Geekbrains

Дипломный проект:

«Инструмент Data Quality в банковском секторе в BIG DATA»

IT-специалист:

Python программист

Савицкий Олег Викторович

г. Москва

2023 г.

Содержание:

Оглавление

[Введение 3](#_Toc151909850)

[Почему была выбран данная тема. 5](#_Toc151909851)

[Анализ данных в банковской сфере 7](#_Toc151909852)

[Data Quality в банковской сфере 11](#_Toc151909853)

[Модель инструмента в банковской сфер 14](#_Toc151909854)

[Архитектура 14](#_Toc151909855)

[Функциональные необходимость. 17](#_Toc151909856)

[Объектная модель DQ 19](#_Toc151909857)

[Структура YAML конфига 22](#_Toc151909858)

[Источник данных. SOURCES 24](#_Toc151909859)

[Параметры источников 26](#_Toc151909860)

[Объекты проверок. CHECK\_OBJECTS 30](#_Toc151909861)

[Метрики. METRICS 31](#_Toc151909862)

[Сравнение. COMPARES 37](#_Toc151909863)

[Группы. Groups 43](#_Toc151909864)

[Параметры запуска по расписанию 45](#_Toc151909865)

[Алерты. Alerts 47](#_Toc151909866)

[Механизм работы prev\_metric (Конвейер метрик) 51](#_Toc151909867)

[Метрики нал результатами из базы DQ (метрики второго порядка) 54](#_Toc151909868)

[Результаты проверок (compare\_result\_view) 56](#_Toc151909869)

[Интеграция в ETL-процесс 58](#_Toc151909870)

[Ролевая модель 59](#_Toc151909871)

[Вывод 60](#_Toc151909872)

# Введение

В современном мире финансовая сфера играет огромную роль в экономическом развитии страны. Для эффективной работы с данными, которые поступают из различных источников, необходимо обеспечить высокое качество информации. Data Quality позволяет выявлять ошибки в данных, устранять их и предотвращать дальнейшие проблемы. В связи с этим вопросы качества данных в финансовой сфере становятся все более актуальными.

В условиях глобальной цифровизации и роста объемов информации, использование больших данных (Big Data) предоставляет банковским организациям огромные возможности для повышения качества услуг и оптимизации бизнес-процессов. Однако, для того чтобы успешно работать с Big Data, необходимо гарантировать высокое качество информации, что требует применения специализированных инструментов.

Цель данного дипломного проекта - исследовать и оценить качество данных в финансовой сфере, выявить проблемы, связанные с этим, и предложить решения, позволяющие улучшить качество данных и повысить надежность финансовых отчетов.

Для достижения поставленной цели были проведены анализ существующих подходов к обработке и хранению данных в финансовой сфере, а также определены основные проблемы, связанные с качеством финансовых данных. В работе были также рассмотрены современные технологии и методы, которые позволяют улучшить качество данных в финансовой сфере.

Таким образом, исследование темы "Инструмент Data Quality в банковском секторе в BIG DATA" может привести к разработке эффективных методов и инструментов, способных обеспечить высокое качество данных и повысить эффективность использования больших данных в банковской сфере

В заключение были сформулированы рекомендации по улучшению качества данных и повышению надежности финансовых отчетов, что, безусловно, позволит увеличить эффективность и надежность финансовой сферы, а также снизить риски, связанные с качеством данных.

В дипломной работе представлен целевой инструмент для работы с качеством данных, когда представленный инструмент — это прототип(немного с другой логикой).

# Почему была выбран данная тема.

Качество данных является ключевым фактором в принятии решений в банковской индустрии. Неверные или недостоверные данные могут привести к ошибочным решениям, а этим - к неудачам в бизнесе. Банки хранят большое количество информации о своих клиентах, своих финансовых операциях, а также о рынке и конкурентах. Если эта информация недостаточно качественна, могут возникнуть серьезные проблемы: недостоверность отчетности, увеличение рисков и потеря доверия со стороны клиентов.

На протяжении, более 10 лет работаю в банковской сфере, в сфере анализа данных, в различных базах данных (Oracle, Teradata. Hive, Postgre) и все чаще встречаюсь с ошибками в данных, которые могут привести к потерям прибыли, уменьшению клиентской базы и снижению эффективности бизнес-процессов.

Одним из основных факторов, влияющих на качество данных, является людской фактор - невнимание, малопрофессиональный подход или недостаток квалификации со стороны персонала, а также слишком быстрое и рискованное принятие решений (пример - плохо протестировали доработку/потоки не тестовом стенде).

Поэтому для организаций качество данных становится все более критически важным. Важно создавать системы и процедуры, направленные на обеспечение правильности, полноты и актуальности данных, а также на их безопасность. Кроме того, необходимо проводить регулярный мониторинг качества данных и регулярно обучать сотрудников работе с данными, чтобы снизить вероятность ошибок.

Для обеспечения качественных данных в банке, требуется современная и интеллектуальная инфраструктура, способная обеспечивать точность, доступность, своевременность и конфиденциальность информации. Эта инфраструктура должна основываться на автоматизированных системах, способных обнаруживать ошибки и предотвращать возможные риски.

Сейчас на рынке есть большое кол-во коробочных решений (Atacama, Informatica и др.) и open-source решений (Great Expectations, Amazon Deequ, Apache Griffin, Agile lab, Soda SQL, Qualitis, Ubisoft), но в последнее время в больших компаниях идет тенденция – создания своих собственных инструментов Data Quality. Это связано с тем, что коробочные решения и open-source решения не всегда удовлетворяют требованиям конкретной компании, а также не всегда позволяют реализовать необходимые бизнес-процессы и функциональность.

Создание своего инструмента Data Quality позволяет компаниям более гибко настраивать процессы обработки данных, а также реализовывать индивидуальные требования и функциональность. Кроме того, это позволяет компаниям лучше контролировать процессы обработки данных и обеспечивать более высокое качество данных.

Создание своего инструмента Data Quality может быть связано с большими затратами на разработку и поддержку, но в долгосрочной перспективе это может оказаться более выгодным, чем использование сторонних решений

В итоге, качество данных в информационных системах является критически важным фактором в успешной работе банка. Постоянный контроль и улучшение качества данных позволят увеличить эффективность работы банка и повысить удовлетворенность клиентов.

Компании, которые осознают значение качества данных, должны работать над улучшением всех вышеперечисленных факторов для того, чтобы увеличить эффективность своего бизнеса и сократить риски, связанные с плохими данными.

# Анализ данных в банковской сфере

В банковской сфере анализ данных играет важную роль для принятия решений, оптимизации бизнес-процессов и улучшения услуг для клиентов. Источниками данных в банках могут быть:

1. Внутренние источники информации для банка — это данные, получаемые внутри организации банка, которые могут использоваться для принятия решений. Такие источники могут включать в себя:
   1. Банковские системы учета и управления финансами: это включает в себя операционные системы, которые управляют банковскими транзакциями и операциями, такими как принятие депозитов, выдача кредитов, переводы денежных средств и т.д.
   2. Системы учета и управления клиентами: банки собирают информацию о клиентах, такую как их личные данные, кредитная история, финансовые показатели и т.д. Эти данные используются для управления отношениями с клиентами, анализа поведения клиентов и принятия решений по кредитованию.
   3. Онлайн-банкинг: системы онлайн-банкинга собирают данные о транзакциях и активности клиентов в реальном времени. Эти данные могут использоваться для отслеживания изменений в поведении клиентов и принятия решений на основе новой информации.
2. Внешние источники информации для банка - это данные, получаемые извне, которые могут использоваться для принятия банковских решений. Такие источники могут включать в себя::
   1. Рейтинговые агентства: такие компании, как Moody's и Standard & Poor's, предоставляют банкам информацию о кредитоспособности и риске заемщика. Эта информация может быть использована банками для принятия решений о выдаче кредита или инвестировании.
   2. Финансовые рынки: банки могут использовать информацию о тенденциях в финансовых рынках и экономике для принятия решений о вложениях, рисках и выдаче кредита. Например, данные об уровнях процентных ставок или изменениях в индексах могут быть использованы для анализа тенденций на рынке.
   3. Данные о клиентах: банки могут использовать данные о клиентах из внешних источников, таких как кредитные бюро, для оценки кредитного риска и принятия решений о выдаче кредитов. Эти данные могут включать в себя информацию о кредитной истории заемщика, его доходах и занятости.
   4. Открытые данные: государственные органы и другие организации предоставляют большое количество открытых данных, которые могут быть использованы банками для исследования рынка, анализа тенденций и поиска новых бизнес-возможностей.
   5. Социальные сети: информация, опубликованная в социальных сетях, может использоваться банками для анализа мнения клиентов, оценки конкуренции и выявления новых трендов.
3. Справочники (НСИ)
4. Репликации баз данных: Некоторые банки используют репликацию баз данных, чтобы обеспечить сохранность данных в случае отказа сервера или иных технических проблем или для объединения всех источников в одной базе данных (BIG DATA)
5. Плоские файлы: Это формат данных, хранящихся в одном файле, например, CSV, TXT или Excel. Такие файлы обычно содержат данные отчеты, которые собираются «руками»

Данные, которые собираются и анализируются в банке, могут включать в себя финансовые показатели, кредитную историю, информацию о транзакциях клиентов, аналитические данные, данные о поведении клиентов и многие другие. Анализ данных в банке может помочь в решении таких задач, как:

1. Оптимизация процессов кредитования: используя данные о кредитной истории клиентов и их финансовых показателях, банк может принимать решение о выдаче кредита, установить размер кредита, процентную ставку и сроки возврата.
2. Управление рисками: с помощью анализа данных банк может оценить риски клиентов, связанные с кредитами и инвестициями, и использовать эту информацию для управления рисками и минимизации возможных потерь.
3. Прогнозирование и планирование: банки используют анализ данных для прогнозирования финансовых показателей, таких как прибыль, потоки наличных денег и т.д. Эти прогнозы могут быть использованы для планирования бизнес-стратегии и принятия решений по расходам и инвестициям.
4. Улучшение качества обслуживания клиентов: анализ данных клиентов может помочь банку понимать их потребности, поведение и предпочтения, что позволяет улучшить качество обслуживания и предоставлять персонализированные услуги.
5. Развитие своей экосистемы
6. Разработка новых продуктов и услуг.
7. Создание персонализированных предложений целевой аудитории.

