УДК 004.8 **Информационные технологии**

Григорьев Михаил Александрович, *студент 2 курса магистратуры, РТУ МИРЭА, Институт кибербезопасности и цифровых технологий, кафедра КБ-3 "Безопасность программных решений", Россия, г. Москва*

Волович Михаил Евгеньевич, *кандидат технических наук, доцент кафедры КБ-3 "Безопасность технических решений" РТУ МИРЭА. Россия, г. Москва*

МЕТОДИКА РЕАЛИЗАЦИИ УСТОЙЧИВОГО К ИЗМЕНЕНИЯМ ИТЕРАЦИОННОГО МЕХАНИЗМА ИНЖЕКТИРОВАНИЯ РЕЛЯЦИОННОГО ХРАНИЛИЩА В ОЗЕРО ДАННЫХ

Аннотация: Данная статья посвящена описанию методики создания устойчивого механизма передачи данных из источников данных корпоративных систем в озеро данных с учетом изменяющегося ландшафта систем. Упомянуты различные архитектуры, которые могут быть использованы в качестве альтернативных вариантов, также описаны их преимущества и недостатки. Рассмотрены конкретные примеры реагирования на изменение схемы данных.

Ключевые слова: озеро данных, схема данных, источник данных, устойчивость, консолидация, модель машинного обучения.

Abstract: This article is devoted to the description of a methodology for creating a sustainable mechanism for transferring data from data sources of enterprise systems to a data lake, taking into account the changing landscape of systems. Various architectures are mentioned that can be used as alternatives, and their advantages and disadvantages are also described. Specific examples of responding to a change in the data schema are considered.

Keywords: data lake, data schema, data source, persistence, consolidation, machine learning model.

Введение

В настоящее время подавляющее большинство информационных систем имеют длительный цикл развития, в течение которого схема данных может многократно изменяться. В обычном случае это приводит к тому, что, помимо требующихся изменений самой системы для работы с новой схемой данных, старые данные требуется либо вручную приводить к новому виду, либо они становятся неподходящими для модели машинного обучения. Так как оба варианта являются неприемлемыми, необходимо автоматически обеспечивать устойчивость и единообразие схем данных, получаемых с разных источников, для того, чтобы в дальнейшем строить модели машинного обучения таким образом, чтобы они не были дискредитированы. Также необходимо обеспечивать согласованность данных. Инженеры часто не задумываются о целостности данных в том смысле, что пользователи в большинстве своем выгружают данные, не смотря на то, что они могут быть не согласованы в данный момент времени. Если собирать данные со всех систем в единый момент времени, то есть вероятность, что в некоторых из этих систем данные будут еще не посчитаны. Поэтому следует, во-первых, собирать данные из источников в разные моменты времени, а во-вторых, потребителю при выборе среза данных для загрузки следует различать дату загрузки и дату консолидации данных [1].

Подходы к преобразованию данных

Различают несколько подходов: изоляция преобразования схемы данных к стандартизированному виду на уровне источника данных и на уровне системы, подготавливающей данные для дальнейшего потребления моделями машинного обучения [2].

Преобразование схемы на уровне системы источника данных имеет ряд существенных недостатков. Во-первых, требуется отдельная поддержка трансформации для каждого источника данных, что как минимум является неоптимальным по трудозатратам, а также это нарушает ETL-процесс, который, как предполагается, должен быть реализован полностью на стороне системы,

подготавливающей данные для моделей машинного обучения [3; 4]. Во-вторых, это требует постоянного наличия свободных ресурсов на стороне системы источника данных. В-третьих, это создает точку отказа: если в процессе преобразования данных произошла непредвиденная ошибка, то данные не попадут в озеро данных. Также система преобразования данных на стороне источника может не знать, как обработать тот или иной случай изменения схемы данных.

