функциональном стиле, позволяющем организовать параллельную, многопоточную обработку данных без необходимости специальной синхронизации их состояния. Тем не менее использование такого подхода приведёт к усложнению архитектуры и потребует большей квалификации специалистов, разрабатывающих систему, что усложнит ее поддержку, особенно как проекта с открытым исходным кодом. Более того, отдельные алгоритмы могут быть не адаптированы для реализации в функциональном стиле, что приведёт к необходимости разработки дополнительных механизмов, еще больше усложняющих систему.

5.5.8. Описание доработки

При реализации протокола внешнего взаимодействия, не основанного на уже поддержанных в системе (TCP и GRPC), необходимы доработки компонента `ProtocolServerAdapter`, обеспечивающего возможность хранения списка всех запущенных серверов в одном массиве. Пример такого изменения: <https://github.com/ClickHouse/ClickHouse/pull/15111>. Фактически, `ProtocolServerAdapter` является аналогом типа `Any` для классов пакета `Server`, что нарушает принципы проектирования системы, делает неявным общий интерфейс классов, реализующих сервера получения внешних запросов, и усложняет внедрение поддержки новых протоколов взаимодействия.

Для обеспечения соответствия реализации архитектуры принятым решениям необходимо разработать единый интерфейс протоколов внешнего взаимодействия с системой, содержащий общие методы существующих классов, и использовать его для реализации хранилища всех запущенных серверов.

Это позволит прекратить использование нарушающего принципы объектно-ориентированного программирования класса ProtocolServerAdapter, что способствует упрощению архитектуры системы, а также повышению качества исходного кода и простоты его модификации, избавляя разработчиков от необходимости создания дополнительного компонента при реализации поддержки нового протокола внешнего взаимодействия.

5.5.9. Исходный код доработки

<https://github.com/ClickHouse/ClickHouse/pull/64413>

5.5.10. Оценка результативности доработки

1. Изменения обратно совместимы, все существующие тестовые сценарии успешно пройдены за исключением отдельных “непостоянных” (flaky) тестов. Ограничений функциональных возможностей системы в связи с изменениями не выявлено. Исходный код изменений готов к рассмотрению сотрудником ClickHouse и слиянию с основной веткой.
2. В результате изменений для реализации нового типа сервера внешнего взаимодействия с системой не требуются модификации существующих компонентов исходного кода, за исключением добавления разработанных алгоритмов в классы пакета ServersManager.
3. Программный интерфейс существующих компонентов пакета Server не изменён. Инкапсуляция существующих методов, реализующих протоколы внешнего взаимодействия с системой, в пакете Server не нарушена.
4. Класс ProtocolServerAdapter, являющийся необоснованным использованием шаблона проектирования “Адаптер”, больше не используется и удален из исходного кода системы.
5. Классы, реализующие протоколы внешнего взаимодействия с системой, следуют единому интерфейсу, явно описанному классом IProtocolServer, все методы которого снабжены необходимыми и достаточными для их понимания комментариями.

6. Приложения

6.1. Словарь терминов

БД – (сокр.) база данных

Вертикальное масштабирование – увеличение ресурсов, доступных каждому компоненту системы, с целью повышения производительности

Горизонтальное масштабирование – масштабирование системы путём разбиения её на структурные компоненты и/или запуск на нескольких физических или логических узлах

Параметры конфигурации – настройки, задаваемые в отдельном файле формата XML, JSON, YAML или др. и обрабатываемые системой во время запуска и выполнения для динамической корректировки поведения

Пользовательские данные – данные, содержащиеся в системе исключительно вследствие действий пользователя

Пропускная способность – метрическая характеристика, показывающая максимальное допустимое количество информации, обрабатываемое в единицу времени

Системная таблица – таблица, содержащая данные, сохранённые и обрабатываемые системой для обеспечения её работоспособности

СУБД – (сокр.) система управления базами данных

Транзакция – группа логически объединённых последовательных операций по работе с данными, выполнение которой возможно исключительно целиком