Это только некоторые примеры задач, которые банки могут решать на основе данных в области Big Data.

Однако, несмотря на все выгоды, анализ данных также имеет свои сложности. Одним из основных вызовов в банковской сфере является качество данных. Некачественные данные могут привести к неправильным выводам и ошибочным решениям. Проблемы с качеством данных могут возникнуть из-за ошибок ввода данных, неполных данных или использования устаревших данных.

Кроме того, наличие большого объема данных может привести к проблемам с их хранением и обработкой. Банки могут столкнуться с проблемами отсутствия инфраструктуры, необходимой для обработки больших объемов данных, и необходимостью использования высококвалифицированных сотрудников, чтобы правильно использовать эти данные.

В целом, анализ данных в банковской сфере представляет собой мощный инструмент для принятия решений и улучшения качества услуг для клиентов. Однако, банки должны быть готовы к проблемам с качеством данных и иметь соответствующие ресурсы и инфраструктуру для их обработки и использования.

# Data Quality в банковской сфере

Качество данных в банковской сфере — это использование средств и методов для обеспечения точности, полноты, своевременности и соответствия данных стандартам и требованиям, позволяющих банкам принимать правильные решения и соблюдать требования законодательства.

Ниже приведены инструменты, которые используются в банковской сфере для обеспечения качества данных:

* Data profiling - анализ и профилирование данных для определения их качества и целостности. Это может включать проверку на наличие дубликатов, пропуски в данных, ошибки формата и другие подобные аномалии.
* Data cleansing - очистка данных от ошибок и аномалий с использованием различных методов, таких как проверка наличия правильных значений, заполнение пропусков, удаление дубликатов и т.д.
* Data governance - управление данными с помощью стандартов и правил, которые обеспечивают соответствие данных требованиям и стандартам. Это может включать оформление документации, установление правил доступа к данным и т.д.
* Data monitoring - постоянный мониторинг данных, чтобы обнаруживать изменения в качестве данных, нарушения правил и другие подобные аномалии.
* Data quality scorecards - использование показателей для определения и оценки качества данных, таких как процент ошибок или отклонений от стандартов.
* Master data management - управление основными данными (например, данные о клиентах, товарах и т.д.) для обеспечения их точности и соответствия стандартам во всех системах и процессах банка.
* Data profiling tools - инструменты для автоматического анализа и профилирования данных для обнаружения аномалий и проблем с качеством данных.
* Data integration tools - инструменты для объединения данных из различных источников и обеспечения их совместимости и соответствия стандартам.
* Data quality dashboards - инструменты для визуализации и отслеживания показателей качества данных на панелях мониторинга.
* Data quality training - обучение сотрудников банка правилам и методам обеспечения качества данных для повышения эффективности и точности работы с данными.

Качество данных становится важным направлением для банковской сферы по нескольким причинам:

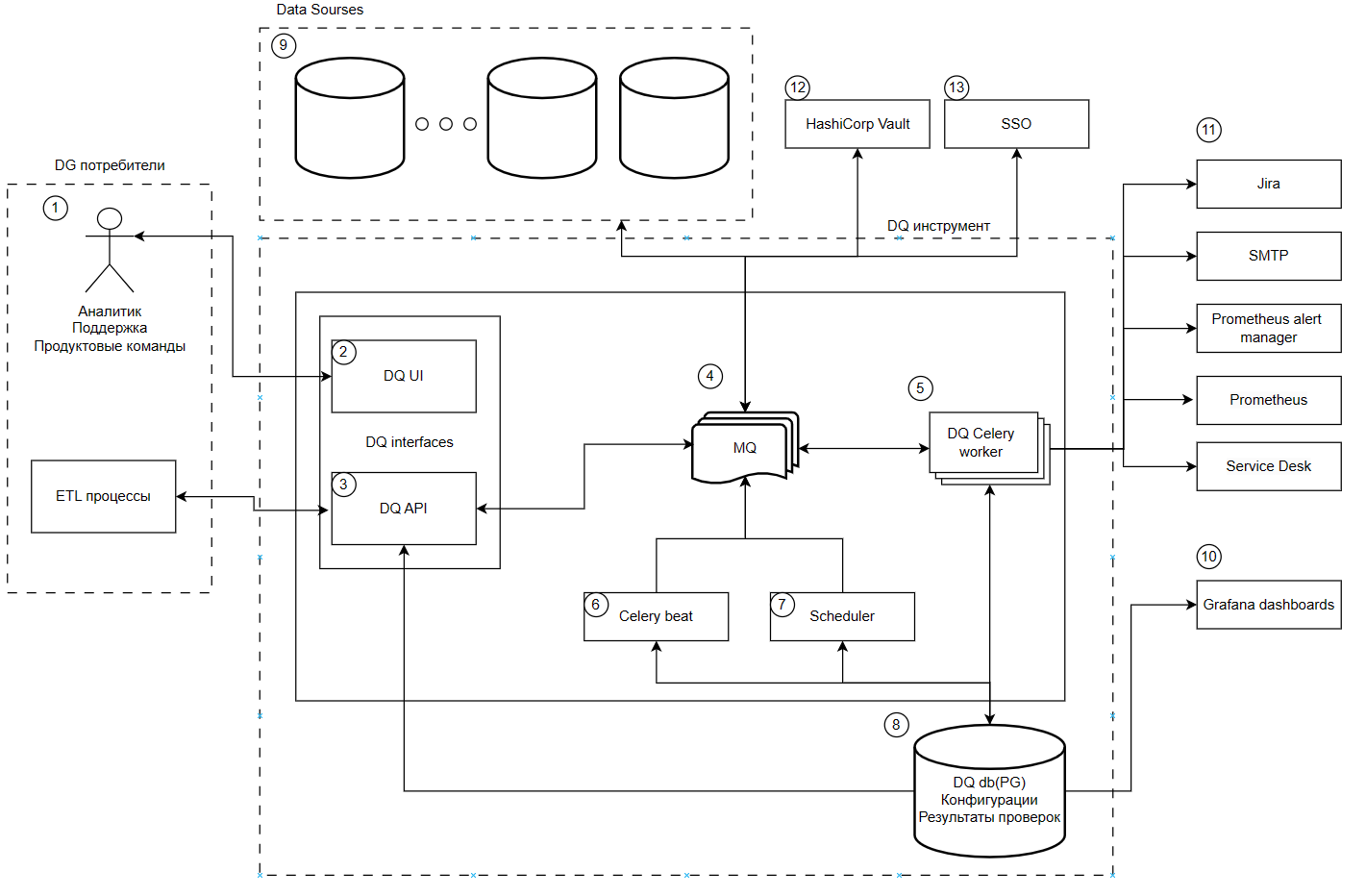
* Больше данных: более современные технологии сбора и хранения данных в банковском секторе приводят к тому, что компании работают с большим количеством данных, чем когда-либо раньше. Это увеличивает значение Data Quality, так как плохие данные могут привести к ошибкам в отчетности и принятии решений.
* Регуляторное давление: Регуляторные органы в банковской сфере все более строго регулируют область управления данными. Банки должны удостовериться, что данные, которые они используют для регуляторной отчетности, являются точными и правильными.
* Уменьшение издержек: Плохие данные могут стать причиной значительных издержек для банков. Ошибки в данных могут привести к потере денег, штрафам за нарушение регуляторных требований и проблемам с клиентами, что в конечном итоге может негативно сказаться на репутации банка и его финансовых результатов.
* В связи с этим банки все больше обращают внимание на Data Quality и вкладывают в эту область обширные ресурсы.

