Санкт-Петербургский государственный университет

Гордиенко Егор Александрович

Выпускная квалификационная работа

Реализация сервиса для генерации отчетности по корпоративному обучению сотрудников

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2019 «Программная инженерия»*

Научный руководитель:
к.ф.-м.н., доцент Д.В. Луцив

Рецензент:
ведущий разработчик .NET ООО «Актион-диджитал» И.В. Бабаков

Санкт-Петербург
2023

Saint Petersburg State University

Egor Gordienko

Bachelor's Thesis

Implementation of a Service for Generating Reports on Corporate Employee Training

Education level: bachelor

Speciality *09.03.04 «Software Engineering»*

Educational Programme *CB.5080.2019 «Software Engineering»*

Scientific supervisor:
C.Sc., docent D.V. Luciv

Reviewer:
Lead .NET Developer, LLC «Action-Digital» I.V. Babakov

Saint Petersburg
2023

Оглавление

Введение **5**

1. Постановка задачи **7**

2. Обзор существующего решения и технологий **8**

2.1. Существующая система внутри платформы 8

2.1.1. Функциональность 8

2.1.2. Достоинства и недостатки 9

2.2. OLTP и OLAP подходы 11

2.3. CDC и реплицирование 12

2.4. Вывод 13

2.5. Использованные технологии 14

2.5.1. Greenplum 14

2.5.2. Debezium 14

2.5.3. Airflow 15

2.5.4. Golang 15

3. Требования **16**

4. Архитектура **17**

4.1. Взаимодействие модулей системы 17

4.2. Сценарии использования 18

4.3. Сервис генерации отчетов 20

5. Особенности реализации **22**

5.1. Применение CDC-событий и реплицирование 22

5.2. Формирование агрегированных таблиц для отчетов . . . 22

5.3. Оркестрация процессов в Airflow 23

5.4. Поддержка HTTP API и асинхронных запросов в Kafka 24

5.5. Развертывание сервиса и мониторинг 24

6. Тестирование и апробация **26**

Заключение **27**

Список литературы **29**

Введение

Необходимым атрибутом любого развивающегося бизнеса является качественное обучение и повышение квалификаций сотрудников компании. В наше время эффективным способом обучать персонал с минимальными затратами является дистанционный формат и специальные системы, которые позволяют получать знания удаленно. Организация обучения на таких платформах предполагает использование готовых материалов и уроков с адаптацией под конкретную компанию при необходимости, передачу доступов сотрудникам, выставление сроков прохождения, проверку усвоенных знаний. Это позволяет специалистам различных сфер (например, бухгалтерии, права, кадров, финансов, госзакупок) получать доступ к корпоративной базе знаний, соответствующей профстандартам, закреплять знания через тесты и проходить аттестацию, не отрываясь от рабочего процесса.

При работе с системами дистанционного обучения руководителю компании предоставляется особый доступ, на основе которого формируется учебное заведение. Руководитель может приглашать на обучение сотрудников, назначать им программы и тесты, следить за прогрессом и успеваемостью учеников. Возможность просматривать статистику по обучению является важной составляющей функциональности платформы, поскольку от ее показателей зависят дальнейшие действия руководителя – например, в случае неудовлетворительных результатов сотрудников стоит мотивировать назначение нового обучения. При этом для обеспечения удобного и надежного пользовательского опыта данные должны корректно вычисляться и отображаться на странице за приемлемое время.

У одной организации в сфере профессиональных СМИ, разрабатывающей платформу для онлайн-обучения сотрудников, возникла необходимость в создании страницы с отображением отчетности для руководителей. Планируемый раздел должен как показывать общую статистику по обучению всех сотрудников (их активность, вовлеченность, успеваемость), так и формировать детальные отчеты с описанием прой-

денных программ и тестов по каждому ученику. Реализация этой функциональности также осложняется большим объемом данных: в рамках одного учебного заведения может быть более 10000 пользователей и десятки тысяч назначений обучений и прохождений программ. Важно своевременно обрабатывать информацию, при этом не затрачивая много времени на загрузку раздела.

Данная работа посвящена созданию сервиса, предоставляющего необходимую информацию для отображения на странице со статистикой по обучению пользователей и генерирующего подробные отчеты, с учетом описанных выше сложностей.

1 Постановка задачи

Целью данной работы является реализация сервиса для генерации отчетности по корпоративному обучению сотрудников для платформы онлайн-образования. Для достижения этой цели были сформулированы следующие задачи.

- Произвести обзор существующего решения, подходов и используемых технологий.
- Собрать требования к сервису.
- Спроектировать архитектуру сервиса.
- Выполнить реализацию сервиса:
  - интегрировать базу данных платформы онлайн-обучения;
  - реализовать обработку и подготовку данных для отчетов;
  - реализовать сервис, предоставляющий платформе рассчитанные данные.
- Провести тестирование и апробацию созданного сервиса.

2 Обзор существующего решения и технологий

Сама по себе идея отображения статистики по обучению пользователей не нова и уже была реализована в рамках платформы онлайн-образования, о которой идет речь. Однако в связи с масштабированием продукта и увеличением объема пользователей выбранный ранеее подход обрел ряд недостатков, из-за которых существующее решение перестало удовлетворять требованиям бизнеса. Далее приведен обзор этого решения, а также возможные подходы и технологии для решения возникших проблем.

2.1 Существующая система внутри платформы

2.1.1 Функциональность

Раздел "Отчеты" представляет из себя несколько страниц с информацией об обучении сотрудников.

На странице со статистикой по обучению выводятся общие показатели сотрудников:

- Вовлеченность сотрудников – сколько всего человек учатся, сколько из них выполняют план, сколько отстают или вообще не обучались. Здесь же можно отправить приглашение на присоединение к учебному заведению.

- Активность в программах и тестах. В этом разделе находится инфографика, отображающая количество тестов и программ, которые проходили пользователи. Если кто-то не успевает в сроки, можно поторопить сотрудников, отправив им письмо.

- Успеваемость. Здесь показаны как лучшие, так и отстающие сотрудники, а также их средний балл, вместе со средним показателем по всей компании и сравнением с результатами по России.