6.2. Задействованные диаграммы

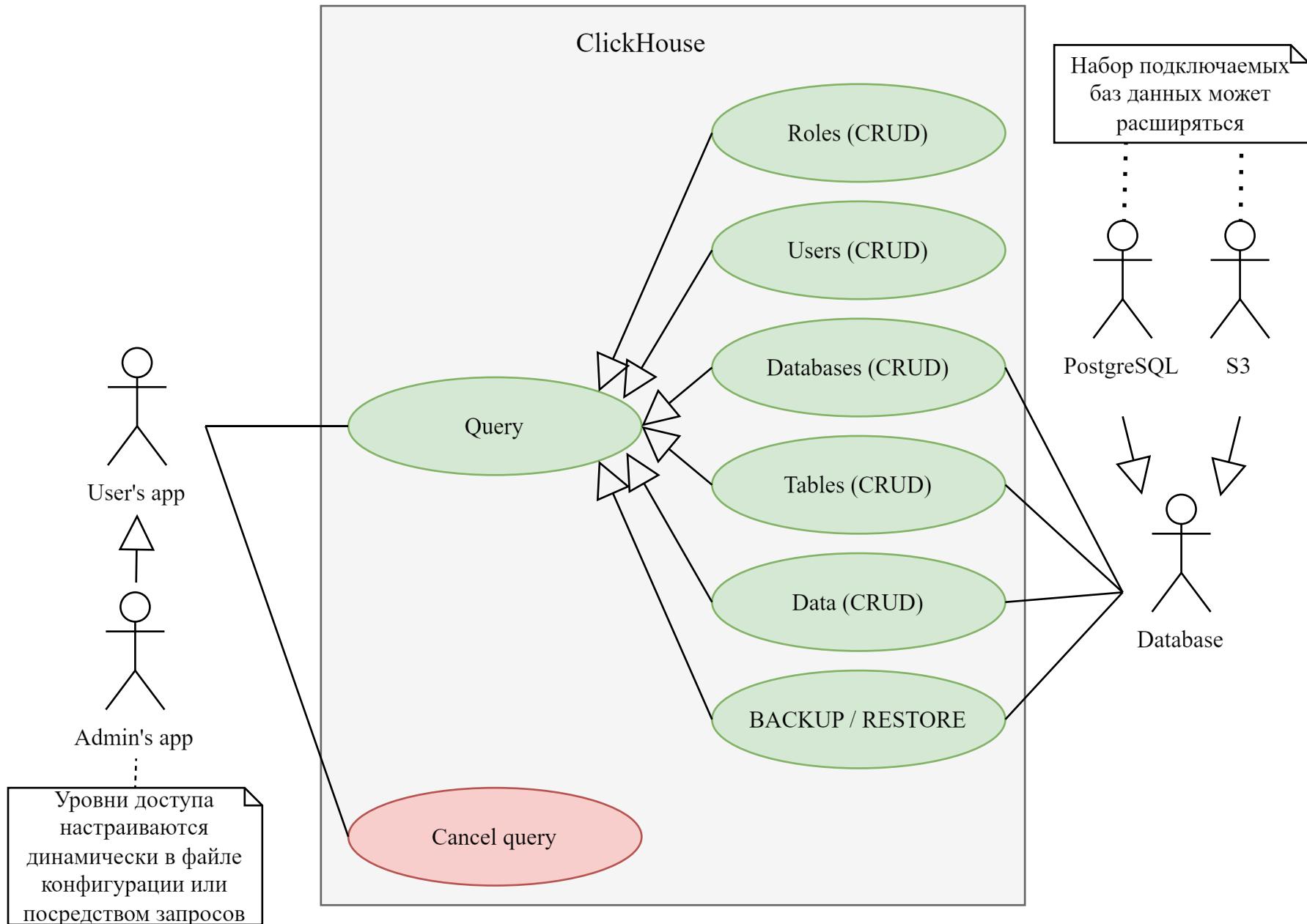


Рисунок 1 - Модель прецедентов




Рисунок 2 - Проектная диаграмма классов