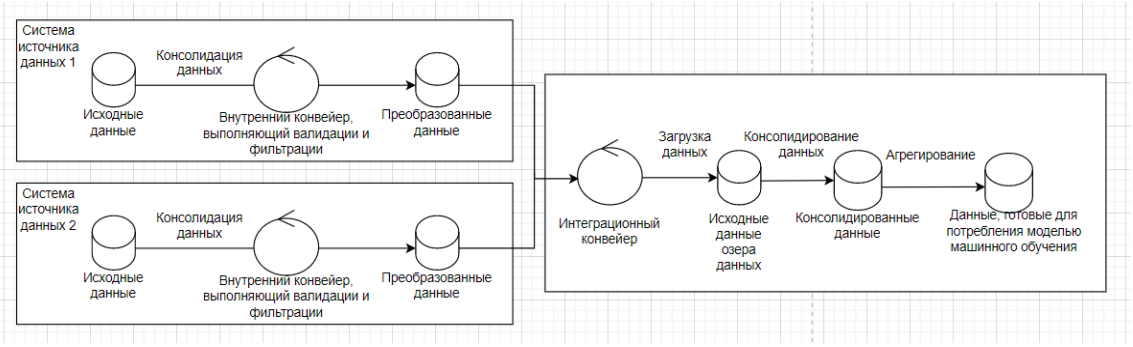

Рисунок 1 – Архитектура системы с преобразованием данных на стороне источника

При преобразовании данных на стороне нашей системы мы, помимо отсутствия недостатков, описанных выше, имеем возможность отслеживать все изменения в модели данных, которые происходили с момента начала существования системы. Также, чтобы избежать точки отказа на стороне системы источника данных, необходимо построить прямое соединение, чтобы забирать данные как есть для их дальнейшей обработки и консолидации на стороне озер данных. Хранение исходных данных на стороне озера не оптимально по затратам памяти, но мы можем использовать политики удаления данных, которые будут заниматься агрегированием данных спустя какое-то время после их появления в системе. Например, за последний месяц могут храниться все дневные данные, а более старые будут агрегированы и будут хранить лишь недельные данные, а еще более старые – месячные. При этом при потере все данные возможно восстановить, так как исходные данные будут архивироваться и храниться в csv формате, а консолидированные – в формате parquet.

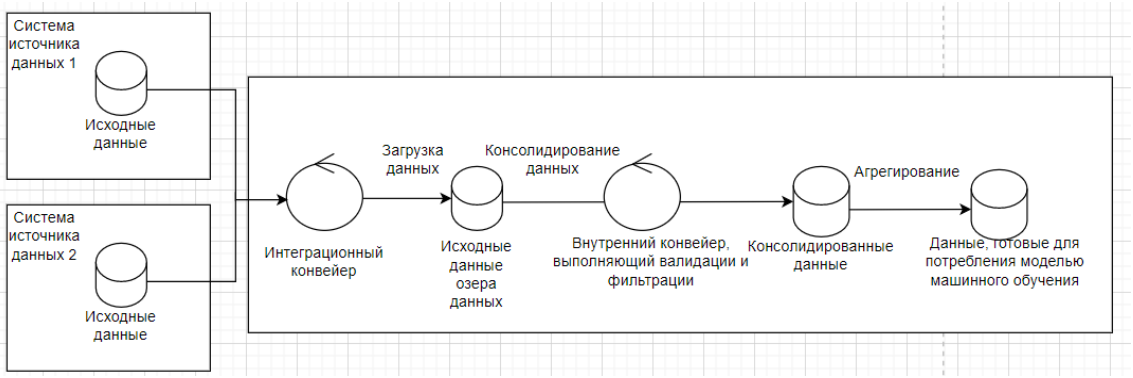

Рисунок 2 – Архитектура системы с преобразованием данных на стороне озера

Таким образом, мы можем использовать два подхода для загрузки данных из источника: либо каждый раз выгружать все данные целиком, либо выгружать только дельту – те данные, которые были изменены с момента последней загрузки. Второй вариант является оптимальным, но является более сложным в реализации и не всегда возможным. Так, например, для возможности выгружать только дельту каждая запись в данных должна содержать признаки временных отсечек: даты создания и последней модификации. Также необходимо иметь возможность отслеживать удаленные записи – для этого на стороне озера данных вводится технический столбец-маркер, обозначающий, была ли запись удалена, значение которого заполняется в случае, если при новой выгрузке данных не обнаружилось записи, которая уже хранится в озере данных [5].