Во вкладке ”Детализация по сотрудникам” (Рис.1) расположена подробная информация, сгруппированная по ученикам. По каждому сотруднику можно увидеть данные о его прохождении программ и тестов, оценки по темам, медали и средние показатели. Кроме того, можно вывести и сравнить показатели, сгруппированные по каждому курсу.

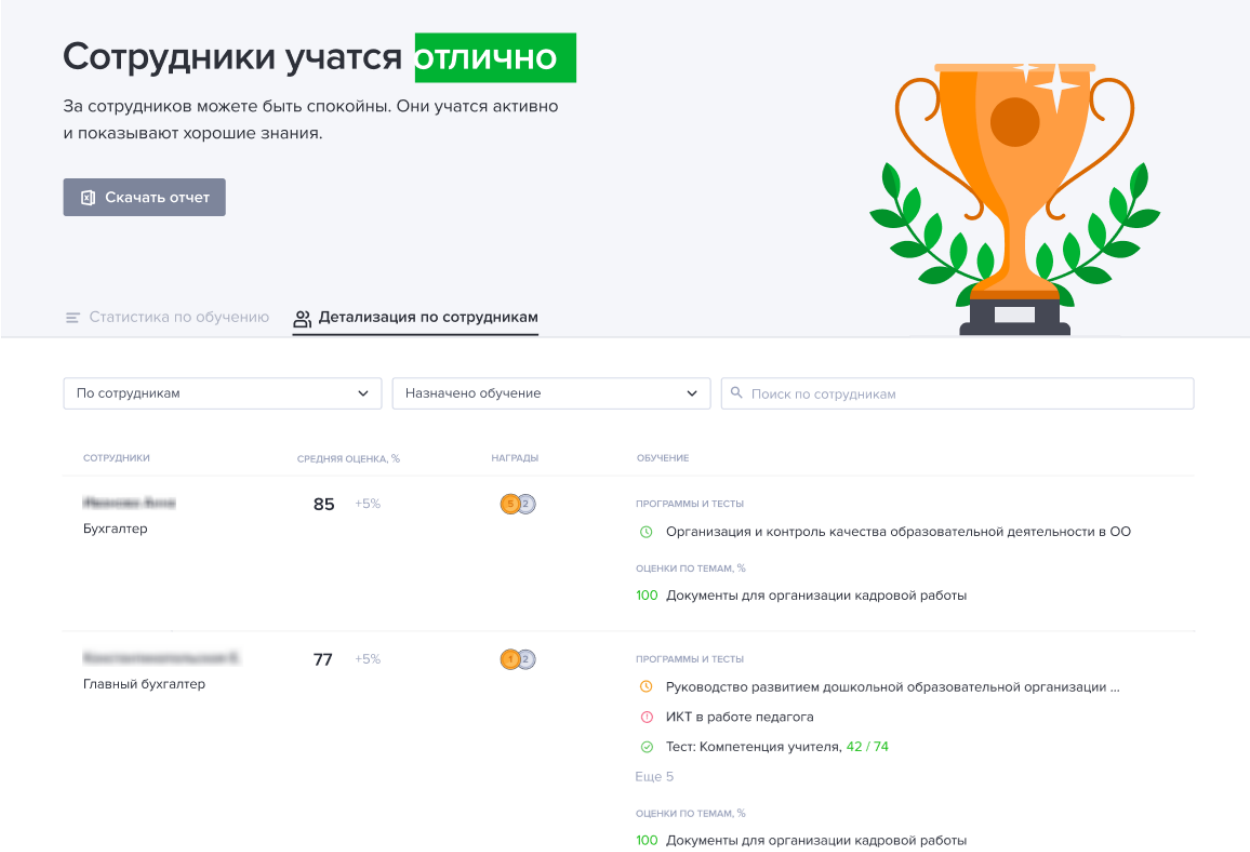

Рис. 1: Страница детализации по сотрудникам

Для получения более полной информации присутствует возможность сгенерировать и скачать два типа отчетов в формате .xls. Первый – отчет по программам и тестам, отражающий данные о сроках и результатах обучения сотрудников. Второй – по ответам на вопросы, включающий сведения по срокам и результатам обучения с ответами в каждом тесте.

2.1.2 Достоинства и недостатки

Вся платформа онлайн-обучения реализована на стеке .NET с применением микросервисной архитектуры. Раздел отчетов также приме-

няет этот подход – как для frontend, так и для backend части. Микросервисы backend-части написаны на языке C# 4 версии .NET Framework и применяют стиль REST [26] с использованием ASP.NET Web API [2]. Сервисы работают в контейнерах кластера Kubernetes и обращаются к данным в таблицах базы MS SQL Server. Для отображения информации на странице основной сервис формирует запросы в базу данных. Функциональность генерации .xls файлов с подробными отчетами реализована в отдельном приложении, куда передаются данные из базового сервиса.

Существующее решение обладает рядом достоинств, присущих классической трехуровновой архитектуре, среди них – масштабируемость компонентов, их изолированность и гибкая конфигурируемость. Однако выросшая со временем нагрузка показала ряд проблем в этом решении. Среди них:

- Из-за роста объема данных в БД время запроса статистики стало увеличиваться, вплоть до таймаутов на стороне БД. Оптимизация запросов давала небольшой выигрыш, но с увеличением числа пользователей, программи и тестов это помогло лишь на определенный промежуток времени.

- Генерация excel-отчетов также занимала долгое время. Помимо этого, тратилось большое количество ресурсов контейнеров, что создавало неоправданно высокую нагрузку на кластер со стороны микросервиса.

- В худшем случае, после длительного ожидания сервис начинал выдавать ошибки с 500 кодом и переставать отвечать.

В пиковые точки нагрузки (Рис. 2) пользователь получал негативный опыт использования сервиса, не имея возможности даже открыть страницу с отчетами.