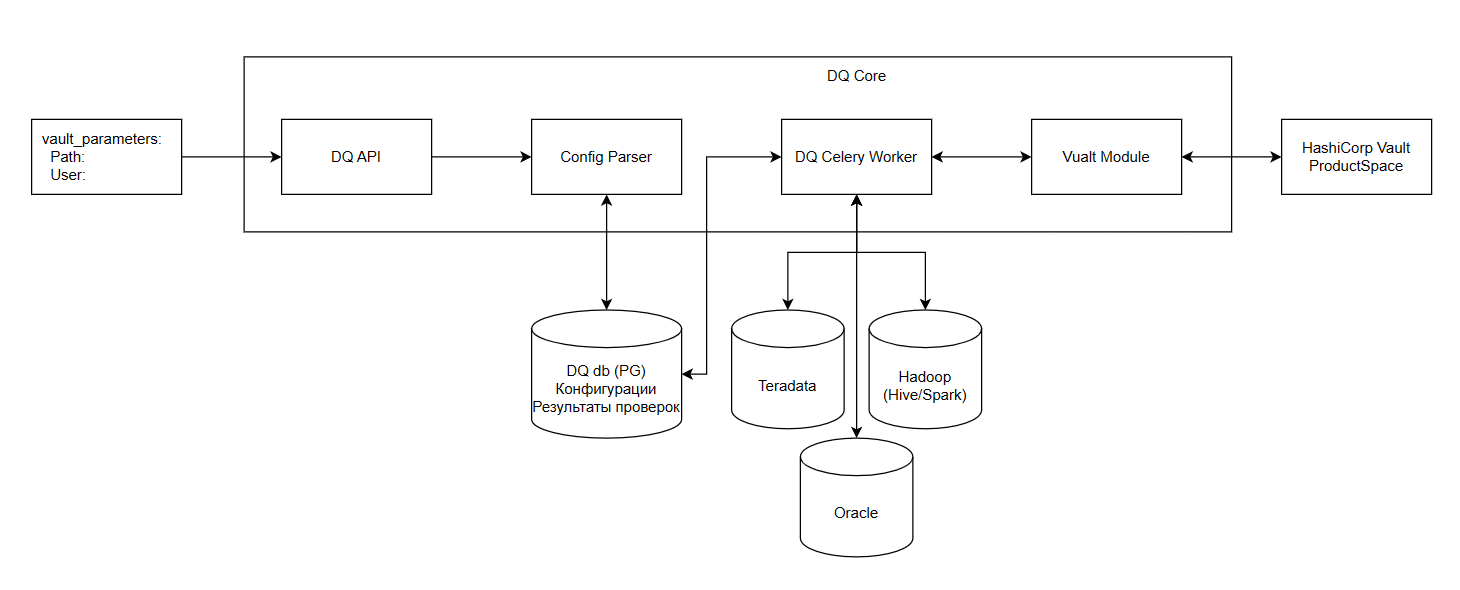
Качество данных в больших компаниях может быть очень разным в зависимости от ряда факторов. Вот некоторые из них:

* Способность организации извлекать, передавать и обрабатывать данные.
* Способность организации хранить данные в централизованном месте, доступном всем необходимым лицам.
* Квалификация персонала, работающего с данными, и их дисциплина в выполнении задач, связанных с обработкой данных.
* Качество работы информационных систем, которые используются для сбора и обработки данных.
* Точность и временные характеристики самих данных, которые собираются и используются для анализа.
* Комплексность организационной системы и структуры, которые могут либо улучшить, либо ослабить качество данных.
* Уровень контроля и управления, которые существуют в организации в отношении доступа и использования данных.
* Бюджет, выделенный на развитие системы, связанной с данными, что позволяет улучшать и совершенствовать процессы.

# Модель инструмента в банковской сфер

## Архитектура





|  |  |  |
| --- | --- | --- |
| № | Элемент | Назначение |
| 1 | Потребители | Пользователи и автоматизированные конвейеры обработки данных (ETL процессы) |
| 2 | DQ UI | Web интерфейс для работы с метриками качества данных |
| 3 | DQ API | HTTP API для взаимодействия с пользователями и интеграции с другими системами |
| 4 | MQ | Rabbit MQ – внутренняя очередь сообщений, используемая для асинхронного запуска проверок на системах источниках, запусками проверок по расписанию.  Управления служебными задачами |
| 5 | DQ Celery Worker | Компонент, непосредственного выполняющий запуск расчета метрик на источниках данных, расчет проверок по результатами метрик, сохраняющий результат в БД DQ |
| 6 | Celery beat | Компонент, отвечающий за запуск служебных задач ( отправка алертов, отправка флагов в Каталог данных, мониторинг задач во внутренней очереди DQ |
| 7 | Scheduler | Создает задачи на расчет метрик, запускаемых по crontab-like расписанию |
| 8 | DQ DB | Внутренняя PostgreSQL база, содержит все метаданные и результаты DQ |
| 9 | Data Sources | Источники данных (СУБД, NoSQL), на которых рассчитываются метрики качества |
| 10 | Grafana | Набор дашбордов в Grafana, используемый для визуализации метаданных и результатов метрик и проверок DQ |
| 11 | Система мониторинга алертинга | Набор сервисов, используемых для нотификаций об инцидентах качества данных (алерты) |
| 12 | HashiCorp Vault | Система Хранения crecentals для подключения к источникам данных |

## Функциональные необходимость.

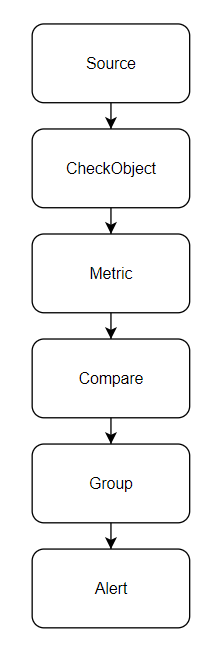
Что должен уметь делать DQ продукт:

* Централизованное хранение информации о качестве данных во всех ИС компании.
* Подключение к различным реляционным и NoSQL источникам данных.
* Стандартный набор метрик с возможностью создания кастомных.
* Продвинутый программный API с возможностью оптимизации проверок и code-first настройки.
* Web интерфейс для создания и запуска проверок.
* Визуализация результатов метрик и проверок на дашбордах BI инструмента.
* Оповещение об инцидентах:
  + Через почту;
  + Интеграция с JIRA;
  + Интеграция с Инцидент инструментом;

Типовой кейс использования продукта:

1. DQ понадобится для проактивного отслеживания ошибок в данных, т.е. для того, как эти ошибки станут заметны пользователям и приведут к финансовым или репутационным потерям.
2. Для реализации проверок над данными необходимо подключить свою систему к DQ.
3. Пользователь настраивает метрики и проверки с использованием внутреннего синтаксиса в формате YAML.
4. Запуск проверок осуществляется через Web UI или один из программных интерфейсов (REST, python).
5. Настройка запуска по расписанию или интегрировать запуск по тригеру в процессе обработки данных.
6. При отклонении проверки от ожидаемого результата, приходит оповещение по выбранному каналу отправки.
7. В оповещении содержится информация о неуспешную проверку и ссылка на дашборд для детального разбора проблемы.

## Объектная модель DQ



**Source**

Описывает параметры и способ подключения к «источнику данных» - инстансу Hadoop кластера(через Spark или Hive) или СУБД. Содержит параметры необходимые для подключения к источнику в том числе username и путь к паролю и HashiCorp Vaulе. Для Hadoop источников каждая команда должна будет создавать свой отдельный Source co своим уникальным пользователем

**CheckObject**

Описывает объект проверки, т.е. какую-то таблицу в Source, для которой будут настраиваться метрики. Описывание детализируется до уровня имени таблицы

**Metric**

Метрика представляет собой результат выполнения некоторой трансформации на объекте проверки. Как правило результатом проверки является какой-то агрегат (count, sum, avg) или распределение (гистограмма) непосредственно на источнике данных через SQL или SparkЫЙД

Расчет метрик происходит непосредственно на источнике данных через SQL или SparkSQL запросы. Значение метрики сохраняется в БД DQ для дальнейшей проверки через механизм compare

**Compare**

Представляет собой правило проверки одной или двух метрик с ожидаемым значением.

Ожидаемое значение может быть фиксированным, например, значение метрики всегда должно быть 0, так и динамичным, например, значение метрики должно отличаться от значения, полученного за предыдущий день не более, чем на 5%

**Group**

Группа объединяет в себе одну или несколько проверок. Группы используются для запуска DQ визуализации результатов в BI инструменте и для создания алертов. Как правило группа содержит в себе проверки по какой-то логической или физической сущности в источнике данных.

**Alert**

Описывает содержимое, правило срабатывания и канал отправки алерта (email, alert manager, jira, Prometheus). Описывание алерта может также содержать служебные поля, необходимые для отправки через тот или иной канал.