Рисунок 3 - Проектная диаграмма пакетов

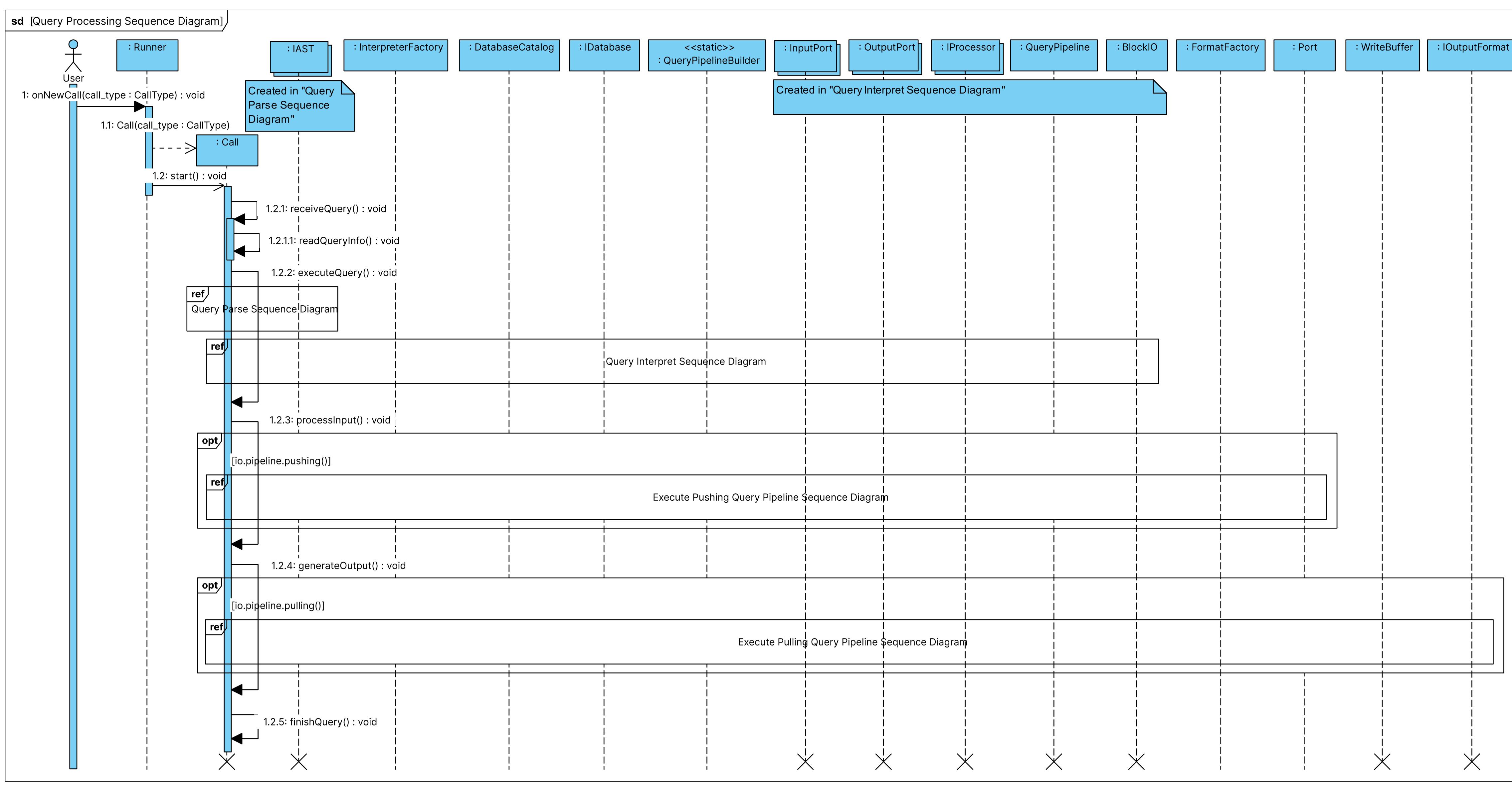


Рисунок 4 - Общая диаграмма последовательности обработки запроса