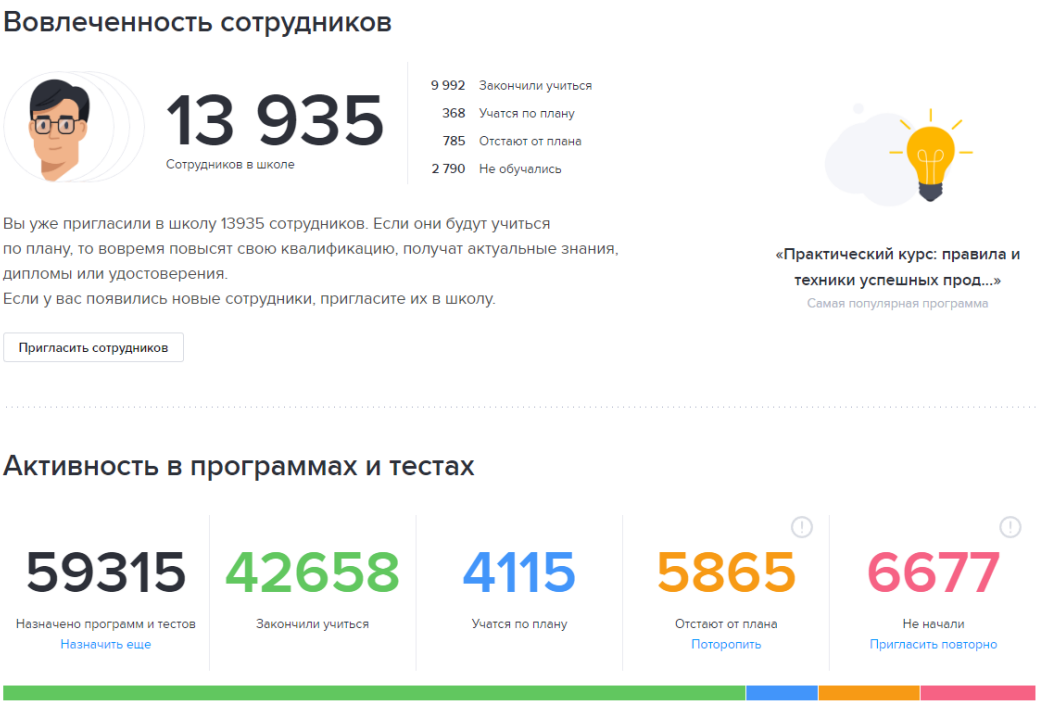

Рис. 2: Показатели нагрузки на сервис, при котором страница не открывалась

2.2 OLTP и OLAP подходы

Рассмотрим два основных подхода к построению хранилищ данных – On-Line Transaction Processing (OLTP) [23] и On-Line Analytical Processing (OLAP) [22].

OLTP базы данных – это системы, которые обрабатывают транзакции с данными. В этом контексте обработка транзакций означает, что клиенты могут выполнять операции чтения и записи с низкой задержкой. Многие из таких баз поддерживают свойства ACID [1] и хорошо подходят для поиска небольшого количества записей по некоторому ключу, используя индекс. Обычно ожидается, что эти системы будут высокодоступными, поскольку они часто имеют решающее значение для функционирования бизнеса. К числу OLTP можно отнести большинство реляционных БД, включая упомянутый ранее MS SQL Server.

OLAP системы, также называемые хранилищами данных, предназначаются для другого типа нагрузок – аналитических запросов, сканирующих большое количество записей, считывающих только несколько

столбцов на запись, и вычисляющих некоторую совокупную статистику. В отличии от OLTP, хранилища данных обычно предоставляют не только текущее состояние, но и исторические данные, а также способны хранить и обрабатывать терабайты и петабайты информации. В качестве примеров таких систем можно привести Greenplum [16], Clickhouse [7], Vertica [29].

Оба типа систем имеют SQL-совместимый интерфейс, однако внутренние компоненты могут выглядеть совершенно по-разному, и сосредоточены на поддержке либо обработки транзакций, либо аналитических нагрузок, но не обоих сразу.

В частности, в большинстве баз данных OLTP хранилище расположено в виде строк: все значения из одной строки таблицы хранятся рядом друг с другом. Аналитические хранилища же используют хранение, ориентированное на столбцы, дающее возможность применять эффективные алгоритмы сжатия и хранить агрегированную статистику по столбцам. Также OLAP базы зачастую используются в виде кластера из нескольких машин, что позволяет распределять нагрузку и масштабировать систему [19].

2.3 CDC и реплицирование

При использовании OLAP подхода хранилище представляет собой отдельную БД, содержащую доступную только для чтения копию данных из OLTP систем. Извлечение информации возможно двумя путями – либо периодическим копированием полного состояния (репликацией), либо получением непрерывного потока обновлений. Последний подход называется CDC (Change Data Capture). По сравнению с репликацией, он обладает рядом преимуществ:

- Эффективность – при использовании CDC для нового состояния учитываются только данные, которые изменились с момента последней репликации. Это позволяет избежать копирования всех объемной таблицы.

- Обработка почти в режиме реального времени. Некоторые реализации CDC, например, на основе журналов, обновляют журнал транзакций по мере внесения изменений в базу данных, чего невозможно достись при полной репликации.

- Отслеживание истории, в том числе операций удаления в исходной базе данных.

- Уменьшение воздействия на исходную базу данных. Многие OLTP БД, например SQL Server, изначально ведут журнал транзакций как часть своей основной функциональности. Это означает, что процессу CDC не требуется перегружать исходные БД, используемые в бизнес-приложениях.

Основными способами реализации CDC являются хранение колонок метаданных в таблице, вычисление разницы между основной и реплицируемой таблицей, установка триггеров, ведение журнала транзакций. По сравнению с остальными, последний метод позволяет избежать нагрузки на базовую таблицу, не применяя логических изменений к объектам и не реализуя дополнительной пользовательской логики. Из минусов следует отметить зависимость от доступности и надежности системы обработки журнала [28].

2.4 Вывод

Рассмотренное ранее существующее решение перестало удовлетворять бизнес по нескольким причинам. Среди них – большие затраты по ресурсам, долгое ожидание пользователей, таймауты, деградация сервисов, неоправданно высокая нагузка. Поэтому, исходя из упомянутых преимуществ OLAP подхода, выгодных для агрегирующих типов запросов, было принято решение перенести функциональность отчетов на сторону вычислительных мощностей кластера хранилища данных. Для уменьшения нагрузки на основную транзакционную базу данных был выбран подход CDC с использованием журнала транзакций. Таким образом, возникла потребность спроектировать и реализовать сервис,

использующий перечисленные практики и отдающий данные на сторону бэкенда онлайн-платформы.