## Структура YAML конфига

Конфиг включает в себя такие сущности как source, check\_objects, metrics, compares, groups и alerts)

Рассмотрим простой кейс проверки данных и пример конфигурации

Задача проверить на кластере cloud-329, таблицу simple\_data\_test. Если в колонке ‘user\_uuid’ появляется значение равное NULL отправлять оповещение о проблемах с качеством данных

sources:  
 - type: spark  
 name: spark\_cloud\_329  
 parametres:  
 spark\_conf\_paramenres:  
 spark.yarn.queue: "example\_queue" *# Необходимо указать очередб команды из yarn* vault\_parametres:  
 path: "dq/spark\_cloud\_329"  
 user: "username"  
 cluster: "cloud\_329"  
 owners: 'savitskiyov'  
  
check\_objects:  
 - name: spark\_simple\_data\_test  
 source: spark\_cloud\_329  
 database: dqfw\_test  
 table: simple\_data\_test  
  
metrics:  
 - name: spark\_msisdn\_count\_value\_simple\_data  
 type: row\_count\_null  
 check\_object: spark\_simple\_data\_test  
 parameters:  
 column: "user\_uuid"  
  
compares:  
 - name: spark\_compare\_with\_statistic\_values\_simple\_data  
 type: compare\_with\_statistic\_values  
 parametres:  
 min\_value: 0  
 max\_value: 0  
 metric: spark\_msisdn\_count\_value\_simple\_data  
 description: "заполнения атрибута user\_uuid"  
  
groups:  
 - name: test\_check  
 compares:  
 - spark\_compare\_with\_statistic\_values\_simple\_data  
  
alerts:  
 - name: test\_check\_alert  
 annotations:  
 description: "Проверка на {description}"  
 labels:  
 project: 'DQFW'  
 group: ['test\_group']  
 fields: ['column', 'description']

## Источник данных. SOURCES

Блок sources содержит описание источников данных. Каждый источник имеет следующие поля:

* Type (строка) – тип источника. Указываем БД, пример (hive, mysql, oracle, postgres и т.д.).
* Name (строка) – уникальное наименование источника.
* Tags (список строк) – список тэгов(меток) для источника(опционально).
* Parametres (словарь) – параметры для подключения к источнику, параметрами Vault (хранение паролей), параметры для настройки ресурсов для spark-задач и т.д.
* Owners(список строк) – список owners или групп.
* Emails\_for\_failed\_task\_alert (список почтовых адресов) – список почтовых адресов для уведомлений о падении запуска проверки.
* Jira\_project – название проекта в jira, в котором будет создана задача при срабатывании алерта.

Если необходимо добавить несколько источников в поле sources передается список словарей

sources:

- type: spark

name: spark\_cloud\_329

parametres:

spark\_conf\_paramenres:

spark.executor.memory: "1g"

spark.driver.memory: "1g"

spark.executor.instances: "1"

spark.yarn.queue: "example\_queue" *# Необходимо указать очередб команды из yarn*

vault\_parametres:

path: "dq/spark\_cloud\_329"

user: "username"

cluster: "cloud\_329"

owners: 'savitskiyov'

product: DQ

- type: spark

name: spark\_cloud\_329\_2

jira\_project: 'DQP'

parametres:

spark\_conf\_paramenres:

spark.executor.memory: "2g"

spark.driver.memory: "2g"

spark.executor.instances: "1"

spark.yarn.queue: "example\_queue" *# Необходимо указать очередб команды из yarn*

vault\_parametres:

path: "dq/spark\_cloud\_329"

user: "username2"

cluster: "cloud\_329"

owners: ['savitskiyov', 'Big Data']

## Параметры источников

1. Параметры для аутентификации на источнике (запуск по УЗ команды)

Все источники содержат vault\_parametres, с помощью него DQ получает пароль от учетной записи пользователей из Vault.

* vault\_parametres (словарь)
  + path (строка) – путь где храниться пароль в Vault
  + user (строка) -имя учетной записи

В самом Vault, внутри path должен быть создан key:value

* key – это имя пользователя (должно совпадать с user из конфига)
* value -это пароль

2. Параметры spark-источника

* spark\_conf\_parameters (словарь) – Содержит параметры для конфигурации spark сессии, такие как spark.executor.memory, spark.executor.instances и т.д. в поле parameters необходимо передать словарь spark\_conf\_parameters
* cluster (строка|Enum) – Название кластера
* executor\_threads (число, по-умолчанию = 1) – кол-во потоков(параллельно выполняемых jobs), на spark executors. Если задано больше 1, spark jobs будут выполняться одновременно, в результате ресурсы кластера могут использоваться более эффективно

При увеличении числа потоков также может потребоваться поднять память выделяемую на executors.

parametres:

spark\_conf\_paramenres:

spark.executor.memory: "2g"

vault\_parametres:

path: "dq/spark\_cloud\_329"

user: "username"

cluster: "cloud\_329"

2.1. Переопределение таймзоны в spark сессии.

По умолчанию spark сессия будет создана с таймзоной UTC.

Таймзону имеет смысл переопределить, если данные в таблице с типом timestamp хранятся с учетом другого часового пояса, например Europe/Moscow.

Пример:

spark\_conf\_paramenres:

spark.executor.memory: "1g"

spark.driver.memory: "1g"

spark.executor.instances: "1"

spark.sql.session.timezone: "Europe/Moscow"

spark.yarn.am.extraJavaOptions: "-Ddhp.version=2.6.5.0-292 -Duser.timezone=Europe/Moscow"

spark.driver.extraJavaOptions: "-Ddhp.version=2.6.5.0-292 -Duser.timezone=Europe/Moscow"

2.2. Допустимые значения параметра cluster:

Поддерживаемые кластера Hadoop, на которых можно собирать метрики DQ

3. Параметры Hive источник

* user (строка) – имя пользователя в Hive
* hosts (строка) – имя name node c HiveServer
* zookeeper (словарь) – Переопределяет hosts
  + hosts (строка) – адрес zookeeper
  + service (строка) – имя сервиса в zookeeper
* configuration (словарь) – Содержит параметры конфигурации для Hive, которая будет применена при подключении
* Kerberos\_service\_name (строка) – имя principal (указывается только для кластеров с Kerberos)
* Database(строка) -База данных, по умолчанию default

parametres:

vault\_parametres:

path: "dq/spark\_cloud\_329"

user: "username"

kerberos\_service\_name: "hive"

zookeeper:

hosts: "cloud\_329-nn-001"

service: "hiveservice2"

configuration:

еуя.queue.name: "dq"

4. Параметры Oracle источника

* hosts (строка) – имя сервера с БД
* port (строка) – порт на котором запущен
* sid (строка) – oracle system identifier (имя инстанса oracle)
* service\_name (строка) – oracle service name

Для oracle нужно обязательно указать один из параметров sid или service\_name.

Пример Oracle

parametres:

vault\_parametres:

path: "dq/spark\_cloud\_329"

user: "username"

host: "localhost"

port: "151210"

sid: "ORCLCDB.localdomain"

5. Параметры Teradata, Postgres (Greenplum), MySQL, MSSQL источника

* hosts (строка) – имя сервера с БД
* port (строка) – порт на котором запущен
* database (строка) – имя БД
* parametres:  
   vault\_parametres:  
   path: "dq/spark\_cloud\_329"  
   user: "username"  
   host: "localhost"  
   port: "151210"  
   database: "dq\_test"

## Объекты проверок. CHECK\_OBJECTS

check\_objects:  
 - name: test\_objects1  
 source: spark\_source  
 database: dqfw\_test  
 table: simple\_data\_test  
 - name: test\_objects2  
 source: spark\_source  
 database: dqfw\_test  
 table: simple\_data\_test\_2  
 - name: test\_objects3  
 source: spark\_source  
 database: dqfw\_test  
 table: simple\_data\_test)3  
 active: False

Поля check\_objects:

* name (строка) – Уникальное имя объекта
* source (строка) – Название источника данных из SOURCES
* database (строка) – Название исходной базы данных(опциональный атрибут)
* schema (строка) – Название схемы (опциональный атрибут)
* table (строка) - Название исходной таблицы
* active (булево) – Индикатор активности, по умолчанию TRUE. Если FALSE – не будут, запускаться метрики и сравнения над этим объектом
* tags (список строк) – список тегов (меток)

## Метрики. METRICS

Поля metrics

* name (строка) – Название метрики
* type (строка) – Тип метрики ( есть предопределенные типы метрик)
* check\_object (строка) – Имя объекта проверки из check\_objects
* check\_object\_link (строка) – используется для создания метрик над метриками
* parametres (строка) – Параметры метрик
* run (словарь) – Параметры для запуска метрики по расписанию
* active (булево) – Индикатор активности, по умолчанию TRUE. Нужен чтобы отключить конкретную метрику и сравнение над этой метрикой
* tags (список строк) – список тегов (меток)

Пример конфига с метриками

metrics:  
 - name: spark\_msisdn\_count\_value\_more\_than\_1  
 type: row\_count  
 check\_object: spark\_simple\_data\_test  
 parametres:  
 column: "msisdn"  
 where: "msisdn is not null and value > 1"  
 tags:  
 - handover\_to\_dq  
 - name: spark\_msisdn\_count\_value\_simple\_data  
 type: custom\_sql  
 check\_object: spark\_simple\_data\_test  
 parametres:  
 column: "value"  
 sql: "select count(\*) from test.spark\_simple\_data where {column} = 0"  
 tags:  
 - zeros\_count  
 - on\_dq\_monitoring  
 - name: spark\_tottal\_msisdn\_count\_distinct  
 type: row\_count\_distinct  
 check\_object: spark\_simple\_data\_test  
 parametres:  
 column: "msisdn"