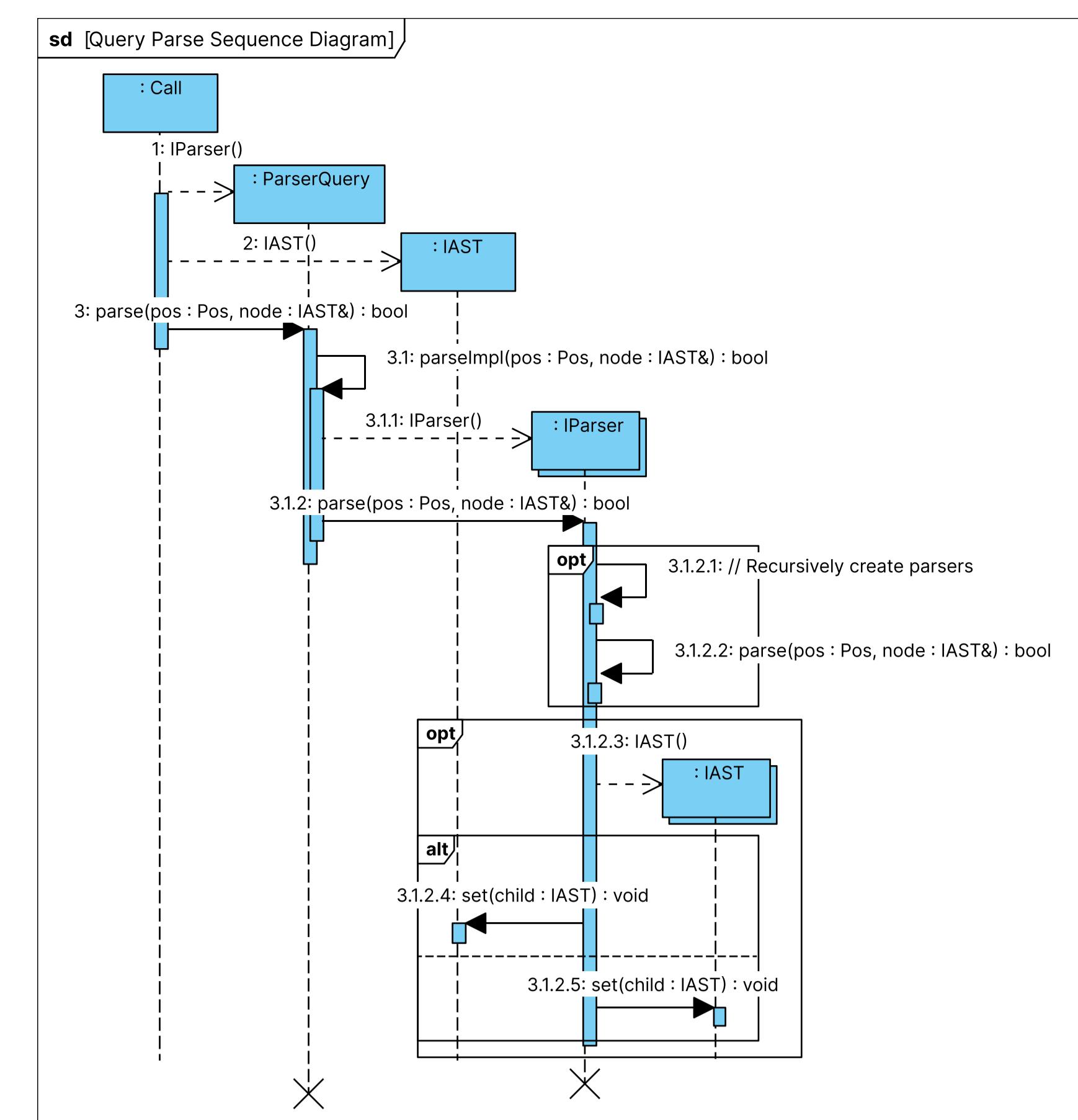


Рисунок 5 - Диаграмма последовательности синтаксического разбора запроса

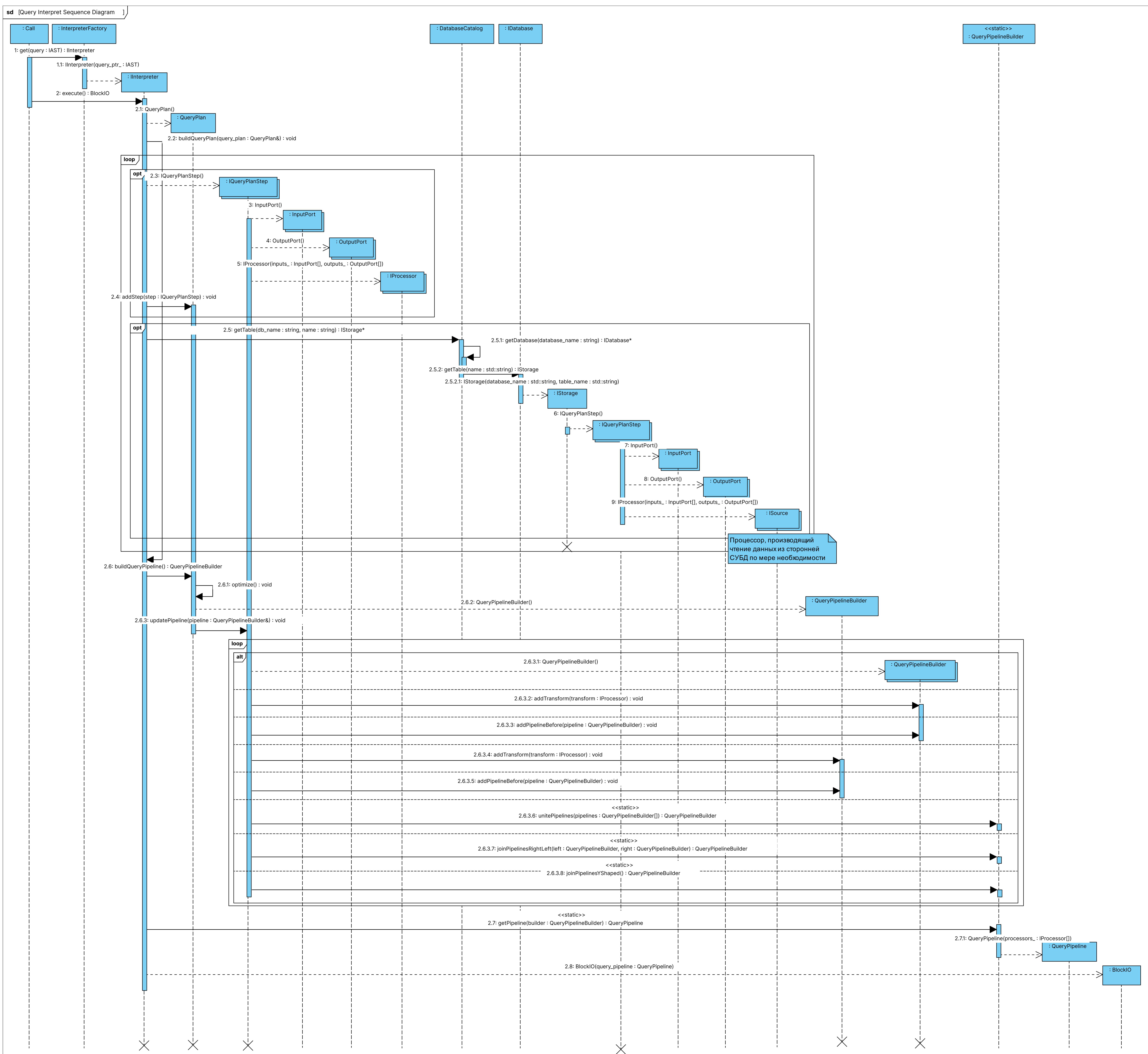