2.5 Использованные технологии

2.5.1 Greenplum

Greenplum [16] – аналитическая база данных, основанная на архитектуре massive parallel processing [21] и технологии с открытым исходным кодом PostgreSQL [24]. Кластер состоит из главного узла, резервного главного узла и сегментных узлов. Все данные находятся на узлах сегмента, а информация каталога хранится в главных узлах. Узлы сегмента отвечают за один или несколько сегментов, которые являются модифицированными экземплярами БД PostgreSQL. Для каждой таблицы данные распределяются между узлами сегмента на основе ключей столбцов распределения с учетом репликации. Помимо основных свойств в виде эффективной оптимизации запросов, высокого параллелизма, масштабируемости и надежности, особым преимуществом является расширение PXF [17], позволяющее интегрироваться с внешними источниками данных, включая Hadoop [4] и PostgreSQL.

Кластер Greenplum компании состоит из нескольких физических и виртуальных машин, в который входят два главных узла и 6 серверов-сегментов с RAID10 дисками общим объемом около 100ТБ.

2.5.2 Debezium

Debezium [8] – это проект с открытым исходным кодом, основанный на чтении журнала транзакций и предоставляющий потоковую передачу данных с низкой задержкой для CDC. Debezium предоставляет единую модель всех событий изменений и сохраняет только успешно совершившиеся транзакции. Кроме того, инструмент записывает историю изменений данных в надежные реплицируемые журналы, в частности, Kafka – распределенный брокер сообщений, узлы которого содержатся на нескольких кластерах. [5]

Kafka гарантирует, что все события изменения данных реплицируются и полностью упорядочиваются, а при перезапуске работа возобновляется именно с того места, на котором остановилась. Каждый клиент может определить, требуется ли ему «exactly-once» или «at-least-once» доставка, а также все события изменения данных для каждой таблицы доставляются в том же порядке, что и в основной базе данных.

2.5.3 Airflow

Apache Airflow [3] – это платформа для программного создания, планирования и мониторинга рабочих процессов, выражающихся в виде ориентированных ациклических графов (DAG). DAG определяется с помощью кода на языке Python.

Airflow состоит из нескольких компонентов, обеспечивающих эффективное управление задачами: планировщика, вебсервера, базы данных с метаинформацией и рабочих узлов.

Основными преимуществами Airflow перед аналогами являются открытый исходный код, большая поддержка сообщества и широкая интеграция с различными источниками данных и библиотеками.

В рамках данной работы Airflow хорошо подходит для оркестрации и мониторинга выполнения задач по выгрузке данных из БД онлайн-платформы во внутреннюю инфраструктуру с Greenplum, преобразованию данных, доставке до базы данных API сервиса.

2.5.4 Golang

В качестве языка программирования для реализации API-сервиса был выбран статически типизируемый компилируемый язык Golang [13]. Основной причиной, прежде всего, стала высокая скорость выполнения программ по сравнению с Python или Java [11]. Кроме того, язык имеет достаточно простой в освоении синтаксис, богатую стандартную библиотеку, а также множество готовых пользовательских решений для работы с базами данных и создания веб-сервисов.

3 Требования

В ходе обсуждения с командой разработки платформы онлайн-обучения к сервису были сформулированы следующие функциональные требования:

- расчет показателей успешности прохождения программ, отдельных тестов и тем пользователями;
- расчет средних показателей успешности прохождения программ, тестов и вопросов пользователя;
- получение основного отчета по сотрудникам, отделам, программам и вопросам;
- получение отчета по медалям пользователей по сотрудникам, отделам и программам;
- общая сводная статистика по всем пользователям;
- генерация подробных отчетов по ответам, программам и тестам в формате JSON в объектное хранилище.

Также к сервису были собраны нефункциональные требования, а именно:

- основная функциональность должна быть доступна через HTTP запросы, максимальное время ответа – 10 сек;
- запросы на генерацию подробных отчетов будут посылаться асинхронно, максимальное время обработки – 10 мин;
- данные должны обновляться не реже, чем раз в 6 часов.

4 Архитектура

4.1 Взаимодействие модулей системы

Для взаимодействия между модулями системы и внешними источниками была спроектирована архитектура, указанная на Рис. 3.