1. Типы предполагаемых метрик

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Описание | Параметры | Обязательный |
| Row\_count | Подсчет количества строк  Select count(<column>) …  <column> может принимать значение \* | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Row\_count\_distinct | Подсчет количества уникальных строк:  Select count(distinct <column>) … | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Row\_count\_null | Подсчет количества NULL  Select count(case when <column> is NULL then 1 end) … | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Row\_count\_null\_or\_empty | Подсчет количества NULL или с пустой строкой(«») в колонке  Select count(case when <column> is NULL or <column>=”” then 1 end) … | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Row\_count\_with\_null | Подсчет количества строк с учетом NULL значений  Select count(\*)  В отличие от обычногоrow\_count посчитаны все записи, включая те, где поле column имеет значение NULL | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Row\_count\_zero | Подсчет количества 0:  Select count(case when <column> = 0 then 1 end) … | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| avg | Подсчет среднего значения в колонке  Select avg(<column>) .. | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| sum | Подсчет суммы всех значений в колонке  Select sum(<column>) .. | Column | true |
| where |  |
| Date\_column\_name |  |
| Group\_by |  |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Custom\_sql | Метрика с кастомным SQL-запросом | Column |  |
| sql | true |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| Custom\_python | Метрика выполняющая любую python функцию  Функция должны иметь main и поддерживать следующий интерфейс def main(spark\_session, \*\*kwargs) -> Union [str, Dict[str,str]] | code | true |
| Prev\_metric |  |
| Prev\_metric\_date\_reference |  |
| column |  |
| Иные методы необходимые для бизнеса | N-ое количество метрик |  |  |

* 1. Параметры метрик
* Column – имя колонки
* Where – условие фильтрации, передается как SQL только без операторов WHERE, например «msisdn is not null and value > 1»
* Date\_column\_name – имя колонки для фильтрации по дате, например «business\_dt»
  + Если указан этот параметр, к метрике автоматически примениться фильтр по дате запуска по указанному полю
* Group\_by – поле для группировки, например «name, region»
* Sql – SQL запрос
* Prev\_metric – имя другой метрики, результат которой будет использован при вычислении текущей метрики
* Prev\_metric\_date\_reference – дата выбора значений Prev\_metric (если не задана , будет использована таже дата , что и для подсчета текущей метрики)
  + PREV – предыдущий результат подсчета метрики
  + PREV\_SUCCESS – предыдущий успешный (положительный статус сравнения) результат подсчета метрики
  + D – определенное количество дней назад, например 1D, 2D или 10D
  + M – определенное количество месяцев назад, например 4М
  + ‘2023-08-01’ – эталонное значение будет выбрано за определенную дату
* Code – питоновский код, поддерживающий интерфейс def main (connection, \*\*kwargs). Типы, которые может возвращать функция str, dict, int. float. Пример конфиига

metrics:  
 - name: test\_metric\_custom\_python  
 type: custom\_python  
 check\_object: test\_obj\_custom\_python  
 parameters:  
 code: |+ *#чтобы учитывать перенос строки* def main(spark\_session, \*\*kwargs):  
 from pyspark.sql.functions import col, avg  
 from pyspark.sql import row  
 df = spark\_session.createDataFrame([Row(a=1, b=1, c=1), Row(a=2, b=2, c=2)])  
 res = df.select(avg(col('b'))).collect()[0][0]  
 return str(res)

* + В kwargs передаются следующие аргументы:
    - Check\_date – дата запуска проверки (значение параметра date в методах запуска)
    - Column – значение параметра column из метрики
    - Prev\_metric – результат метрики из параметра prev\_metric(если он указан)
    - Prev\_metric\_date\_reference – значение параметра Prev\_metric\_date\_reference из метрик
    - Database – значение атрибута check\_object.database
    - Schema -значение атрибута check\_object.schema
    - Table – значение атрибута check\_object.table

В параметрах where и sql можно использовать следующие плейсхолдеры:

* Database – атрибут объекта проверки
* Schema – атрибут объекта проверки
* Table -атрибут объекта проверки
* Column -параметр проверки
* Current\_date -дата проверки
* Current\_date\_no\_sep – дата проверки в формате YYYYmmdd
* Current\_day – день из даты проверки
* Current\_month – месяц из даты проверки
* Current\_year – год из даты проверки
* Prev\_metric – результат метрики из параметра prev\_metric, посчитанный за дату prev\_metric\_date\_reference

В параметре aggregate\_function можно использовать плейсхолдер column

Пример использования плейсхолеров:

metrics:  
 - name: PG\_cloud-329.test.standart\_compare.custom\_sql  
 type: custom\_sql  
 check\_object : PG\_cloud-329.test.standart\_compare  
 parametres:  
 column: "rel\_value"  
 sql: "SELECT COUNT({column}) FROM {schema}.{table} WHERE business\_dt = {current\_date}"  
 *# Запрос к БД: SELECT COUNT(rel\_value) FROM test.stantard\_compare WHERE business\_dt = '2023-08-01'* - name: PG\_cloud-329.test.standart\_compare.row\_count  
 type: row\_count  
 check\_object: PG\_cloud-329.test.standart\_compare  
 parametres:  
 column: "rel\_value"  
 where: "business\_dt = '{current\_year}/{current\_month}/{current\_date}'"  
 *# Запрос к БД: SELECT COUNT(rel\_value) FROM test.stantard\_compare WHERE business\_dt = '2023/08/01'* - name: PG\_cloud-329.test.standart\_compare.row\_count2  
 type: row\_count  
 check\_object: PG\_cloud-329.test.standart\_compare  
 parametres:  
 column: "rel\_value"  
 where: "business\_dt = '{current\_date\_no\_sep}'"  
 *# Запрос к БД: SELECT COUNT(rel\_value) FROM test.stantard\_compare WHERE business\_dt = '20230801'*

## Сравнение. COMPARES

Поле compares

* Name (строка) – Название сравнения
* Type (строка) – Тип сравнения
* Metric (строка) – Имя метрики
* Parametres (словарь) – Параметры сравнения
* Run (словарь) – Параметры для запуска сравнения по расписанию
* Description (строка) – Описание проверки в свободной форме(используется для алертов)
* Active (булево) – Индикатор активности сравнения, по умолчанию True, нужен чтобы отключить конкретное сравнение и привязанную к нему метрику
* Tags (список строк) – список тэгов(меток) для compare(опционально)

Примеры конфига

compares:  
 - name: compare\_two\_metric\_absolute  
 type: absolute\_ratio  
 parametres:  
 min\_value: 100  
 max\_value: 110 *# Бесконечность можно выставить при помощи слова 'inf'* reference\_metric: spark\_msisdn\_count\_value\_more\_than\_1  
 metric: spark\_msisdn\_count\_value\_simple\_data  
 description: "Сравнение значения метрки spark\_msisdn\_count\_value\_simple\_data с некторым числом"  
 - name: compare\_two\_metric\_delta  
 type: percent\_delta  
 parametres:  
 min\_value: 110  
 max\_value: 120 *# Бесконечность можно выставить при помощи слова 'inf'* date\_reference: 'PREV'  
 reference\_metric: spark\_msisdn\_count\_value\_simple\_data  
 metric: spark\_msisdn\_count\_value\_more\_than\_1  
 - name: spark\_compare\_with\_static\_values  
 type: compare\_with\_static\_values  
 parametres:  
 min\_value: 90  
 max\_value: 110  
 metric: spark\_msisdn\_count\_value\_more\_than\_1

1. Типы сравнений

В описании сравнений ниже:

Metric\_1 – это metric в конфигурации

Metric\_2 – это reference\_metric в конфигурации

Min\_value и max\_value указываются включаться в интервал допустимых значений.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Описание | Параметры | Обязательный |
| Compare\_with\_static\_values | Сравнение метрики с эталонным значением | min\_value | true |
| max\_value | true |
| grouped\_field\_value |  |
| absolute\_ratio | Сравнение разницы между актуальным значением (metric\_1 - metric\_2) и min\_value true  статическим значением | min\_value | true |
| max\_value | true |
| grouped\_field\_value |  |
| date\_reference |  |
| reference\_metric |  |
| full\_diff |  |
| percent\_delta | сравнение разницы в процентах между актуальным значением  compare\_result = (metric\_1/metric\_2\*100)  Если metric\_1 == metric\_2  compare\_result = 100  Если metric\_2 == 0  compare\_result = +infinity | min\_value | true |
| max\_value | true |
| grouped\_field\_value |  |
| date\_reference |  |
| reference\_metric |  |
| full\_diff |  |
| percent\_ratio | сравнение процентного соотношения между актуальным значением  compare\_result = ((metric\_1-metric\_2)/ metric \_2\*100)  Если metric\_1 == metric\_2  compare\_result = 0  Если metric\_2 == 0  compare\_result = +infinity | min\_value | true |
| max\_value | true |
| grouped\_field\_value |  |
| date\_reference |  |
| reference\_metric |  |
| full\_diff |  |
| histogram\_compare | Сравнение гистограмм (SUM(ABS(metric\_1 – metric\_2))/2) | min\_value | true |
| max\_value | true |
| date\_reference |  |
| reference\_metric |  |
| Histogram\_psi\_compare | Вычисление PSI по двум метрикам | min\_value | true |
| max\_value | true |
| date\_reference |  |
| reference\_metric |  |
| Histogram\_ratio\_compare | Сравнение гистограмм SUM(ABS(metric\_1) – metric\_2/SUM(metric\_2))/2 |  | true |
| max\_value | true |
| date\_reference |  |
| reference\_metric |  |