Рисунок 6 - Диаграмма последовательности построения плана выполнения запроса

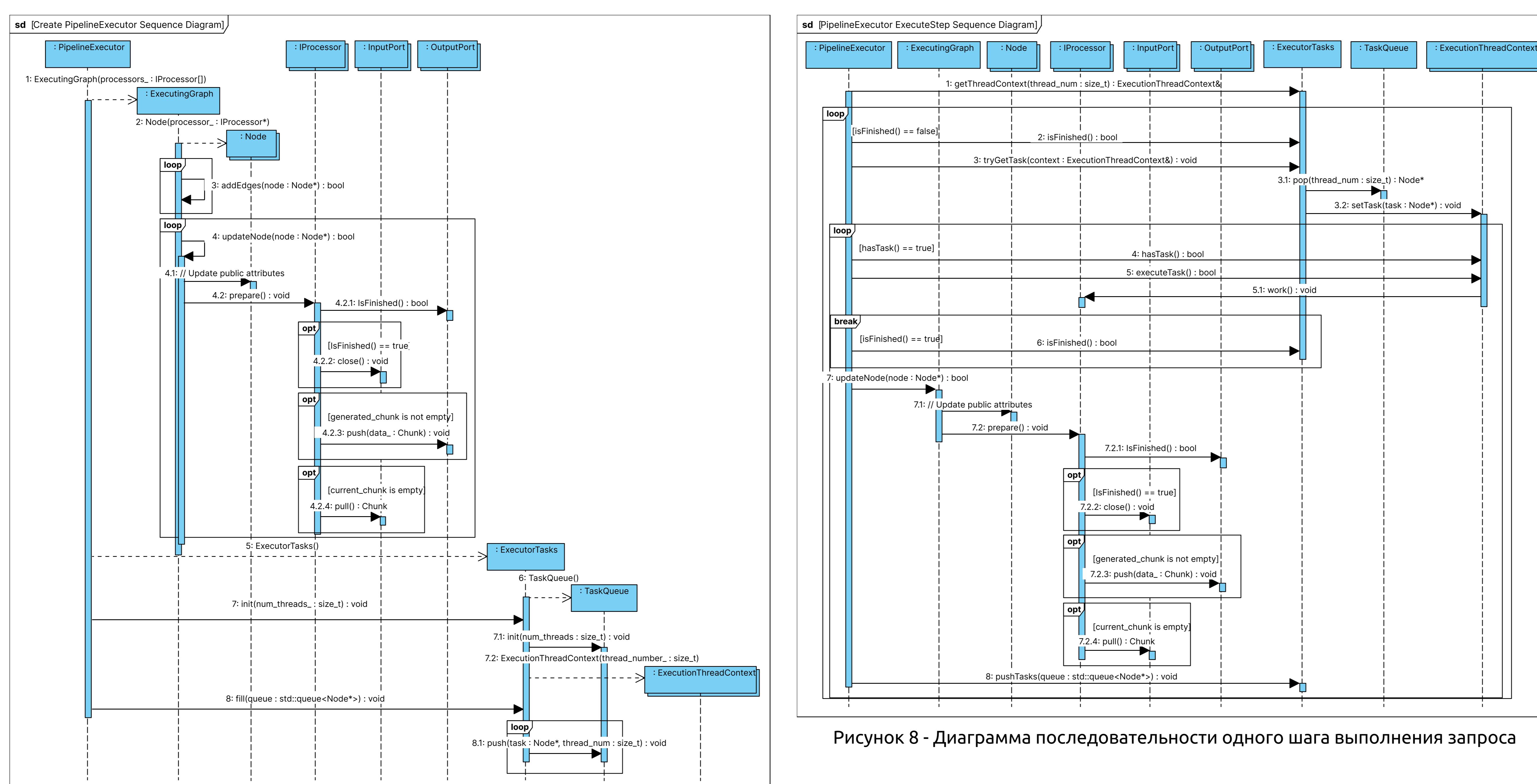


Рисунок 8 - Диаграмма последовательности одного шага выполнения запроса

Рисунок 7 - Диаграмма последовательности создания экземпляра класса PipelineExecutor