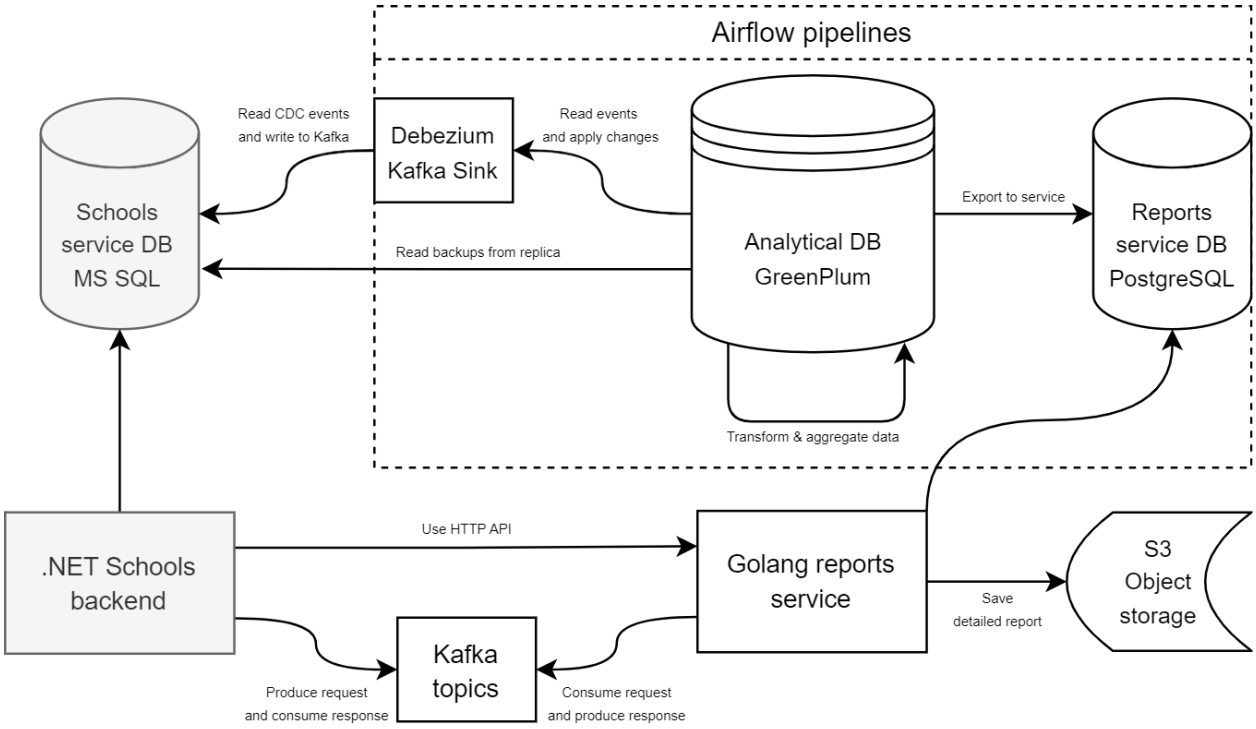

Рис. 3: Схема взаимодействия между модулями системы

Серым цветом обозначены существующие компоненты платформы образования. Остальную систему можно разделить на 3 части:

- Выгрузка таблиц из источника (базы данных MS SQL Server) в аналитическое хранилище. Этот процесс происходит с использованием инструмента Debezium, который записывает CDC-события по мере их поступления в очередь сообщений Kafka. Затем эти изменения периодически (с инетрвалом 6 часов) применяются к аналогичным таблицам в хранилище данных Greenplum. Кроме того, раз в сутки для синхронизации данных и устранения потенциальных ошибок происходит полная перезапись таблиц путем чтения с реплики БД-источника, что минимизирует нагрузку на базу-источник.

- Трансформация и агрегация данных в аналитическом хранилище и доставка до БД сервиса отчетов. На этих этапах выполняются тяжеловесные SQL запросы, создающие или обновляющие необходимые таблицы со статистикой, а также осуществляется копирование этих данных в отдельную транзакционную базу PostgreSQL, предназначенную для быстрого извлечения записей с фильтацией по входным параметрам.

  Стоит отметить, что вышеперечисленные процессы выполняются по расписанию и организованы в виде кода в соответствующих Airflow DAG.

- Веб-сервис отчетов, с которым непосредственно взаимодействует платформа онлайн-обучения. Он предоставляет интерфейс взаимодействия по протоколу HTTP, а также возможность посылать запросы через очередь сообщений Kafka. Ответы сервис формирует на основе полученных данных из PostgreSQL, а сгенерированные объемные отчеты загружает в объектное хранилище с протоколом S3.

Данная архитектура позволяет поддерживать актуальное состояние данных, избегая нагрузки на базу-источник. При этом элементы системы слабо связаны между собой, что позволяет отдельным компонентам продолжить работу, несмотря на состояние остальных (например, если будет недоступно хранилище Greenplum, то сервис все равно сможет обрабатывать запросы, используя последний снимок данных в PostgreSQL). Кроме того, систему можно легко масштабировать, добавляя новые таблицы-источники и расширяя API.

4.2 Сценарии использования

Основная цель спроектированной системы – предоставление сервисом отчетов API для бэкенда платформы онлайн-образования и дальнейшее использование полученных данных для отображения на странице веб-сайта. Поскольку сторона визуализации выходит за рамки дан-

ной работы, будет рассматриваться только непосредственное взаимодействие с сервисом-потребителем API, который и является актором на диаграмме последовательностей (Рис. 4).