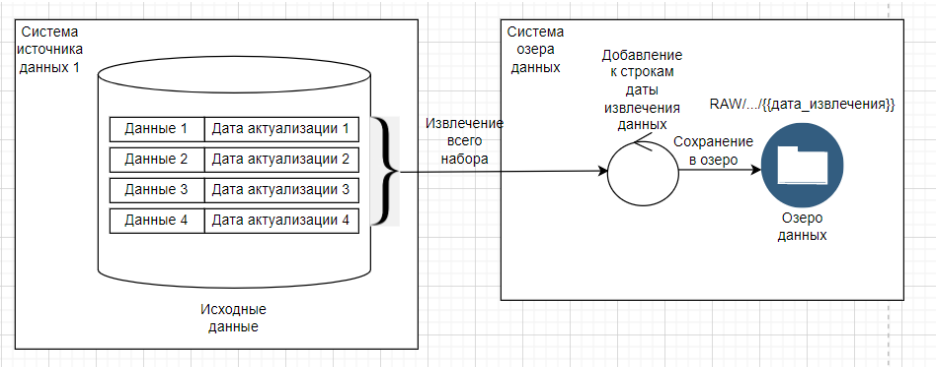

Рисунок 3 – Подход с полной выгрузкой данных из источника

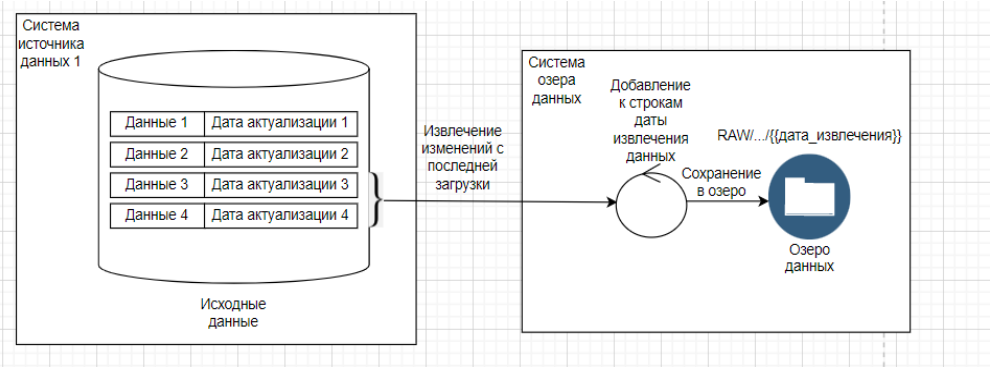

Рисунок 4 – Подход с выгрузкой дельты из источника

При подходе, в котором данные преобразуются на стороне озера данных, присутствуют другие проблемы, связанные с изменениями внутри системы и необходимостью их контроля [6]. Необходимы политики системы, которые будут описывать требования к алгоритму обработки каждого возможного случая изменения схемы данных. Некоторые случаи могут быть обработаны автоматически; при возникновении определенных случаев следует приостановить процесс преобразования данных, отослать нотификацию и ожидать решения проблемы оператором системы вручную. Главная цель, на которую нужно опираться при разработке политик – не допустить не согласованных данных, иначе это приведет к дискредитации модели.

Примеры реагирования на изменения схемы данных

Ниже описаны реакции на различные виды изменений в схеме данных, которые могут быть приняты или не приняты с точки зрения потребителя данных.

При изменении порядка столбцов в схеме никаких изменений не требуется, потому что система изначально хранит схему данных и производит выборку из источника именно в том порядке следования столбцов, который описан в сохраненной схеме.

При добавлении нового столбца в источник в процессе преобразования схемы не требуется изменений – в старых записях появится этот столбец с пустым значением.