1.1. Параметры сравнений

* Min\_value (число) – минимальное пороговое значение
* Max\_value (число) – максимальное пороговое значение
* Grouped\_field\_value (строка) – значение поля, по которому была группировка при подсчете метрик с параметром group\_by
* Date\_reference (строка) -параметр для выбора эталонного значения
  + PREV – предыдущий результат подсчета метрики
  + PREV\_SUCCESS – предыдущий успешный (положительный статус сравнения) результат подсчета метрики
  + D – определенное количество дней назад, например 1D, 2D или 10D
  + M – определенное количество месяцев назад, например 4М
  + ‘2023-08-01’ – эталонное значение будет выбрано за определенную дату
* Reference\_metric (строка) – имя метрики с которой происходит сравнение
  + Параметр опциональный, по умолчанию сравнение идет с метрикой из атрибута metric
  + Если параметр задан и при этом не задан date\_reference метрики будут сравниваться за одну дату (дату запуска)
* Full\_diff (true\false) – актуален при сравнении метрик с группировкой (group by или custom\_sql), т.е. результат которых представляет из себя набор пар key-value
  + False (по умолчанию) – метрики будут сравниваться по принципу left join, где left метрика это результат метрики атрибута metric, right – из параметра reference\_metric
  + True – метрики будут сравниваться по принципу full join

1.2. Сравнение метрик с группировкой

Результат метрики может быть 3 типов:

1. скаляр (число), например, значение кол-во записей – результат count()
2. распределение – результат метрик типа histogram\_\*
3. словарь – набор key/value пар, например count() в разрезе по каким-то полям (если в метрике указан параметр group\_by или результат custom\_sql,custom\_python метрик)

Рассмотрим способы проверки метрик 3-го типа.

Например, нам нужно получить список жанров фильмов и кол-во записей по каждому жанру

По каждому жанру нужно убедиться, что кол-во фильмов по сравнению со вчерашним числом расходится не более, чем на 90%. Также мы хотим видеть, какие жанры были вчера и какие новые появились сегодня.

Наша метрика

metrics:  
 - name: count\_by\_genre  
 type: row\_count  
 check\_object: films\_table  
 parameters:  
 column: "\*"  
 date\_column\_name: business\_dt  
 group\_by: genre

Результат может выглядеть следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| Вчера | | Сегодня | |
| Key | value | Key | value |
| Action | 100 |  |  |
| Comedy | 180 | Comedy | 200 |
| horror | 160 | horror | 200 |
|  |  | triller | 299 |

Параметры compare для нашей метрики:

compare:  
 - name: count\_by\_genre\_compare  
 type: percent\_delta  
 parameters:  
 min\_value: 90  
 max\_value: 100  
 reference\_metric: count\_by\_genre  
 date\_reference: 1D  
 full\_diff: "false" *# false - значение по умолчанию* metric: count\_by\_genre  
 description: "Сравнение жанров за вчера/сегодня"

В зависимости от параметра full\_diff (true/false) будут получены следующие результаты compare:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вчера | | Сегодня | | compare.full\_diff = false | | compare.full\_diff = true | |
| key | value | key | value | compare\_value | status | compare\_value | status |
| action | 100 |  |  | None | 0 (NO REFERENCE) | None | 0 (NO REFERENCE) |
| comedy | 180 | comedy | 200 | 90 | 1 (OK) | 90 | 1 (OK) |
| horror | 160 | horror | 200 | 80 | -1 (NOT OK) | 80 | -1 (NOT OK) |
|  |  | triller | 299 |  |  | None | -2 (NO METRIC VALUE) |

Пороговое значение min\_value/max\_value применяется к каждой записи из метрики.

Если мы хотим задать частные пороги для каких = то жанров - нужно создать отдельный compare с параметром grouped\_field\_value.

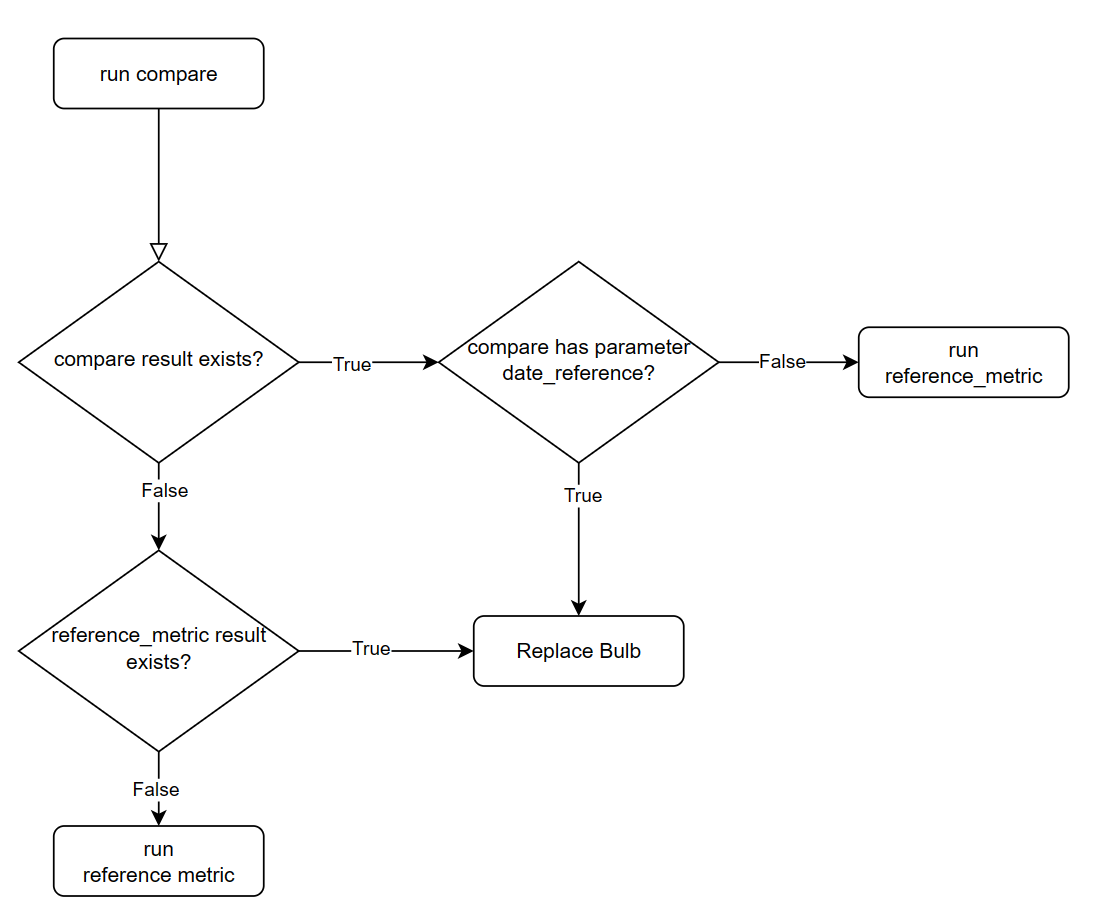
compares:  
 - name: count\_by\_genre\_compare\_comedy  
 type: percent\_delta  
 parameters:  
 min\_value: 80  
 max\_value: 100  
 reference\_metric: count\_by\_genre  
 date\_reference: 1D  
 group\_field\_value: horror  
 metric: count\_by\_genre

Результат нового compare:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вчера | | Сегодня | | compare | |
| key | value | key | value | compare\_value | status |
| action | 100 |  |  |  |  |
| comedy | 180 | comedy | 200 |  |  |
| horror | 160 | horror | 200 | 80 | 1 (OK) |
|  |  | triller | 299 |  |  |

1.3 Алгоритм запуска метрики из параметра reference\_metric у сравнения

Если в compare задач параметр reference\_metric, сама метрика reference\_metric будет посчитана по следующему алгоритму



## Группы. Groups

Конфиг для создания групп проверок включает такие поля как:

* Name (строка) – уникальное имя группы
* Compares (список) – список с именами сравнений
* Metrics (список) – список с именами метрик, опциональный. Добавляется в случае, если нужно запускать метрики, которые не привязаны к определенным сравнениям. Частый кейс — это метрика из поля reference\_metric, в сущности, compares

compares:  
 - name: compare\_two\_metrics\_absolute  
 type: absolute\_ratio  
 parameters:  
 min\_value: 100  
 max\_value: 110  
 reference\_metric: metric\_two  
 metric: metric\_one

* Active (булево) – индикатор активности группы, по умолчанию True. Нужен чтобы выключить целиком всю группу проверок
* Run (словарь) -параметры для запуска группы по расписанию
* Tags (список строк) – список тэгов(меток) для группы(опционально)
* Source\_parametres (список) – опциональный список с параметрами источников, дает возможность переопределить параметры для определенного источника. Представляет собой список словарей, где ключ — это имя источника, а значение параметр этого источника. Например:

source\_parametres:  
 - source\_name\_1:  
 spark\_conf\_parametres:  
 spark.executor.memory: "1g"  
 spark.driver.memory: "512m"  
 spark.executor.instances: "1"  
 - source\_name\_2:  
 spark\_conf\_parametres:  
 spark.executor.memory: "512m"  
 spark.driver.memory: "1g"  
 spark.executor.instances: "1"