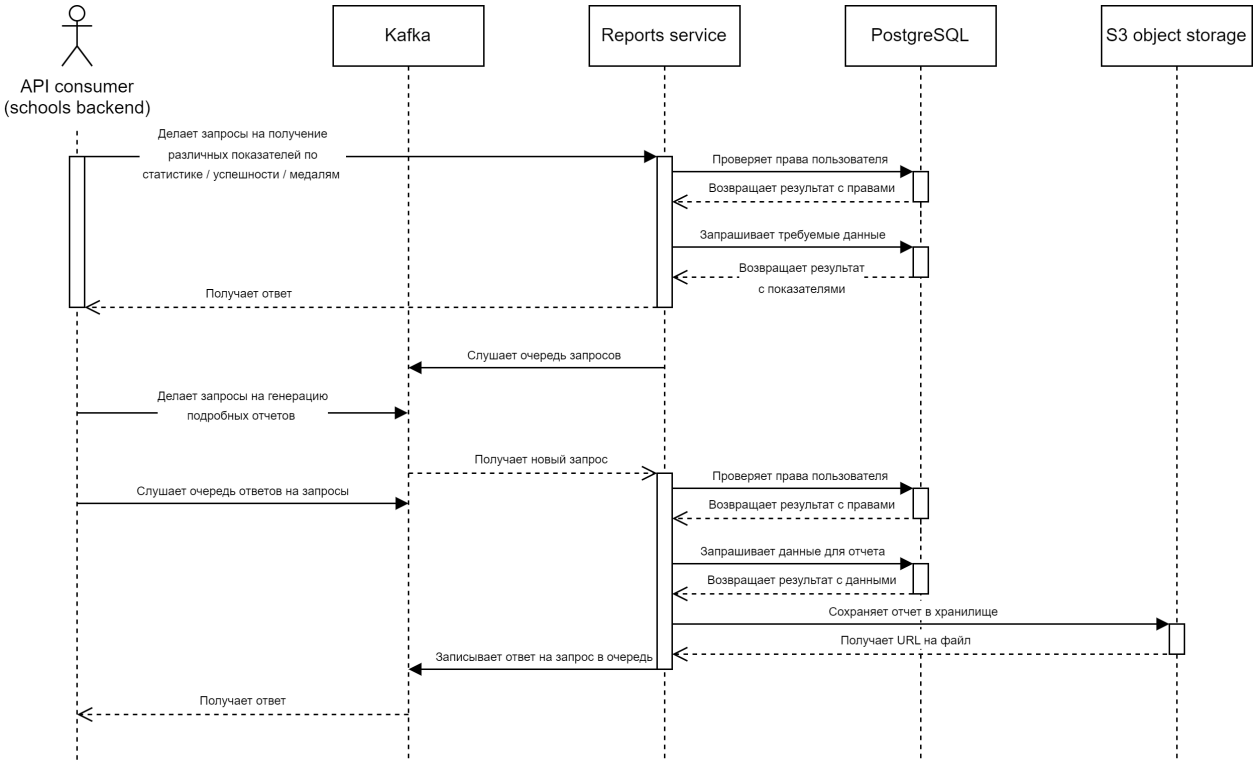

Рис. 4: Главные сценарии использования системы

Первым сценарием использования является инициация синхронного запроса к веб-серверу на получение различных показателей (например, средних показателей успешности прохождения программ, тестов, вопросов, или отчета по медалям пользователей). В этом случае запрос передается по протоколу HTTP с необходимыми параметрами, после чего сервис отчетов сначала проверяет права доступа пользователя, по которому запрашивается информация, и, в случае успеха, формирует и отправляет запрос на получение требуемых показателей к базе данных PostgreSQL. Полученный результат обрабатываются сервисом и формируется финальный ответ в заранее согласованном формате, который получает потребитель.

Другой сценарий использования подразумевает асинхронный обмен сообщениями. Потребитель создает запрос на генерацию подробного отчета и помещает его в отдельную очередь Kafka. Тем временем, сервис

отчетов слушает эту очередь и, по мере получения нового сообщения, начинает его обрабатывать: проверяет права доступа и делает запрос в базу PostgreSQL. Полученные данные сервис трансформирует и сохраняет в формате JSON в объектное хранилище с протоколом S3. После успешного сохранения формируется сообщение, включающее в себя сгенерированный идентификатор файла, и отправляется в очередь Kafka с ответами на запросы. Все это время потребитель, подписанный на эту очередь, ожидает ответного сообщения.

4.3 Сервис генерации отчетов

Архитектура сервиса генерации отчетов разделена на 4 логических уровня (Рис. 5).

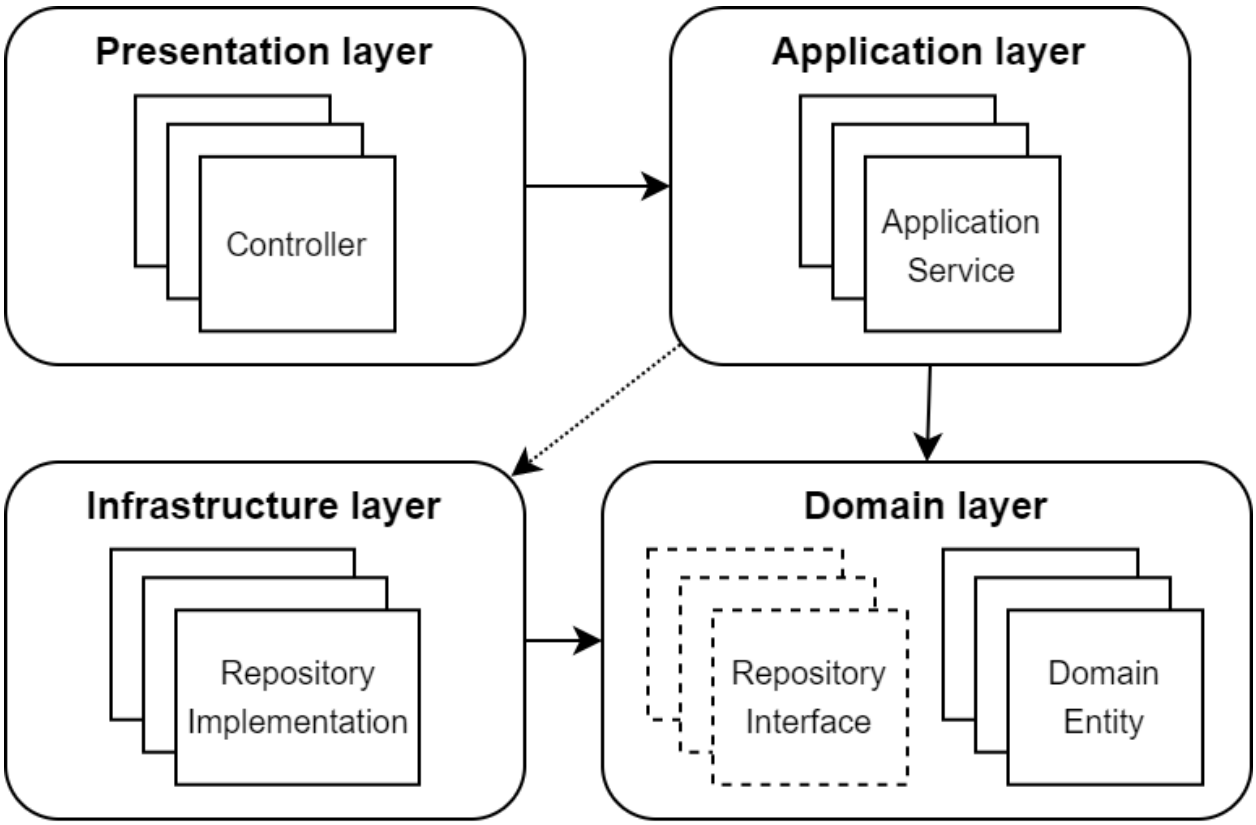

Рис. 5: Логические уровни сервиса отчетов

- Уровень презентации содержит точки входа в API сервиса. Здесь определены объекты-контроллеры, которые принимают запросы,

валидируют входные параметры и формируют ответы, обращаясь к сервисам прикладного уровня. Именно в контроллерах реализованы контракты по взаимодействию с клиентами сервиса, например, структура сообщения Kafka или HTTP запроса.