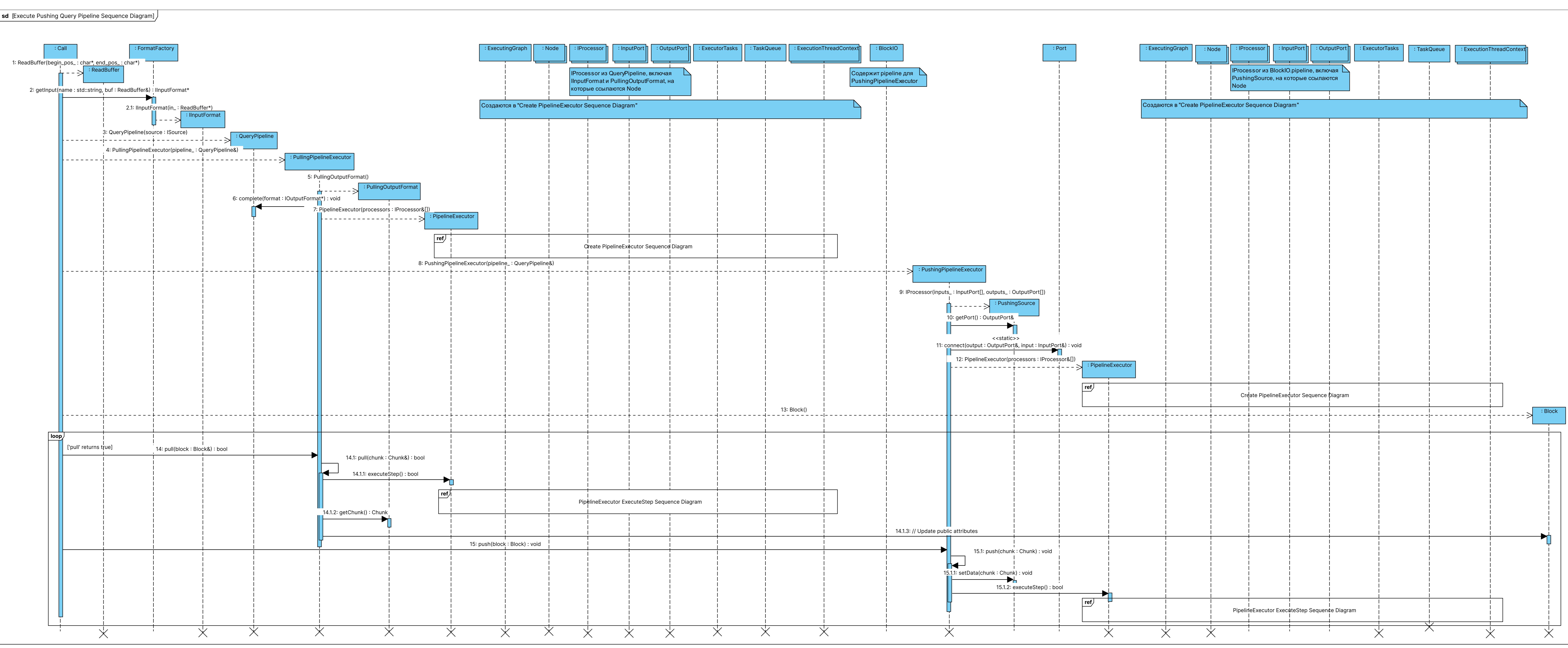


Рисунок 9 - Диаграмма последовательности выполнения запроса на запись