Пример создания конфига:

groups:  
 - name: test\_spark  
 compares:  
 - compare\_two\_metrics\_absolute  
 - compare\_two\_metrics\_delta  
 - spark\_compare\_with\_static\_values\_simple\_data  
 - spark\_compare\_with\_static\_values  
 - spark\_compare\_percent\_delta  
 - spark\_compare\_percent\_delta\_two  
 - spark\_compare\_with\_percent\_ratio  
 - spark\_compare\_with\_absolute\_ratio  
 - spark\_compare\_with\_histogram  
 - spark\_compare\_with\_histogram\_ratio  
 - spark\_compare\_with\_histogram\_psi  
 metrics:  
 - spark\_metric\_histogram\_ratio  
 - spark\_metric\_histogram\_psi

## Параметры запуска по расписанию

Сущности metric, compare и group можно запускать по crontab расписанию с помощью встроенного в DQ планировщика

Если необходимо просто запустить проверки с некоторой периодичностью без интеграции процессами ETL, достаточно добавить в конфиг блок параметров run:

* Schedule(строка) – crontab like расписание, запуск будет выполняться по таймзоне МСК (например, ’00 07 \* \* \*’)
* Days\_data\_lag(число) (опционально) – на сколько дней в прошлом по бизнес дате необходимо запустить метрики/проверки. Кейс использования: Проверки запускаются ежедневно, но данные грузятся с опозданием в 3 дня. То есть сегодня будут загружены данные за current\_date – days\_data\_lag. Соответственно, если указать days\_data\_lag: 3, то проверки будут выполняться в источнике по дате = current\_date – days\_data\_lag
* Max\_retries (число) (опционально) – максимальное число попыток перезапуска (по умолчанию: 0). Механизм аналогичный retry в Airflow.
  + Проверки будут перезапущены, если хотя бы одна метрика «упала» с исключением
* Retry\_delay (число) (опционально) – интервал между перезапусками в секундах ( по умолчанию: 600 секунд)

Пример группы, которая будет запускаться по расписанию

groups:  
 - name: test\_spark  
 compares:  
 - compare\_two\_metrics\_absolute  
 - compare\_two\_metrics\_delta  
 - spark\_compare\_with\_static\_values\_simple\_data  
 - spark\_compare\_with\_static\_values  
 - spark\_compare\_percent\_delta  
 - spark\_compare\_percent\_delta\_two  
 - spark\_compare\_with\_percent\_ratio  
 - spark\_compare\_with\_absolute\_ratio  
 - spark\_compare\_with\_histogram  
 - spark\_compare\_with\_histogram\_ratio  
 - spark\_compare\_with\_histogram\_psi  
 metrics:  
 - spark\_metric\_histogram\_ratio  
 - spark\_metric\_histogram\_psi  
 run:  
 schedule: '0 3 \* \* \*' *# Запуск в 3 часа ночи по Москве, ежедневно* days\_data\_lag: 1

## Алерты. Alerts

Алерты о срабатывании проверок могут быть реализованы по одному из 4-х каналов отправки:

* 1. SMTP – письмо на почту для указанного списка адресатов
  2. Alert manager – письмо на почту, отправлено через сервис Alertmanager Prometheus.
  3. Jira – создание или обновление тикета в Jira
  4. Prometheus – создание метрики по алерту DQ. Позволяет интегрировать алерты с другими системами мониторинга и инцидент менеджмента

**1. Параметры alerts**

* Name (строка) – уникальное название алерта
* Fields (список) - опциональный список с именами полей, для отображения в алерте. По умолчанию отображает все поля
  + Compare\_date – дата сравнения
  + Compare\_name – имя сравнения
  + Value – значения метрики
  + Metric\_name – имя метрики
  + Check\_object – имя объекта проверки
  + Description – описание compare
  + Column –имя поля
  + Status – статус compare
  + Run\_id – id запуска
  + Log\_un – ссылка на кибану с логами запуск
  + Mertic\_key – значения ключа группировки из метрик с group by
* Groups (список) – группы проверок для которых формирует алерт
* Channels – список каналов, на которые осуществляется отправка сообщений.
  + Smtp – алерт будет отправлен через почту напрямую от DQ
  + Alert manager – алерт будет отправлен через почту через сервис Alert manager
  + Jira – по алерту будет создан jira тикет
  + Prometheus – по алерту будет создана запись с метрикой для Prometheus
* Annotations (словарь) – атрибут annotations в Alertmanager Prometheus. Имеет поля:
  + Description (строка) – описание алерта в свободной форме с параметрами в виде плейсхолдеров в фигурных скобках – {}, которые заменяются на значения из поля fields. Например, «Проверка {column} на {desceription}».
  + Summary (строка) -краткое описание алерта, также может содержать плейсхолдеры
* Labels (словарь) – можно передавать любое количество label’ов поддерживаемых Alertmanager Prometheus.
* Emails (список) – список получателей алерта
* Jira\_project – название проектов в jira, в котором будет создана задача при срабатывании алерта
* Jira\_options – параметры для тикета, указываются только для алертов с каналом jira
  + Project – название проекта в jira
  + Assignee – исполнитель тикера
  + Issue\_type – Тикет будет создан с указанным типом (Bug, Task) По умолчанию тикет создается с типом Bug
  + Components – Массив компонентов тикета
* Active (булево) – Индикатор активности, по умолчанию True. Если False – не будут отправляться алерты.
* Tags (список строк) – список тэгов (меток) для алерта (опционально)
* Statuses (список) – Опциональный список статусов Compare, по которому формируется условия срабатывания алертов. По умолчанию это статусы -1, -2. Варианты статусов:

|  |  |
| --- | --- |
| 0 | Нет значения метрики, относительно которой происходит сравнение (reference\_metric) |
| 1 | Проверка прошла успешно |
| -1 | Проверка прошла не успешно (несоответствие пороговым значениям из правила проверки) |
| -2 | Нет значения метрики |

**2. Настройка получателей alerts**

2.1. Для канала alert manager

alerts:  
 - name: TEST\_ALERT  
 annotations:  
 description: "Проверка на {description} по {check\_object} на {compare\_date} завершилась неудачно"  
 labels:  
 project: '<Название проекта в Prometheus Alertmanager>'  
 fields: ['column', 'description', 'status', 'compare\_date', 'check\_object']  
 group: ['pg\_test\_data\_group']  
 statuses: [-1, -2]  
 channels: ['alert manager']

2.2. Для канала smtp

alerts:  
 - name: TEST\_ALERT\_2  
 annotations:  
 description: "Проверка на {description} по {check\_object} на {compare\_date} завершилась неудачно"  
 emails: ['email@test.ru']  
 fields: ['column', 'description', 'status', 'compare\_date', 'check\_object']  
 group: ['pg\_test\_data\_group\_2']  
 statuses: [-1, -2]  
 channels: ['smtp']

2.3. Для канала jira

Тикет в jira будет создан с параметрами, которые можно указать в поле jira\_options.

jira options:  
 project: '<Название проекта в Jira>'  
 issue\_type: '<Тип тикета Bug, Task>'  
 assignee: '<исполнитель>'  
 components: ['component\_1']  
  
alerts:  
 - name: TEST\_ALERT\_3  
 annotations:  
 description: "Проверка на {description} по {check\_object} на {compare\_date} завершилась неудачно"  
 fields: [ 'column', 'description', 'status', 'compare\_date', 'check\_object' ]  
 group: [ 'pg\_test\_data\_group\_2' ]  
 statuses: [-1, -2]  
 jira\_options:  
 project: 'DA'  
 issue\_type: 'Bug'  
 assignee: 'default\_asignee'  
 components: ['DQ alerts']  
 channels: ['jira']

2.4. Для канала Prometheus

По группам проверок из настроек алерта будут созданы метрики в Prometheus

Сам алерт нужно настраивать непосредственно в Prometheus alert manager.

alerts:  
 - name: TEST\_ALERT\_4  
 group: ['pg\_test\_data\_group\_4']  
 statuses: [-1, -2]  
 channels: ['prometheus']

## Механизм работы prev\_metric (Конвейер метрик)

В любую метрику можно добавить параметр prev\_metric.

Параметр должен содержать имя другой метрики, которая будет посчитана перед тем, как запустится текущая метрика.

Результат prev\_metric можно использовать внутри других параметров через макрос {prev\_metric} (where='business\_dt={prev\_metric}')

Это позволяет формировать линейную последовательность, в которой метрики будут запускаться и переиспользовать результаты выполнения на источнике, выстраивая таким образом "конвейер" метрик.