При удалении столбца в источнике нельзя допустить появления пустых

значений в новых записях в этом столбце, так как модель машинного обучения может в дальнейшем агрегировать данные в этом столбце, и при существовании пустых значений в нем не сможет корректно это сделать, или в худшем случае сгенерирует исключение. Поэтому в зависимости от типа столбца необходимо преобразовать пустые значения в незначащие в зависимости от типа: например, “0” для численных типов, или пустую строку для строковых.

Изменение типа столбца можно обработать двумя способами. Первый заключается в том, чтобы технически реализовать это как две операции: удаление столбца и добавление нового, но в таком случае модель машинного обучения должна знать о том, что тип столбца был изменен, поэтому это не является оптимальным решением. Таковым является попытка привести старые значения столбца к новому типу, однако это не во всех случаях является возможным или целесообразным. Например, не всегда возможно привести строковый тип к целочисленному типу или к типу даты. В качестве другого примера можно взять приведение типа с плавающей точкой к целочисленному типу – технически это всегда возможно, но не всегда целесообразно. Также приведение типа даты к строковому типу не всегда может осуществлено автоматически, потому что необходимо знать формат записи даты, в котором она представлена в столбце в новых записях, чтобы привести старые значения ровно к такому же формату.

Для определения, в каких случаях необходимо вмешательство оператора, хранится файл, в котором описаны допустимые трансформации – если текущее изменение схемы не описано в файле, вся обработка приостанавливается, и посылается нотификация с информацией о том, что необходимо вмешательство [7].

В целом основные моменты системы, реализующей такую методику устойчивого механизма инжектирования данных в озеро, приведены ниже:

Прежде всего, необходимо хранить эталон схемы данных и при каждой загрузке данных из источника сверяться с ним, а также необходим файл, описывающий допустимые случаи изменения схемы [8]. При выгрузке данных

следует провести первоначальную проверку качества данных, как минимум то, что количество выгруженных строк соответствует количеству строк на стороне источника данных. После выгрузки данных следует обработать их на случай изменения схемы путем, описанным выше. Затем, в зависимости от того, каким образом выгружаются данные – целиком или же дельтой – следует разбить данные на актуальные и исторические, при этом нужно учесть то, что уже сохраненные в озере данных записи могли быть изменены или удалены на стороне источника. После этого осуществляется группировка и агрегирование данных в зависимости от требований, диктуемых моделью машинного обучения. На каждом этапе осуществляется журналирование промежуточных состояний.

Вывод

В данной статье были описаны различные методики преобразования и выгрузки данных из источника в озеро, описаны их преимущества и недостатки, а также была выбрана оптимальная методика. Такая методика может быть реализована на любой методологической базе, так как используемые механизмы позволяют работать как с реляционными базами данных, так и с не реляционными. Были приведены примеры реагирования на изменения схемы данных, а также была описана высокоуровневая архитектура всей системы.

Библиографический список:

1. Д. Аношин, Д. Фошин, Р. Сторчак. Azure Data Factory Cookbook. Build and manage ETL and ELT pipelines with Microsoft Azure's serverless data integration service. // Packt Publishing, 2020.

2. Н. Марц, Д. Уоррен. Большие данные. Принципы и практика построения масштабируемых систем обработки данных в реальном времени // Вильямс, 2017.

3. Сенько А. В. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. // Питер, 2018.

4. Су Кеннет, Анналин Ын. Теоретический минимум по Big Data. Всё что нужно знать о больших данных. // Питер, 2019.

5. J. Goldberg, L. Pierson. Managing Big Data Workflows for Dummies. // John Wiley & Sons, Inc., 2016.

6. Служба хранилища Azure Data Lake. URL:https://azure.microsoft.com/ru-ru/services/storage/data-lake-storage/ [Электронный ресурс].

7. Преобразование источника в потоке данных для сопоставления. URL:https://docs.microsoft.com/ru-ru/azure/data-factory/data-flow-source [Электронный ресурс].

8. Извлечение, преобразование и загрузка (ETL). URL: https://docs.microsoft.com/ru-ru/azure/architecture/data-guide/relational-data/etl [Электронный ресурс].