- Прикладной уровень состоит из объектов-сервисов, которые содержат логику приложения. На этом слое происходит создание и работа с доменными сущностями с доступом к инфрастуктурному слою по интерфейсам. Таким образом, обеспечивается изоляция доменного уровня от уровня презентации.

- Доменный уровень представляет сущности, процессы и правила, непосредственно связанные с бизнес-концепциями. Например, тут определены структуры конкретных типов отчетов, оценок и статистик. Кроме того, на этот уровень вынесены интерфейсы взаимодействия с внешними системами для работы с данными – репозиториями. Доменный уровень полностью независим от других слоев приложения.

- Инфраструктурный уровень определяет конкретные способы реализации хранения данных. Здесь происходит взаимодействие с базой данных PostgreSQL и хранилищем S3.

Разделение сервиса на слои позволяет структурировать приложение и работать с абстракциями, что упрощает возможность переиспользования, изменения и расширения отдельных компонентов.

5 Особенности реализации

5.1 Применение CDC-событий и реплицирование

Загрузка данных из базы-источника происходит с помощью коннектора инструмента Debezium для MS SQL Server, который использует механизм change data capture [30] и сохраняет операции INSERT, UPDATE и DELETE в специальные системные таблицы. При первичном чтении Debezium делает снимки таблиц в виде событий, которые записываются в отдельные очереди Kafka. Затем Debezium начинает отслеживать последующие изменения строк.

С помощью Kafka Connect [18] события сохраняются в надежную распределенную файловую систему HDFS в сжатом формате parquet. Отдельный запускающийся по расписанию процесс периодически извлекает события из этих файлов и последовательно применяет их к соответствующим таблицам в Greenplum, используя расширение Platform Extension Framework (PXF) [17].

В редких случаях описанный механизм может привести к ошибкам в данных. Поскольку процесс синхронизации с Greenplum выполняет операции удаления в последнюю очередь, то запись, которая сначала была удалена, а затем добавлена, может потеряться. Для исправления этой проблемы ежедневно (во время наименьшей нагрузки на источник) запускается чтение полной резервной копии таблиц с помощью инструмента sqoop [6], что позволяет держать данные в Greenplum в актуальном состоянии.

5.2 Формирование агрегированных таблиц для отчетов

Из более чем 50 таблиц-источников происходит формирование 9 таблиц с агрегированными данными. Обновление таблиц выполняется с помощью SQL запросов, использующих операции фильтрации, группировки и соединения. Для упрощения кода и оптимизации в некоторых случаях используются представления и временные таблицы. Также ско-

рость выполнения запросов зависит от способа распределения таблицы в кластере Greenplum – для этого ключом распределения были выбраны поля, наиболее часто встречающиеся в операциях JOIN. Там, где это возможно, используется партиционирование, что сокращает объем данных для сканирования, позволяя запросу читать только нужные разделы. В случаях таблиц, обновляющихся через полную перезапись, хранение данных происходит в колоночном формате с использованием компрессии.

Агрегированные данные копируются в БД PostgreSQL, с которой взаимодействует сервис отчетов, через внешние таблицы с использованием расширения Greenplum PXF. Для сохранения максимальной доступности создаются промежуточные таблицы, которые затем подменяют настоящие через операцию RENAME. Также для ускорения доступа к данным создаются индексы по полям фильтрации, которые приходят в запросах к сервису.

5.3 Оркестрация процессов в Airflow

Процессы оркестрации и мониторинга обновления данных описаны с помощью трех Airflow DAG на языке Python:

- DAG, применяющий события CDC. Он использует внутненнюю библиотеку, которая позволяет задавать источники через конфигурационные файлы. Обновление происходит раз в 6 часов.
- DAG, делающий резервную копию источников ежедневно в ночное время. Здесь используется `SSHOperator` для вызова утилиты загрузки данных sqoop.
- DAG, обновляющий агрегированные таблицы и загружающий их в PostgreSQL. Он начинает работу после применения CDC-событий и использует `PostgresOperator` для выполнения запросов.

5.4 Поддержка HTTP API и асинхронных запросов в Kafka

Сервис генерации отчетов написан на языке Go версии 1.19 и представляет собой веб-сервер на основе библиотеки fasthttp [31]. Сервис принимает HTTP запросы на чтение данных, используя метод POST, что позволяет передавать параметры в теле запроса и избежать ограничения на длину строки URL. Асинхронная обаботка осуществляется за счет создания отдельной горутины [14] на запрос, а для ограничения времени выполнения запроса используется контекст с таймаутом.

Объекты сервиса управляются библиотекой внедрения зависимостей dig [32], основанной на рефлексии. Взаимодействие с базой данных PostgreSQL осуществляется за счет библиотеки gorm [12], обеспечивающей объектно-реляционное отображение.

Работа с Kafka происходит в отдельном фоновом потоке. Для приема и передачи сообщений инициализируется клиент sarama [27]. Для сервиса создается консьюмер-группа, которая подписана на очередь запросов. При поступлении очередного запроса он добавляется в локальный буфер, сообщения из которого по мере достижения лимита или таймаута параллельно обрабатываются. Также сервис выступает и в роли продьюсера, публикуя результаты обработки в очередь с ответами. Учтен случай ребалансировки, обрабатывающийся через завершение контекста и каналы ошибок.

5.5 Развертывание сервиса и мониторинг

Для создания образа сервиса используется Dockerfile [9]. Сервис развернут в Kubernetes [20] – платформе для автоматического управления контейнеризованными приложениями, а обновления происходят с помощью werf [33], утилиты командной строки, помогающей в организации всего цикла доставки приложений в Kubernetes. Для обеспечения надежности используется 3 экземпляра сервиса, нагрузку между которыми распределяет балансировщик Kubernetes, принимая запросы на

единый адрес.