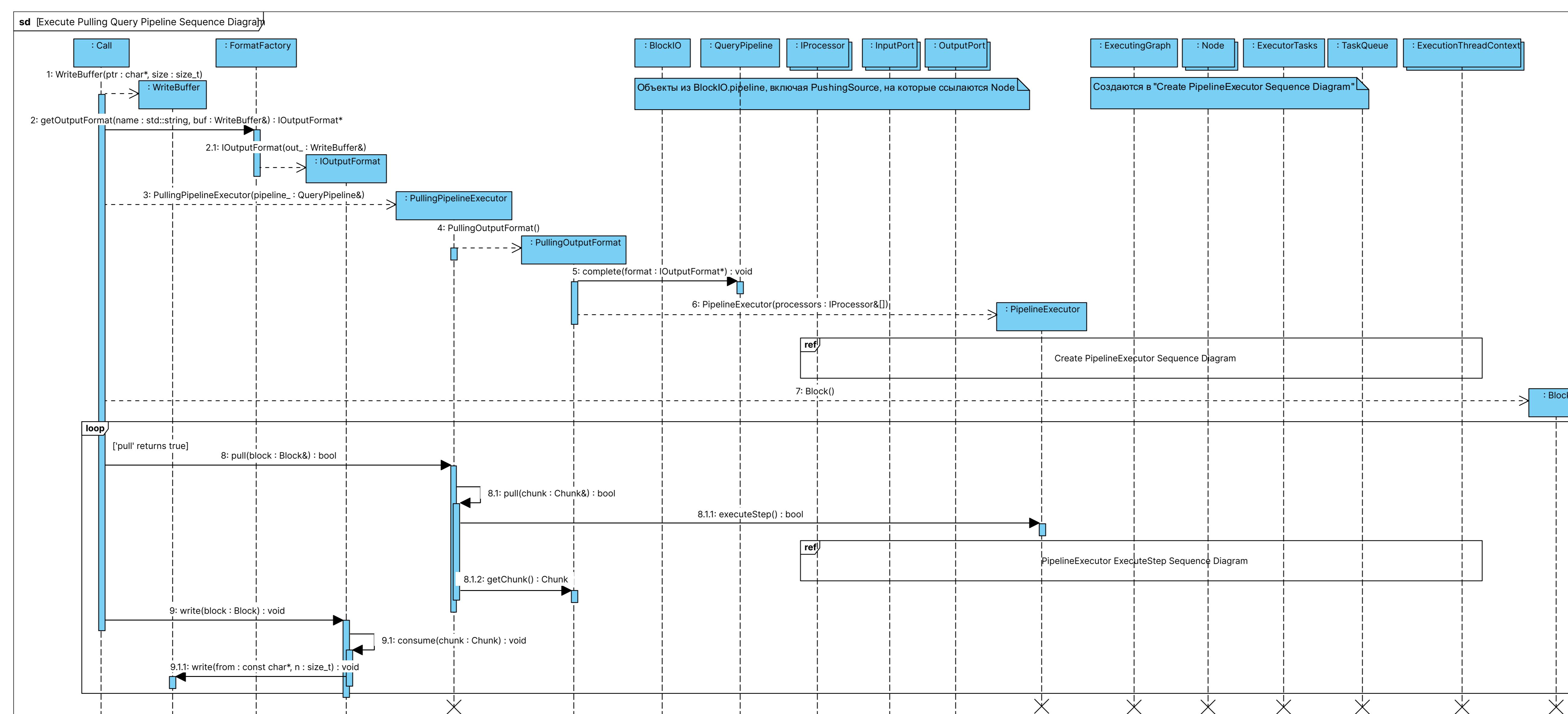


Рисунок 10 - Диаграмма последовательности выполнения запроса на чтение