Пример:

Нужно посчитать 2 метрики по одной таблице:

1) количество записей за сутки с условием по полю input\_file\_name

2) количество записей за сутки с условием по полю input\_file\_name и дополнительным условием (where)

Условие по полю input\_file\_name одинаково в обоих метриках и преставляет из себя подзапрос:

select distinct input\_file\_name from raw.table1 where raw\_dt = {current\_date} order by input\_file\_name desc limit 1

В стандартном варианте конфигурации подзапрос будет дважды пересчитан на источнике (по одному разу для каждой метрике)

metrics:  
- name: total\_row\_count\_daily\_check\_object\_1  
 type: custom\_sql  
 check\_object: raw.table\_1  
 parameters:  
 sql: "select count(\*) from raw.table\_1  
 where raw\_dt = '{current\_date}' and input\_file\_name = (  
 select distinct input\_file\_name   
 from raw.table\_1 where raw\_dt = '{current\_date}'   
 order by input\_file\_name desc limit 1)"  
- name: total\_row\_count\_daily\_with\_condition\_check\_object\_1  
 type: custom\_sql  
 check\_object: raw.table\_1  
 parameters:  
 sql: "select count(\*) from raw.table\_1  
 where raw\_dt = '{current\_date}' and input\_file\_name = (  
 select distinct input\_file\_name   
 from raw.table\_1 where raw\_dt = '{current\_date}'   
 order by input\_file\_name desc limit 1) and column1>column2"

Через механизм конвейера метрик, повторяющийся подзапрос можно объявить, как отдельную метрику, и переиспользовать ее результат при расчете других метрик. Вариант конфигурации через Prev\_metric:

metrics:  
- name: latest\_input\_file\_name.check\_object\_1  
 type: custom\_sql  
 check\_object: raw\_table\_1  
 parameters:  
 sql: "select distinct input\_file\_name  
 from raw.table\_1 where raw\_dt = '{current\_date}'  
 order by input\_file\_name desc limit 1"  
  
- name: total\_row\_count\_daily.check\_object\_1  
 type: row\_count  
 check\_object: raw.table\_1  
 parameters:  
 column: "\*"  
 where: "raw\_dt = '{current\_date}' and input\_file\_name = '{prev\_metric}'"  
 prev\_metric: latest\_input\_file\_name.check\_object\_1  
- name: total\_row\_count\_daily\_with\_condition.check\_object\_1  
 type: row\_count  
 check\_object: raw.table\_1  
 parameters:  
 column: "\*"  
 where: "raw\_dt = '{current\_date}' and input\_file\_name = '{prev\_metric}'and column1>column2"  
 prev\_metric: latest\_input\_file\_name.check\_object\_1

Параметр prev\_metric в этом примере содержит имя метрики, результат которой можно использовать через плейсхолдер '{prev\_metric}' в других параметрах. Подзапрос будет выполнен на источнике только один раз

Особенности:

1. Дополнительно можно использовать параметр prev\_metric\_date\_reference – дата, за которую будет выбран результат {prev\_metric}. Формат параметра идентичен параметру data\_reference в Compare
2. Если параметр prev\_metric\_date\_reference не указан {prev\_metric} будет заменен результатом метрики за текущую дату проверки
3. Если за дату из prev\_metric\_date\_reference для {prev\_metric} нет результата в базе DQ – метрика из “prev\_metric” будет запущена перед подсчетом основной метрики

## Метрики нал результатами из базы DQ (метрики второго порядка)

DQ представляет 2 специальных объекта (check\_object) с результатами метрик и проверок. Объекты представляют собой view’s над несколькими таблицами в базе DQ

Этими объектами можно воспользоваться, как и любыми другими, если необходимо переиспользовать результаты метрик или проверок из базы DQ в других метриках

**Результаты метрик (metric\_result\_view)**

Создание метрик, охватывающих большой временной интервал, не создавая при этом дополнительную нагрузку на источник.

Используя метрики 2-го порядка, можно реализовать такие бизнес проверки как:

- среднее за определенный диапазон

- скользящее среднее за определенный диапазон

- минимальные/максимальные значения из истории

**Как использовать**

metric\_result\_view имеет следующие поля:

* metric\_dt – дата результата
* metric\_name - имя метрики
* metric\_type - тип метрики
* value – результат подсчета метрики
* key - значение поля, по которому была группировка при подсчете метрик с параметром group\_by (для скалярных метрик - в этом поле будет Null)
* database - база, над которой считалась метрика
* schema - схема, над которой считалась метрика
* table - таблица, над которой считалась метрика

Чтобы создать метрику, над вьюхой metric\_result\_view, в конфиге нужно будет задать атрибут:

* check\_object\_link: metric\_result\_view
* атрибут check\_object также остается обязательным и будет отображаться на дашборде в графане как объект метрики

Примеры:

metrics:  
 - name: count\_results  
 type: row\_count  
 check\_object: spark\_simple\_data\_test  
 check\_object\_link: metric\_result\_view  
 parameters:  
 column: "\*"  
 where: "metric\_name='msisdn\_count\_spark'"

metrics:  
 - name: count\_average\_per\_month  
 type: custom\_sql  
 check\_object: spark\_simple\_data\_test  
 check\_object\_link: metric\_result\_view  
 parameters:  
 prev\_metric: spark\_msisdn\_count\_value\_simple\_data  
 sql: "select avg(cast(value is int)) from {table}  
 where matric\_name = 'spark\_msisdn\_count\_value\_simple\_data'  
 and metric\_dt between '2023-10-01' and '2023-11-01'"

## Результаты проверок (compare\_result\_view)

Compare\_result\_view содержит статусы проверок и связанные с ними значения метрик.

Объект можно использовать для реализации гибких правил алертинга.

Например, у нас есть проверка на количество записей в разрезе поля регион (метрика с группировкой).

Если регионов много, скажем 100, и все они не прошли проверку, в алерте мы получим 100 записей с детальной информацией по каждому региону.

Можно создать еще одну метрику и проверку над ней, которая вернет количество ключей(регионов) из исходной метрики, которые не прошли проверку.

Таким образом при срабатывании алерты мы получим более сжатую информацию о кол-ве регионов, не прошедшие проверку.

**Как использовать**

Compare\_result\_view имеет следующие поля:

* compare\_dt – дата результата
* compare\_name – имя проверки
* compare\_type – тип проверки
* value – результат проверки (значение, полученной после выполнения функций compare, которое потом сравнивается с min\_value/max\_value порогами)
* status – статус проверки
* actual\_metric\_key – значений поля, по которому была группировка при подсчете метрик с параметром group\_by (для скалярных метрик, в этом поле будет NULL)
* actual\_metric\_value – результат подсчета метрики
* min\_value – значение параметра min\_value, актуальное на момент выполнения проверки
* max\_value – значение параметра max\_value, актуальное на момент выполнения проверки

Чтобы создать метрику над вьюхой compare\_result\_view, в конфиге необходимо задать атрибут:

* check\_object\_link: compare\_result\_view
* атрибут check\_object также остается обязательным и будет отображаться на дашборде в графане как объект метрики

Примеры:

metrics:  
 - name: count\_results  
 type: row\_count  
 check\_object: spark\_simple\_data\_test  
 check\_object\_link: metric\_result\_view  
 parameters:  
 column: "\*"  
 date\_column\_name: "compare\_dt"  
 where: "compare\_name='msisdn\_count\_spark.compare' and status < 0"

## Интеграция в ETL-процесс

Этапы работы DQ – запуск расчета и трансформаций и сравнений, а также получение статуса DQ метрик – могут быть встроены в ETL – процессы

Для этого в DQ должны быть реализованы следующие интерфейсные решения:

* Python модуль для интеграции в Airflow
* HTTP API сервис (REST)

## Ролевая модель

Кто может стать владельцем (owner):

Владельцы сущностей DQ (source, check, object, metric, compare, group) может быть как отдельный пользователь, так и целая группа

Что может владелец:

Только владелец может изменять и удалять принадлежащие ему сущности

Запуск метрик и проверок также доступен только их владельцу

Как стать владельцем:

Задавать владельцев можно при создании source в атрибуте owners. Все остальные сущности использующие этот source получают того же владельца и права на их использование будет только у него

У объекта может быть много владельцев: в атрибут owners можно передавать список учетных записей пользователей

Если владельцем является группа, то права на объекты получают все пользователи, которые состоят в этой группе

# Вывод

Data Quality играет важную роль в банковском секторе в условиях использования BIG DATA. Большие объемы данных, поступающих в банки, требуют систематической проверки и обеспечения качества данных, чтобы гарантировать точность и достоверность информации.

Одним из основных результатов работы является выявление основных проблем, связанных с качеством данных в банковском секторе. Это включает в себя ошибки ввода данных, дублирование информации, отсутствие актуальности данных и другие факторы, которые могут негативно сказаться на принятии решений и работе банка в целом.

Каждый банк должен выбирать инструмент, учитывая свои уникальные потребности и особенности. В связи с чем, все больше банков начинают писать свой собственный DQ Framework, а не пользоваться коробочными решениями

Однако, несмотря на значимость инструмента Data Quality, было выявлено, что не все банки активно используют его в своей работе. Это может быть связано с отсутствием осознания важности качества данных или с недостаточной информированностью о существующих инструментах.

В целом, использование инструмента Data Quality в банковском секторе в BIG DATA имеет большой потенциал для улучшения качества данных и повышения эффективности работы банка. Однако, для успешной реализации такого инструмента необходимо обеспечить подготовку персонала, обновление технической инфраструктуры и создание соответствующих процедур и политик внутри организации.

В заключение, рекомендуется банкам активно внедрять инструмент Data Quality в свою работу, чтобы обеспечить высокое качество данных и повысить свою конкурентоспособность в условиях использования BIG DATA. Это позволит банкам принимать более обоснованные решения, улучшать взаимодействие с клиентами и повышать уровень доверия со стороны общественности.