Мониторинг состояния сервиса происходит с помощью метрик, которые доступны по служебному адресу для сбора Prometheus [25]. Наблюдать за метриками можно в инструменте визуализации Grafana [15]: на основе метрик были построены дашборды, показывающие как внутреннее состояние (количество горутин, работу сборщика мусора, потребление памяти), так и количественные метрики (например, количество запросов, время ответа, длительность записи в БД, отставание по обработке очереди Kafka).

Логи сервиса сохраняются с помощью стека ELK [10], что обеспечивает удобный поиск и эффективное хранение.

6 Тестирование и апробация

Разработка Airflow DAG велась с использованием тестовых данных и отдельного изолированного окружения для разработки. Кроме того, после завершения загрузки таблиц были встроены базовые проверки на ненулевое число добавленных записей. В будущем набор проверок для обеспечения качества данных будет расширяться.

Тестирование сервиса генерации отчетов можно разделить на несколько частей.

- Модульные тесты, подтверждающие, что код отдельных сервисов работоспособен на тестовых примерах. Такими проверками покрывались сервисы прикладного и доменного уровня, покрытие составило более 80% строк кода этих слоев.

- Интеграционные тесты, проверяющие работу HTTP API. На каждый новый вид запроса был написан тест, проверяющий ответы с разными статусами, всего было протестировано более 15 видов запросов.

- Нагрузочное тестирование, при котором на сервис с продуктовыми данными поступало по 10 запросов в секуду в течение получаса. Средняя скорость ответа – 61 мс, а 99 перцентиль составил 2 секунды, что удовлетворяло договоренностям с командой-заказчиком. Созданием таких тестовых сценариев занимался отдельный сотрудник.

- Ручное тестирование со стороны команды-заказчика на стенде разработки нововведений. В результате обратной связи были найдены и исправлены ошибки и пограничные случаи, которые не были учтены при разработке.

По результатам исправления замечаний сервис был внедрен заказчиком в работу бэкенда платформы онлайн-образования.

Заключение

В ходе данной работы были получены следующие результаты.

- Произведен обзор существующего решения, которое перестало удовлетворять требованиям бизнеса ввиду роста объема данных. Выполнено сравнение OLTP- и OLAP-подходов, механизмов CDC и репликации; сделан вывод о том, что OLAP-хранилище вместе с CDC позволяет эффективно хранить и агрегировать данные, не нагружая источник.

- Выявлены и сформулированы требования к системе: функциональные (типы генерируемых отчетов, показателей и статистик), а также нефункциональные (способы взаимодействия с системой, частота обновления данных, время ответа).

- Спроектирована архитектура решения, включающая в себя компоненты по загрузке, агрегации и экспорту данных, а также сервис, предоставляющий интерфейс для получения данных отчетов.

- Реализована следующая функциональность:

  - настроены процессы загрузки данных из источника в аналитическое хранилище Greenplum с помощью Airflow и Debezium;
  - созданы таблицы с агрегированной статистикой и написан Airflow DAG для их обновления и загрузки в транзакционную базу сервиса PostgreSQL;
  - реализован сервис генерации статистики по отчетам на Go, предоставляющий как синхронное HTTP API, так и асинхронно обрабатывающий очередь запросов в Kafka, покрыт модульными тестами и настроен мониторинг.

- Проведено нагрузочное и приемочное тестирование, в результате чего были исправлены дефекты и недоработки со стороны бизнес-логики и производительности системы.

- Сервис был внедрен и использован для визуализации статистики по обучению сотрудников.

Список литературы

[1] ACID. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/ACID.

[2] ASP.NET Web APIs. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://dotnet.microsoft.com/en-us/apps/aspnet/apis.

[3] Apache Airflow. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://airflow.apache.org/.

[4] Apache Hadoop. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://hadoop.apache.org/.

[5] Apache Kafka. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://kafka.apache.org/.

[6] Apache Sqoop. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://sqoop.apache.org/.

[7] ClickHouse. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://clickhouse.com/docs/ru.

[8] Debezium. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://debezium.io/.

[9] Dockerfile reference. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://docs.docker.com/engine/reference/builder/.

[10] The ELK Stack: From the Creators of Elasticsearch | Elastic. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://www.elastic.co/what-is/elk-stack.

[11] Fastest cpu secs Go versus Java. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/go.html.

[12] GORM - The fantastic ORM library for Golang. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://gorm.io/.

[13] The Go programming language. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://go.dev/.

[14] Goroutines. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://go.dev/doc/effective_go#goroutines.

[15] Grafana: The open observability platform | Grafana Labs. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://grafana.com/.

[16] Greenplum. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://greenplum.org/.

[17] Greenplum Platform Extension Framework (PXF). — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://greenplum.docs.pivotal.io/6-7/pxf/overview_pxf.html.

[18] Kafka Connect. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://docs.confluent.io/platform/current/connect/index.html.

[19] Kleppmann Martin. Designing Data-Intensive Applications. — Beijing : O'Reilly, 2017. — ISBN: 978-1-4493-7332-0. — URL: https://www.safaribooksonline.com/library/view/designing-data-intensive-applications/9781491903063/.

[20] Kubernetes Documentation. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://kubernetes.io/docs/home/.

[21] Massive Parallel Processing. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/Massively_parallel.

[22] Online analytical processing. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/Online_analytical_processing.

[23] Online transaction processing. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/Online_transaction_processing.

[24] PostgreSQL: The world's most advanced open source database. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://www.postgresql.org/.

[25] Prometheus - Monitoring system time series database. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://prometheus.io/.

[26] Representational state transfer. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/Representational_state_transfer.

[27] Sarama is a Go library for Apache Kafka. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://github.com/Shopify/sarama.

[28] Understanding Change Data Capture (CDC): Definition, Methods and Benefits. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://airbyte.com/blog/change-data-capture-definition-methods-and-benefits.

[29] Vertica. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://en.wikipedia.org/wiki/Vertica.

[30] What is change data capture (CDC)? — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server.

[31] fasthttp. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://github.com/valyala/fasthttp.

[32] A reflection based dependency injection toolkit for Go. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://github.com/uber-go/dig.

[33] werf: Эффективный и консистентный CI/CD с Kubernetes. — [Электронный ресурс]. — (дата обращения: 01.05.2023). URL: https://ru.werf.io/.